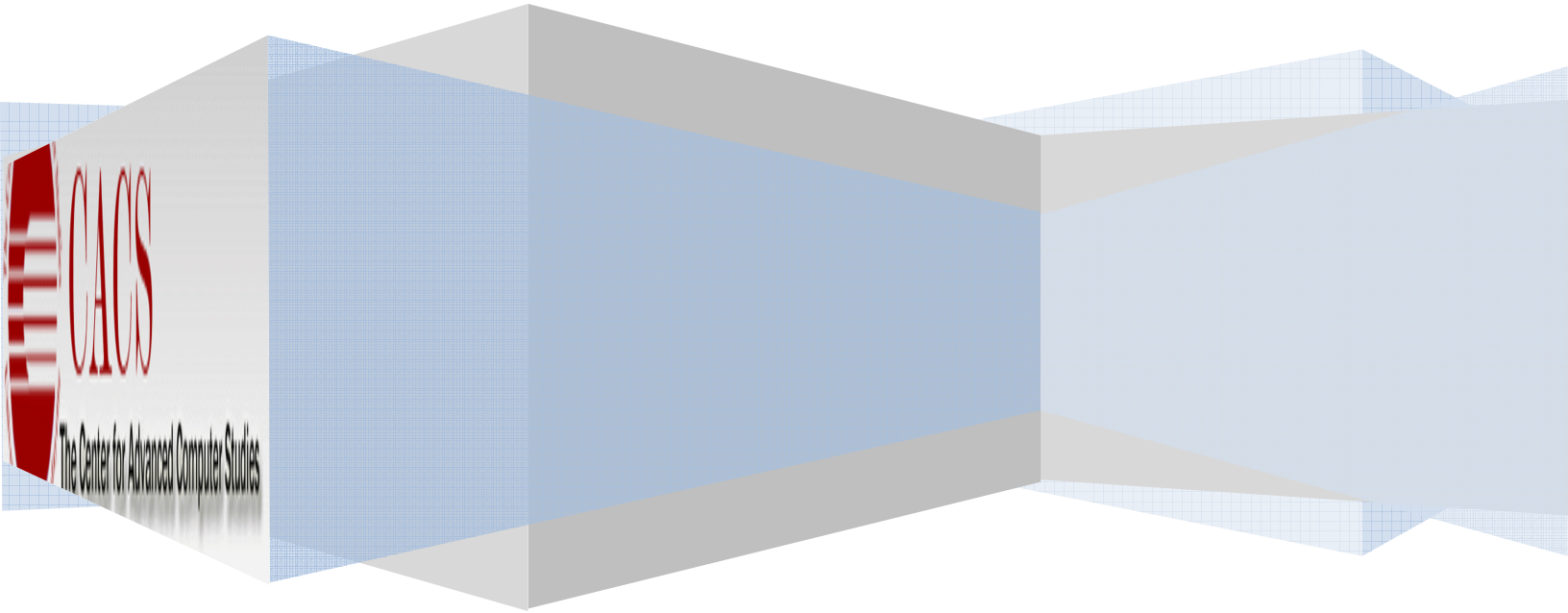
University of Louisiana at Lafayette
The Center for Advanced Computer Studies (CACS)

Bluetooth on Android Devices



By:

- Sajjad Pourmohammad

Supervised by:

- Dr. Honguyi Wu

Table of Contents

Table of figures

1. Introduction

One of the essential steps in designing new communication protocols is practical and real-world implementation of the protocol. There might be some issues that are not considered in design stage and they might be remained undetected in simulations. Using real devices and emulating a more realistic network can reveal that kind of undetected problems. Network design procedure can be divided in 4 different stages as shown in Figure 1 [1].

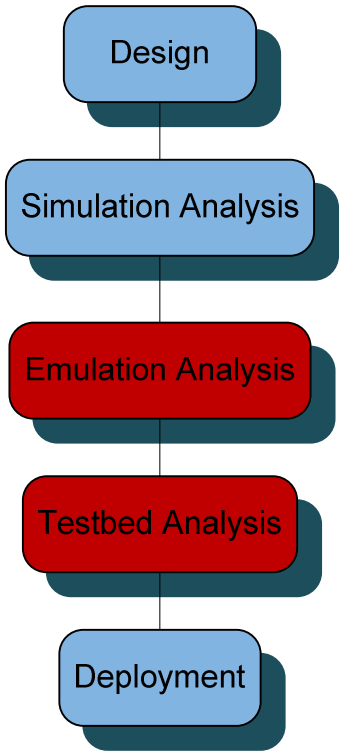


Figure 1: Network Protocol Design Stages.

The highlighted stages need to establish a connection between real devices. In these project basic building blocks of establishing such wireless connection especially over Bluetooth is investigated. The target devices are Dell Streak 7 mobile tablets (Figure 2).

Figure 2: Dell Streak 7 Mobile Tablet.

The software stack installed on the devices is Android. Google purchased it from the original developer in 2005[1]. According to data released by Nielsen [2], half of consumers who recently purchased a Smartphone purchased an Android phone. This is twice the "recent acquirer" market share of Apple's iOS, which made up 25 percent. The next-closest competitor was RIM, whose Blackberry devices pulled a 15 percent share (Figure 3).

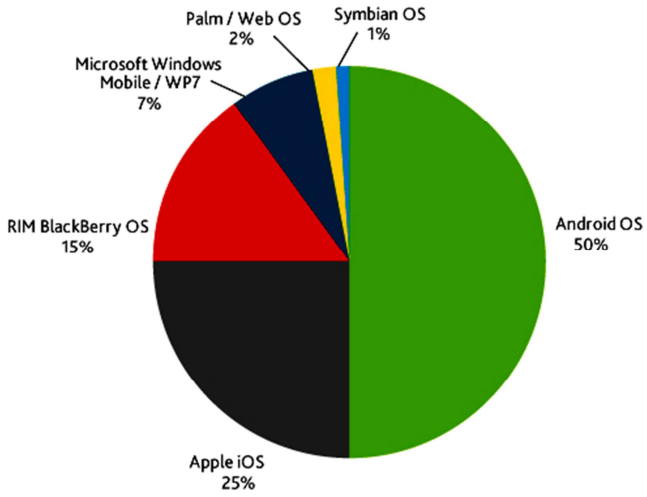


Figure 3: Android Devices Market Share 2011 [2].

[1] Original Page from BuisinessWeek Website

By providing an open development platform, Android offers developers the ability to build extremely rich and innovative applications. Developers are free to take advantage of the device hardware, access location information, run background services, set alarms, add notifications to the status bar, and much, much more [3].

Using the development tools provided for the developers writing a code to run in android devices is a lot easier than other platforms. By making the devices communicate with each other, we can build practical testbeds and emulation scenarios to test any new protocol or idea in wireless communication systems.

In this project all the basics for establishing a Bluetooth connection between android devices are presented. It's been tried to write the project report in a way that makes it an easy starter for anyone who wants to enter this area. All the useful references, pseudo-codes and issues discovered in debugging stage are included.

2. Used Tools

In this section we are going to give a brief overview of the tools used in this project. Overall description and useful information which are necessary for the project are included. To know more about installation and setup procedure for the tools listed in this section, you may refer to the first chapters of standard books in this area [4],[5].

2.1 Dell Streak 7 Tablet

This tablet (Figure 2) is America's first 4G Android tablet from T-Mobile[2]. The specifications and features that this tablet provides are listed below [6]:

- NVIDIA® Tegra™ 2 Mobile Processor
- Android™ 2.2 (Froyo)
- Vibrant 7" multi-touch screen display with Corning® Gorilla® Glass and full Adobe® Flash® 10.1
- Rear 5MP Auto Focus camera with Flash and 1.3 MP front-facing camera

[2] T-Mobile Web Site

- 16GB of internal storage, plus Wi-Fi™,
- Bluetooth®
- Built-in GPS

The most interesting features for us are Android 2.2 and Bluetooth. All the development tools and APIs for wireless application development are provided in Android 2.2 and the device has Bluetooth adapter. So, we're good to go.

2.2 Eclipse

Eclipse (Figure 4) is a multi-language software development environment comprising an IDE[3] and an extensible plug-in system. It is written mostly in Java and can be used to develop applications in Java and, by means of various plug-ins, other programming languages including Ada, C, C++, COBOL, Perl, PHP, Python, R. Ruby (including Ruby on Rails framework), Scala, Clojure, and Scheme [7].

Figure 4. Eclipse Startup Splash Screen

The Eclipse Project was originally created by IBM[4] in November 2001 and supported by a consortium of software vendors. The Eclipse Foundation was created in January 2004 as an independent not-for-profit corporation to act as the steward of the Eclipse community. The independent not-for-profit corporation was created to allow a vendor neutral and open, transparent community to be established around Eclipse. Today, the Eclipse community consists of individuals and organizations from a cross section of the software industry [8].

[3] Integrated Development Environment
[4] International Business Machines

Most of Android developers use this software to develop Android applications because it's well-adopted for Java programming. Android programming is actually Java programming with some new features and considerations like totally different UI, a lot of APIs and some precautions about managing the applications activities and resources. Furthermore Google recommends using this software [9]; so it's wise to do what they say.

2.3 JDK[5]

JDK is needed for writing and running Java programs. There are different bunch of JDKs. However, the most popular JDK is the one published by Sun Micro Systems (or let say Oracle!). It's available to download at Oracle's website[6].

2.4 Android SDK

The Android SDK[7] includes a comprehensive set of development tools. These include a debugger, libraries, a handset emulator (based on QEMU), documentation, sample code, and tutorials. Currently supported development platforms include computers running Linux (any modern desktop Linux distribution), Mac OS X 10.4.9 or later, Windows XP or later [10].

The Android SDK includes a mobile device emulator (Figure 5), a virtual mobile device that runs on your computer. The emulator lets you develop and test Android applications without using a physical device. When the emulator is running, you can interact with the emulated mobile device just as you would an actual mobile device, except that you use your mouse pointer to "touch" the touch screen and can use some keyboard keys to invoke certain keys on the device [11].

[5] Java Development Kit
[6] JDK Download Page
[7] Software Development Kit

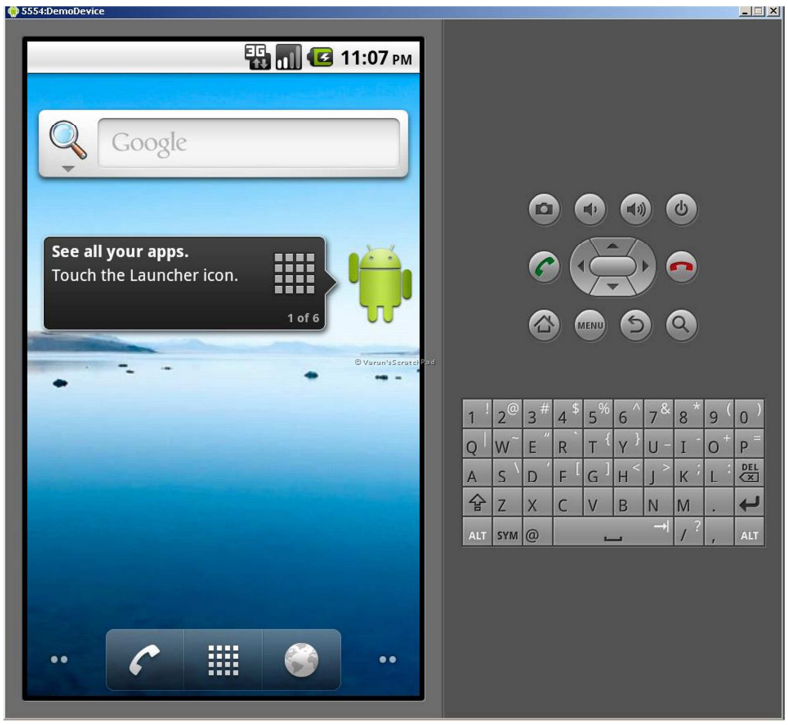


Figure 5. Android Emulator

Although the emulator doesn't support Bluetooth, using the emulator for other parts of the code is really useful.

2.5 ADT Plug-in for Eclipse

ADT[8] is a plugin for the Eclipse IDE that is designed to give the programmer a powerful, integrated environment in which to build Android applications. ADT extends the capabilities of Eclipse to let you quickly set up new Android projects, create an application UI, add components based on the Android Framework API, debug your applications using the Android SDK tools, and even export signed (or unsigned) .apk files in order to distribute your application [12]. Figure 6 shows all the softwares together.

[8] Android Development Tools

Figure 6. Tools needed for Android application development

3. Starting Android Programming

There are some books for Android application development [4][13][14][5][15][16]. However, most of the books are not well-suited for our purpose. Since we are trying to focus on wireless communication between Android devices, a lot of unnecessary information is not needed. On the other hand, a good tutorial-fashioned work can accelerate the process of starting Android wireless application development. This section tries to present all the basics which are collected after spending considerable time on learning, testing and debugging Android projects. Basic programming experience has been supposed. For the students who don't have any experience in Java, going through the first chapters from standard self-study java books is recommended [17][18].

3.1 Android Project Anatomy

Before proceeding let's create an Android project and then discuss the project's components. If you have installed all the tools listed in the previous section, you should be able to build an Android project by following the following steps:

1. From the File menu choose "New" and then "Other…"
2. Choose Android Project from the Android root tree.
3. Follow the wizard's steps and select the target Android version. You can also load an example from sample projects.

When you build the first Android project a bunch of files and folders will be created in the project's folder. A sample screenshot from an Android project is shown in Figure 7.

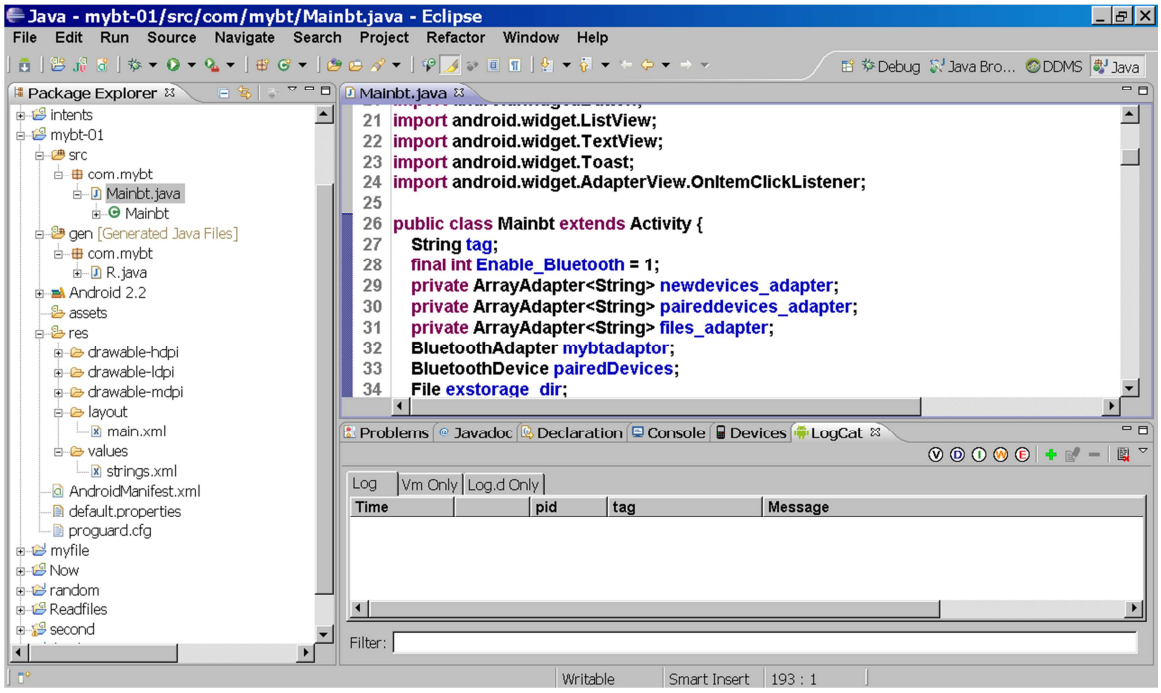


Figure 7. Sample Android Project

In this example the Android project's name is mybt-01. After expanding the project in package explorer window, you will find out that the following parts exist in the project. We are going to briefly describe their functionality.

- **src**: this folder contains the source Java files. Actually, here is the place for the code you write. Inside this folder the package name, chosen in building the project, is located. The package can contain several java files. But, there is at least one Java file there (in this case Mainbt.java as shown in Figure 8). All the classes inside that java file is shown too.

Figure 8. src folder contains all java classes

- **gen**: This folder contains the Java files generated by the system itself. The layout, all the user interface components like Buttons and Textviews are listed in R.java and have a unique id that makes it easy to refer to them inside our code. In other words R.java helps the programmer to access UI[9] components, variables defined and attributes.

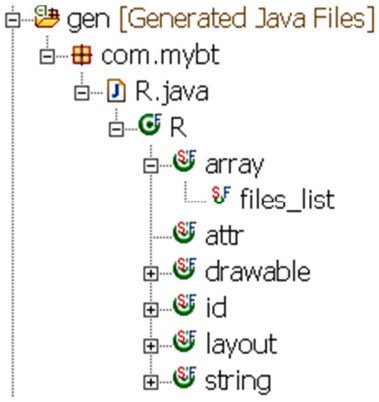


Figure 9. gen folder contains system generated files.

- **res**: drawable folders contain the image and icon files used in the application. You can put your own icon file here to use as default icon. Layout folder contains the layouts in your application. When you press a button and navigate to another page the layout actually changes. All the layouts are created and located in this folder. Values contain the variables defined in your application. For example here strings.xml contains all the strings defined and their values.

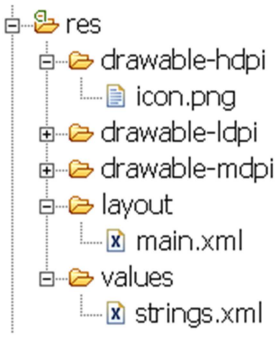


Figure 10. res contains the icons, layouts and values.

[9] User Interface

- ❖ *Attention*: Put the variables that you may change in future in "values" folder. This way you don't need to look for that variable inside your application!

- **AndroidManifest.xml:** This is the place that a lot of things happen there! In this XML file you define all the activities, permissions and intents related to the application.

The other parts in the package explorer are not used in this project.

4. Bluetooth Programming in Android Devices

Android provides some API's for using Bluetooth devices that let applications wirelessly connect to other Bluetooth devices, enabling point-to-point and multipoint wireless features. All the API's are accessible in Bluetooth package [19]. Using these API's the programmer can:

- Scan for other Bluetooth devices
- Query the local Bluetooth adapter for paired Bluetooth devices
- Establish RFCOMM[10] channels
- Connect to other devices through service discovery
- Transfer data to and from other devices
- Manage multiple connections

  - ❖ Attention: In order to be able to use Bluetooth in your application you have to use at least one of *BLUETOOTH* or *BLUETOOTH_ADMIN* permissions.

To declare the permissions we can put something like the following code in Manifest file:

[10] **Radio frequency communication**

```
<manifest ... >
  <uses-permission android:name="android.permission.BLUETOOTH" />
  ...
</manifest>
```

4.1 Setting up Bluetooth Adapter

The first step is to find the device Bluetooth adapter and its parameters. BluetoothAdapter is a class that does all this. The BluetoothAdapter lets you perform fundamental Bluetooth tasks, such as initiate device discovery, query a list of bonded (paired) devices, instantiate a BluetoothDevice using a known MAC address, and create a BluetoothServerSocket to listen for connection requests from other devices.

For example the following code gets the device default Bluetooth adapter and checks to see if it's enabled or not. If it's not enabled an intent to enable Bluetooth will be started:

```
BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.
getDefaultAdapter();
if (mBluetoothAdapter == null) {
    // Device does not support Bluetooth!!
}
if (!mBluetoothAdapter.isEnabled()) {Intent enableBtIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
    startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
}
```

A dialog will appear requesting user permission to enable Bluetooth, as shown in Figure 11.

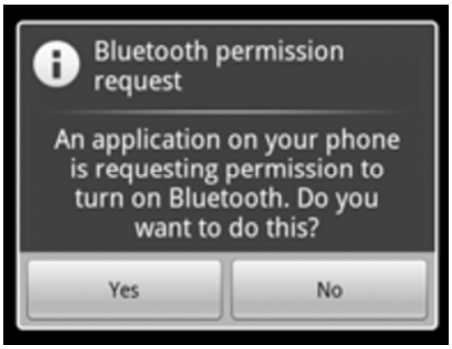


Figure 11. Enabling Bluetooth Adapter

If the user responds "Yes," the system will begin to enable Bluetooth and focus will return to your application once the process completes (or fails). If enabling Bluetooth succeeds, your Activity will receive the RESULT_OK result code in the

onActivityResult() callback. If Bluetooth was not enabled due to an error (or the user responded "No") then the result code will be RESULT_CANCELED.

4.2 Device discovery

Device discovery is a scanning procedure that searches the local area for Bluetooth enabled devices and then requesting some information about each one. However, a Bluetooth device within the local area will respond to a discovery request only if it is currently enabled to be discoverable. If a device is discoverable, it will respond to the discovery request by sharing some information, such as the device name, class, and its unique MAC address. Using this information, the device performing discovery can then choose to initiate a connection to the discovered device[19].

Before performing device discovery, 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 BluetoothDevices representing paired devices. For example, you can query all paired devices and then show the name of each device to the user, using an ArrayAdapter. A sample code to do so is represented in the following:

```
Set<BluetoothDevice> pairedDevices =
mBluetoothAdapter.getBondedDevices();
// Check to see if there is any bonded device
if (pairedDevices.size() > 0) {
    // Add the names to the list for example
    for (BluetoothDevice device : pairedDevices) {
       mArrayAdapter.add(device.getName()+"\n"+device.getAddress());
    }
}
```

To start discovering devices, simply call startDiscovery(). The process is asynchronous and the method will immediately return with a boolean indicating whether discovery has successfully started. The discovery process usually involves an inquiry scan of about 12 seconds, followed by a page scan of each found device to retrieve its Bluetooth name.

application must register a BroadcastReceiver for the ACTION_FOUND Intent in order to receive information about each device discovered. For each device, the system will broadcast the ACTION_FOUND Intent. This Intent carries the extra fields EXTRA_DEVICE and

EXTRA_CLASS, containing a BluetoothDevice and a BluetoothClass, respectively. For example, here's how you can register to handle the broadcast when devices are discovered:

```
// Create a BroadcastReceiver for ACTION_FOUND
private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        // When discovery finds a device
        if (BluetoothDevice.ACTION_FOUND.equals(action)) {
            // Get the BluetoothDevice object from the Intent
            BluetoothDevice device =
intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
      mArrayAdapter.add(device.getName()+"\n"+device.getAddress());
        }
    }
};
// Register the BroadcastReceiver
IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
registerReceiver(mReceiver, filter);
```

If you would like to make the local device discoverable to other devices, call startActivityForResult(Intent, int) with the *ACTION_REQUEST_DISCOVERABLE* action Intent. This will issue a request to enable discoverable mode through the system settings (without stopping your application). By default, the device will become discoverable for 120 seconds. You can define a different duration by adding the *EXTRA_DISCOVERABLE_DURATION* Intent extra. The maximum duration an app can set is 3600 seconds, and a value of 0 means the device is always discoverable. Any value below 0 or above 3600 is automatically set to 120 secs). For example, this snippet sets the duration to 300:

```
Intent discoverableIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATIO
N, 300);
startActivity(discoverableIntent);
```

A dialog will be displayed, requesting user permission to make the device discoverable. If the user responds "Yes," then the device will become discoverable for the specified amount of time. You do not need to enable device discoverability if you will be initiating

the connection to a remote device. Enabling discoverability is only necessary when you want your application to host a server socket that will accept incoming connections, because the remote devices must be able to discover the device before it can initiate the connection.

4.3 Connecting devices

In order to create a connection between your application on two devices, you must implement both the server-side and client-side mechanisms, because one device must open a server socket and the other one must initiate the connection (using the server device's MAC address to initiate a connection). The server and client are considered connected to each other when they each have a connected BluetoothSocket on the same RFCOMM channel. At this point, each device can obtain input and output streams and data transfer can begin.

The server device and the client device each obtain the required BluetoothSocket in different ways. The server will receive it when an incoming connection is accepted. The client will receive it when it opens an RFCOMM channel to the server. One implementation technique is to automatically prepare each device as a server, so that each one has a server socket open and listening for connections. Then either device can initiate a connection with the other and become the client. Alternatively, one device can explicitly "host" the connection and open a server socket on demand and the other device can simply initiate the connection. Here's the basic procedure to set up a server socket and accept a connection:

1. Get a BluetoothServerSocket by calling the listenUsingRfcommWithServiceRecord (String, UUID).

The string is an identifiable name of your service, which the system will automatically write to a new Service Discovery Protocol (SDP) database entry on the device (the name is arbitrary and can simply be your application name). The UUID is also included in the SDP entry and will be the basis for the connection agreement with the client device. That is, when the client attempts to connect with this device, it will carry a UUID that uniquely

identifies the service with which it wants to connect. These UUIDs must match in order for the connection to be accepted (in the next step).

2. Start listening for connection requests by calling accept().

This is a blocking call. It will return when either a connection has been accepted or an exception has occurred. A connection is accepted only when a remote device has sent a connection request with a UUID matching the one registered with this listening server socket. When successful, accept() will return a connected BluetoothSocket.

3. Unless you want to accept additional connections, call close().

The accept() call should not be executed in the main Activity UI thread because it is a blocking call and will prevent any other interaction with the application. It usually makes sense to do all work with a BluetoothServerSocket or BluetoothSocket in a new thread managed by your application. To abort a blocked call such as accept(), call close() on the BluetoothServerSocket (or BluetoothSocket) from another thread and the blocked call will immediately return. Note that all methods on a BluetoothServerSocket or BluetoothSocket are thread-safe. Here's a simplified thread for the server component that accepts incoming connections:

```
private class AcceptThread extends Thread {
    private final BluetoothServerSocket mmServerSocket;

    public AcceptThread() {
        // Use a temporary object that is later assigned to
mmServerSocket,
        // because mmServerSocket is final
        BluetoothServerSocket tmp = null;
        try {
            // MY_UUID is the app's UUID string, also used by the
client code
            tmp =
mBluetoothAdapter.listenUsingRfcommWithServiceRecord(NAME, MY_UUID);
        } catch (IOException e) { }
        mmServerSocket = tmp;
    }

    public void run() {
        BluetoothSocket socket = null;
        // Keep listening until exception occurs or a socket is
returned
        while (true) {
            try {
```

```
                socket = mmServerSocket.accept();
            } catch (IOException e) {
                break;
            }
            // If a connection was accepted
            if (socket != null) {
                // Do work to manage the connection (in a separate
thread)
                manageConnectedSocket(socket);
                mmServerSocket.close();
                break;
            }
        }
    }

    /** Will cancel the listening socket, and cause the thread to
finish */
    public void cancel() {
        try {
            mmServerSocket.close();
        } catch (IOException e) { }
    }
}
```

In this example, only one incoming connection is desired, so as soon as a connection is accepted and the BluetoothSocket is acquired, the application sends the acquired BluetoothSocket to a separate thread, closes the BluetoothServerSocket and breaks the loop.

4.4 Transferring bytes over Bluetooth

When you have successfully connected two (or more) devices, each one will have a connected BluetoothSocket. This is where the fun begins because you can share data between devices. Using the BluetoothSocket, the general procedure to transfer arbitrary data is simple:

- Get the InputStream and OutputStream that handle transmissions through the socket, via getInputStream() and getOutputStream(), respectively.
- Read and write data to the streams with *read(byte[])* and *write(byte[])*.

We should use a dedicated thread for all stream reading and writing. This is important because both *read(byte[])* and *write(byte[])* methods are blocking calls. *read(byte[])* will block until there is something to read from the stream. *write(byte[])* does not usually block, but can block for flow control if the remote device is not calling *read(byte[])* quickly enough and the intermediate buffers are full. So, your main loop in the thread

should be dedicated to reading from the InputStream. A separate public method in the thread can be used to initiate writes to the OutputStream.

5. File transfer over Bluetooth

In this project a simple file transfer application has been developed. The application tries to follow all the steps described in the previous section and transfer all the bytes from a file to the destination device. But, first we should read all the bytes from the source file. The next subsection shows how to do this.

5.1 Working with raw data from files

First of all we should locate the file and put the path in a file. For example the following code tries to read from source.mkv file and reports the result of trial to read.

```
File sourcedir = new File(exstorage + "/sdcard2/mydir/");
                 sourcefile = new File(sourcedir, "source.mkv");
                 boolean fileexist = sourcefile.exists();
                 if (fileexist) {
                         Log.d(tag, "source file exists");
                 } else
                         Log.d(tag, "source file doesn't exist");
                 boolean canread = sourcefile.canRead();
                 if (canread) {
                         Log.d(tag, "source file can be read");
                 } else
                         Log.d(tag, "source file cann't be read");
```

The next step is creating an input and output stream from the source and destination files. The following example code copies the bytes from source file to the destination.

```
try {
                         is = new FileInputStream(sourcefile);
                         os = new FileOutputStream(destfile);
                 } catch (FileNotFoundException e) {
                         Log.d(tag, "The IO streams creation FAILED");
                         e.printStackTrace();
                 }
                 bis = new BufferedInputStream(is);
                 bos = new BufferedOutputStream(os);
                 try {
                         while ((fileendflag = bis.read(buffer, 0, bytecount))!=-1) {
                                 bos.write(buffer, 0, fileendflag);
                         }
                 } catch (IOException e) {
                         Log.d(tag, "A Problem occured in copying file");
                         e.printStackTrace();
                 } finally {
```

```
                    try {
                        is.close();
                        os.close();
                        Log.d(tag, "DONE SUCCESSFULLY :D");
                    } catch (IOException e) {
                        Log.d(tag, "Couldn't close the streams!!!");
                        e.printStackTrace();
                    }
                }
```

5.2 Transferring files over Bluetooth

Sending bytes is as easy as writing to the connected thread. After establishing a connection a socket will be created between two devices. Input and output streams can be created from this connected socket. The following sample code shows how to do this:

```
try {
                    file_is = new FileInputStream(sourcefile);
                } catch (FileNotFoundException e) {
                    Log.d(TAG,e.toString());
                    e.printStackTrace();
                }
                buffered_file_is = new BufferedInputStream(file_is);
    send_file_button = (Button) findViewById(R.id.send_button);
    send_file_button.setOnClickListener(new OnClickListener() {
                    @Override
                    public void onClick(View v) {
                        if (mChatService.getState() !=
BluetoothChatService.STATE_CONNECTED) {
                        Log.d(TAG, "Not connected ====>  Send is not working");
                        mConversationArrayAdapter.add("Not connected! Don't try !!!! ");
                        } else {
                            byte[] buffer = new byte[512];
                            int completed_percentage = 0;
                            try {
                            while ((numread = buffered_file_is.read(buffer)) != -1 ) {
                                    Log.d(TAG, "read from source file:   " +
completed_percentage + " out of " + file_size);  // will show the number of bytes read in one
iteration
                                    mChatService.write(buffer);

                completed_percentage = completed_percentage + numread;

                                    }
                            } catch (IOException e) {
                                Log.d(TAG, e.toString());
                                e.printStackTrace();
                            } finally {
                                Log.d(TAG, "finally  ....");

                                Log.d(TAG, "read from source file:   " +
completed_percentage + " out of " + file_size);
                                mChatService.write(buffer);
```

```
                                        mConversationArrayAdapter.add("source file
has been sent succesfully" );
                                        mConversationArrayAdapter.add("total bytes
read from the file: " + completed_percentage );
                                        Log.d(TAG, "buffer bytes size: " + file_size);
                                        try {
                                                file_is.close();
                                                buffered_file_is.close();
                                        } catch (IOException e) {
                                                Log.d(TAG, e.toString());
                                                e.printStackTrace();
                                        }
                                }
```

6. Conclusions

In this project all the basics for connecting Android devices over Bluetooth has been studied. Working with network sockets, multi-thread programming and file I/O were examples of topics studied during this project. An application has been successfully tested for transferring files over Bluetooth.

7. Bibliography

[1] S Doshi, U Lee, R Bagrodia, and D McKeon, "NETWORK DESIGN AND IMPLEMENTATION USING EMULATION-BASED ANALYSIS," , Orlando, 2007.

[2] Ed Oswald. (2011, April) PCWorld. [Online]. http://www.pcworld.com/article/226339/android_market_share_growth_accelerating_nielsen_finds.html

[3] Google. (2011) Android Developers. [Online]. http://developer.android.com/guide/basics/what-is-android.html

[4] Shane Conder and Lauren Darcey, *Android Wireless Application Development*, 2nd ed.: Adison-Wesley, 2011.

[5] Wei-Meng Lee, *Beginning Android ApplicAtion development*.: Wiley, 2011.

[6] Dell Inc. Dell Streak 7. [Online]. http://www.dell.com/us/p/mobile-streak-7/pd

[7] Wiki. Wikipedia. [Online]. http://en.wikipedia.org/wiki/Eclipse_%28software%29

[8] Eclipse. Eclipse. [Online]. http://www.eclipse.org/org/#about

[9] Google. Google Developers. [Online]. http://developer.android.com/sdk/index.html

[10] Google. Android Developers. [Online]. http://developer.android.com/guide/developing/tools/index.html

[11] Google. Android Developers. [Online]. http://developer.android.com/guide/developing/tools/emulator.html

[12] Google. Android Developers. [Online]. http://developer.android.com/sdk/eclipse-adt.html

[13] Shane Conder and Lauren Darcey, *SamsTeach Yourself Android Application Development*.: SAMS, 2010.

[14] Donn Felker and Joshua Dobbs, *Android Application Development For Dummies*.: Wiley, 2011.

[15] Mark L. Murphy, *The Busy Coder's Guide to Android Development*, 36th ed.: CommonsWare, 2011.

[16] James Steele and Nelson To, *The Android Developer's Cookbook Building Applications with the Android SDK*.: Addison-Wesley, 2011.

[17] Kathy Sierra and Bert Bates, *Head First Java*, 2nd ed.: O'Reilly, 2005.

[18] Joshua Bloch, *Effective Java*, 2nd ed.: Addison-Wesley.

[19] Google. Bluetooth APIs. [Online]. http://developer.android.com/reference/android/bluetooth/package-summary.html

[20] Tim Edwards. opencircuitdesign. [Online]. http://opencircuitdesign.com/magic/

[21] William W. Peterson and T. D. Brown, "Cyclic Codes for Error Detection," , 1961.

[22] R. F. Hobson and K. L. Cheung, "A high-performance CMOS 32-bit parallel CRC engine," vol. 34, 1999.

[23] Sadiq M. Sait and W. Hasan, "Hardware design and VLSI implementation of a byte-wise CRCgenerator chip," 1995.

[24] Borelli Chris, "IEEE 802.3 Cyclic Redundancy Check," 2001.

[25] Tanenbaum Andrew S., *Computer Networks*., 2003.