

MINI PROJECT REPORT

On

EASYSHARE: A FILE SHARING APPLICATION

Submitted By:

Prachi Bansal (181500459)

Neha Adnekar (181500421)

Mani Bansal (181500364)

Vivek Goyal (181500817)

Jitendra Singh (181500300)

**Department of Computer Engineering & Applications
Institute of Engineering & Technology**



**GLA University
Mathura- 281406, INDIA
2020**

Certificates of Completion

digitaldefynd
find the best courses online

PRIME CERTIFICATE

Prachi Bansal

has successfully completed Android development course and has been awarded this Digital Defynd Certificate in November 2020



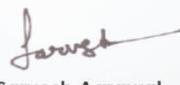
Certificate UID : D02008084

INTERNSHALA TRAININGS

CERTIFICATE OF TRAINING
Android App Development

Neha Adnekar from GLA University has successfully undergone a six weeks online winter training on Android App Development. The training program consisted of Introduction to Android, World of Kotlin, Android Kick-Off, Higher Order Functionalities and The Final Project modules and lasted for six weeks from 6th January, 2020 to 17th February, 2020.

We wish Neha all the best for future endeavours.


Sarvesh Agrawal
Founder & CEO

Date of certification: 2020-03-26
Certificate Number : 0B106AF3-D2EC-A630-DAFA-00520F28A820
For certificate authentication please visit https://trainings.internshala.com/verify_certificate



PRIME CERTIFICATE

Vivek Goyal

has successfully completed Android Development Course and has been awarded this Digital Defynd Certificate in November 2020

Certificate UID : D02008072



PRIME CERTIFICATE

Mani Bansal

has successfully completed Java for Android course and has been awarded this Digital Defynd Certificate in November 2020

Certificate UID : D02008042





ACKNOWLEDGEMENT

The project work in this report is an outcome of continuous work over a period and drew intellectual support from various sources. We would like to articulate our profound gratitude to all those people who extended their wholehearted co-operation and have helped us in completing this project successfully.

We are thankful to our mentor **Neeraj Khanna** for teaching and assisting us in making the project successful. We would also like to thank other fellow mates for guiding and encouraging us throughout the duration of the project.

We would also like to thank teaching staff for their constant encouragement, support and guidance which helped us in successfully completing the project work.

Neha Adnekar Prachi Bansal Mani Bansal Vivek Goyal Jitendra Singh



**Department of Computer Engineering and Applications
GLA University, Mathura**
**17 km. Stone NH#2, Mathura-Delhi Road, P.O. – Chaumuha,
Mathura – 281406**

DECLARATION

We hereby declare that the work which is being presented in the Mini Project Report **“File Sharing Application”** in partial fulfillment of the requirements for Mini Project on **EasyShare** is an authentic record of our own work carried under the supervision of **Neeraj Khanna, Department of Computer Engineering and Applications, GLA University.**

Name of the candidates:

| | | | | |
|------------------------------|-----------------------------|----------------------------|----------------------------|-------------------------------|
| Prachi Bansal (181500459) | Neha Adnekar (181500421) | Mani Bansal (181500364) | Vivek Goyal (181500817) | Jitendra Singh (181500300) |
|------------------------------|-----------------------------|----------------------------|----------------------------|-------------------------------|

ABSTRACT

In this we will build an android application that will transfer data from one device to another device using Wi-fi Direct. Our Team named this app "Easy Share". Easy Share is an open source Wi-fi Direct file sharing application that will enable sharing of data between Android devices running Android 4.0 or higher using a Wi-Fi direct connection without the use of a separate Wi-Fi access point. This will enable data transfer between devices without relying on any existing network infrastructure. This application is intended to provide a much higher speed alternative to Bluetooth file transfer.

There will be a client and a server i.e. the two devices that are connected through Wi-Fi direct. Once the connection is established, you can transfer data. Easy Share involves creating and registering a broadcast receiver for the application itself, discovering peers, connecting to a peer, and transferring data to a peer.

In this project comparative study and analysis of file sharing applications has been performed and presented. These file sharing applications enable the users for faster sharing rate among them through WLAN. They provide ease of access among multiple compatible devices. Several tests have been conducted including transferring several types of files and results have been gathered and these results are used in the analysis. A comparative analysis has also been done and the challenges or issues that are present in the concerned file sharing applications have been mentioned. The work primarily focuses on the commonly used file sharing applications in Smartphone's worldwide namely Xender and SHARE it.

Table of Contents

| | Page no. |
|---|-----------|
| ➤ Acknowledgement | 5 |
| ➤ Declaration | 6 |
| ➤ Abstract | 7 |
| Chapter 1 introduction | 9 |
| 1.1 Overview | 10 |
| 1.2 Working | 10 |
| 1.3 Motivation | |
| 1.3.1 Application Area | |
| 1.3.2 Development Challenges | 11 |
| 1.4 Problem Statements | |
| Chapter 2 Project functionalities & S/w & H/w Requirements | 12 |
| 2.1 Project Functionalities | 12 |
| 2.2 Software Requirements | 12 |
| 2.3 Hardware Requirements | |
| Chapter 3 Theory and/or Technologies | 13 |
| 3.1 Wi-Fi Direct | 13 |
| 3.2 Languages | 14 |
| 3.2.1 JAVA | |
| 3.2.2 XML | |
| 3.3 Database Designing | 15 |
| 3.4 Encryption | 15 |
| 3.5 Android Application | 16 |
| Chapter 4 Software Life Cycle | 18 |
| Spiral model | |
| Chapter 5 Software Design | 20 |
| 5.1 MODULE 1 Installing and configuring Android Studio | |
| 5.2 MODULE 2 User Interface creation | |
| 5.3 MODULE 3 Development of Java Files | |
| 5.4 MODULE 4 Development of Android Application | |
| Chapter 6 Implementation and Coding | 22 |
| 6.1 Layout Files Implementation Structure | 27 |
| 6.2 Java Files Implementation Structure | 33 |
| 6.3 Android Module Implementation Structure | 36 |
| Chapter 7 Software Testing and Debugging | 38 |
| 7.1 Software Testing | |
| 7.1.1 Unit testing | |
| 7.1.2 Integration testing | |
| 7.1.3 System testing | |
| 7.2 Approaches for testing | 39 |
| 7.2.1 White Box Testing | |
| 7.2.2 Black Box Testing | |
| 7.3 Debugging | |
| 7.3.1 Brute force method | |
| 7.3.2-Backtracking | |
| Chapter 8 Conclusion and Future Scope | 41 |
| Source codes & Screenshots | 42 |
| References | 54 |

CHAPTER 1

INTRODUCTION

1.1 Overview

Today's computers are capable of sharing all types of files, including documents, songs, videos, and full applications. File sharing is a key function for many businesses and other use cases.

Easy Share is an open source Wi-fi Direct file sharing application that will enable sharing of data between Android devices. File sharing is transferring or sharing the digital data such as image, audio, video, document etc. stored on an electronic device (mobile, laptop etc.) with similar compatible electronic devices. In this project we are using a Wi-Fi direct connection without the use of a separate Wi-Fi access point for sharing the files.

Wi-Fi Direct, initially called Wi-Fi P2P, is a Wi-Fi standard enabling devices to easily connect with each other without requiring a wireless access point. It is usable for everything from internet browsing to file transfer, and to communicate with more than one device simultaneously at typical Wi-Fi speeds. One advantage of Wi-Fi Direct is the ability to connect devices even if they are from different manufacturers. Only one of the Wi-Fi devices needs to be compliant with Wi-Fi Direct to establish a peer-to-peer connection that transfers data directly between them with greatly reduced setup.

Wi-fi Direct negotiate the link with a Wi-fi Protected Setup System that assigns each device a limited wireless access point. The "pairing" of Wi-fi direct devices can be set up to require the proximity of a near field communication, a Bluetooth, signal, or a button press one or all Devices. Wi-fi Direct may not only the need for Routers, but may also replace the need of Bluetooth for Application that do not rely on low battery.

1.2Working

The basic steps for a Wi-fi direct application are:

- Initial Setup
- Discovering Peers
- Connecting to Peers
- Transferring Data

We followed these steps in our application and registered a Broadcast receiver to listen to the intents. The way Wi-fi direct works is that we can call certain methods from the main activity to perform certain Wi-fi direct functions like search peers and connect. The methods will interact with the hardware. When hardware is ready, it will broadcast some intents to the OS and software. The broadcast receiver then can catch these intents and do the follow up works accordingly.

1.3 Motivation

The motivation for the study comes from the unique challenges offered in the varied application domain. We will now discuss the various Application Area of this project and then the problems statements.

1.3.1 Application Area

Cyber Security field- File sharing apps are a common tool used in cyber-attacks, both by professional hackers and ordinary individuals.

Education field- Study stuff & business documents instantly shared.

Government- Government use file sharing to share the important information on websites with limited rights.

IT Industries-With file sharing app, employee effort and time—become efficiently utilized.

1.3.2 Development Challenges

Scalability: The aim of scaling is to allow the Android Application to expand in scale without disrupting the activities of the users. The enormous data sharing should not lead to corrupt or any other issues.

Heterogeneity: The Android Application should be available and accessible from different platform without any difference in performance.

Real-Time Data Delivery: The Android Application should be able to deliver data in real-time when-ever data is requested.

Limited Bandwidth: Although Internet is available to all devices (within the targeted), but many of devices have very slow connection speed and for better costing Limited Bandwidth is to be taken care of.

1.4 Problem Statement

To design file sharing android application we having following problem statements -

1. Create an android-platform as user interface for Sending and Receiving or sharing of files.
2. To create Data base as back end to store Received files.
3. Transfer multiple files at once without losing progress if the transfer is interrupted or cancelled.
4. File transfer speed is about 2-4 MBPS.
5. Designing an user friendly interface for easy handling.
6. Ensuring the encryption of data during sharing process for batter security.
7. Applying several business concepts like Group owner information and server-client information.

CHAPTER 2

PROJECT FUNCTIONALITIES

2.1 Project Functionalities:

- Cross-platform - Only Android platform.
- Transfer multiple files at once, without losing progress if the transfer is interrupted or cancelled.
- File transfer speed is about 2-4 Mbps.
- Files encrypted in transit.
- It supports large file transfer.

2.2 Hardware Requirements:

Processor: i3
Hard Disk: 5GB
Memory: 1GB RAM
Android Phone with KitKat and higher

2.3 Software Requirements:

Windows 7 (ultimate, enterprise, Windows 8, Windows 10
Android Studio

CHAPTER 3

Theory and Technologies

Throws light on the theories and technologies related to this project.



3.1 Wi-fi Direct

- Wi-Fi direct is new technology defined by the Wi-Fi alliance aimed at enhancing direct device to device communication without requiring a wireless access point.
- Wi-Fi direct has simple setup, longer range and much higher transfer speed.
- Wi-Fi Direct is a better option than Bluetooth for wireless file transfers between two devices.
- Wi-Fi Direct implement Wi-Fi Protected Setup (WPS) to support a secure connection,

How it works?

1. The app uses Wi-fi P2P API to establish connection.
2. After connection establishment, a socket connection is established between two devices.
3. The group Owner acts as server and other acts as client.
4. Data is sent through streams.

3.2 Languages

3.2.1 Java



- Java is a computer programming language that is concurrent, class-based, object-oriented.
- Java was originally developed by James Gosling at Sun Microsystems (which has since merged into Oracle Corporation) and released in 1995 as a core component of Sun Microsystems' Java platform.
- It works as "write once, run anywhere" (WORA), meaning that code that runs on one platform does not need to be recompiled to run on another.
- Java applications are typically compiled to bytecode (class file) that can run on any Java virtual machine (JVM) regardless of computer architecture.
- Java is, as of 2014, one of the most popular programming languages in use, particularly for client-server web applications, with a reported 9 million developers.
- The language derives much of its syntax from C and C++, but it has fewer low-level facilities than either of them.

3.2.2 Xml



- The design goals of XML emphasize simplicity, generality, and usability over the Internet.
- It is a textual data format with strong support via Unicode for the languages of the world.
- Although the design of XML focuses on documents.

- It is widely used for the representation of arbitrary data structures, for example in web services.
- Many application programming interfaces (APIs) developers with processing XML data, and several based languages have been developed to aid software exist to aid in the definition of XML.

3.3 DATABASE DESIGNING



Database design is the process of producing a detailed data model of a database. This logical data model contains all the needed logical and physical design choices and physical storage parameters needed to generate a design in a Data Definition Language, which can then be used to create a database. A fully attributed data model contains detailed attributes for each entity.

The term database design can be used to describe many different parts of the design of an overall database system. Principally, and most correctly, it can be thought of as the logical design of the base data structures used to store the data. In the relational model these are the tables and views. In an object database the entities and relationships map directly to object classes and named **relationships**. However, the term database design could also be used to apply to the overall process of designing, not just the base data structures, but also the form sand queries used as part of the overall database application within the database management system (DBMS).The process of doing database design generally consists of a number of steps which will be carried out by the database designer.

3.4 ENCRYPTION



In cryptography, encryption is the process of transforming information (referred to as plaintext) using an algorithm (called a cipher) to make it unreadable to anyone except those possessing special knowledge, usually referred to as a key. The result of the process is encrypted information (in cryptography, referred to as cipher text). The reverse process, i.e., to make the encrypted information readable again, is referred to as decryption. Encryption can be used to protect data "at rest", such as files on computers and storage devices (e.g. USB flash drives) mobile telephones, wireless microphones, wireless intercom systems, Bluetooth devices and bank automatic teller machines.

3.5 ANDROID APPLICATION



Android, the world's most popular mobile platform, Android powers millions of phones, tablets, and other devices and brings the power of Google and the web into our hands.

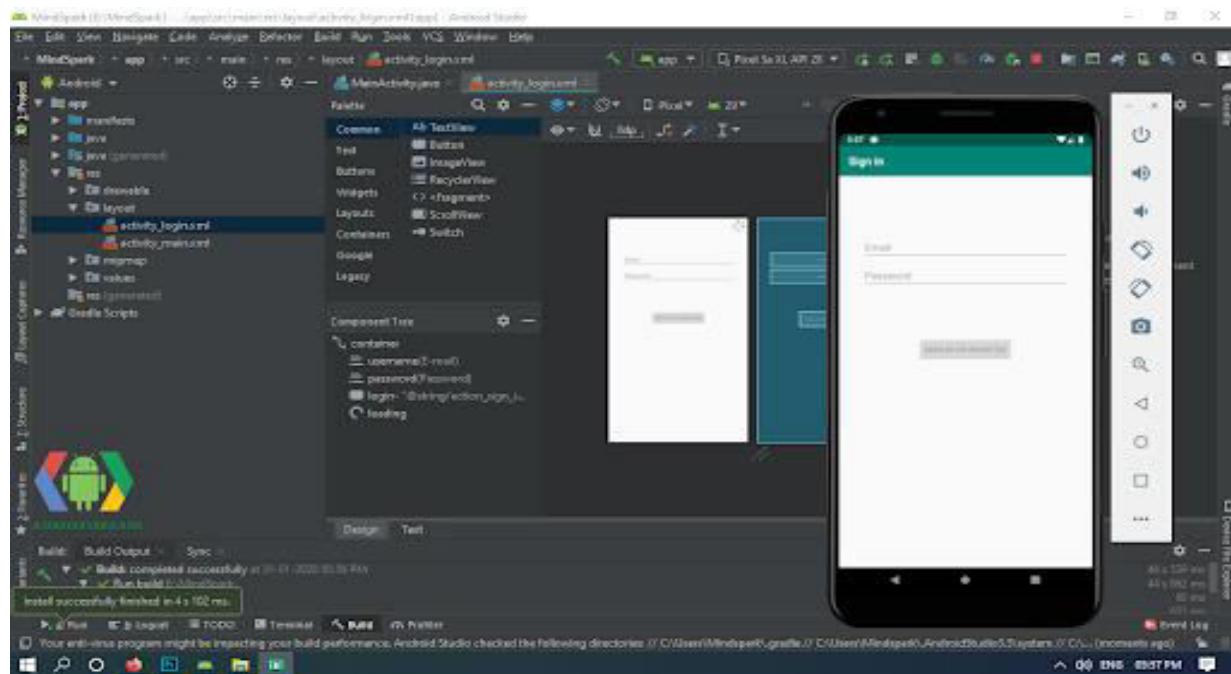
With an amazingly fast browser, cloud sync, multi-tasking, easy connect & share, and the latest Google apps (and thousands of other apps available on Google Play) our Android powered device is beyond smart.

The Android SDK includes a virtual mobile device emulator that runs on our computer. The Emulator let's you prototype, develop and test Android applications without using a physical device.

The Android emulator mimics all of the hardware and software features of a typical mobile device, except that it cannot place actual phone calls. It provides a variety of navigation and control keys, which you can "press" using your mouse or keyboard to generate events for your application. It also provides a screen in which our application is displayed, together with any other active Android applications.

The emulator also includes a variety of debug capabilities, such as a console from which you can log kernel output, simulate application interrupts (such as arriving SMS messages or phone calls), and simulate latency effects and dropouts on the data network.

The Android emulator is an application that provides a virtual mobile device on which you can run your android applications. It runs a full Android system stack, down to the kernel level, that includes a set of preinstalled applications.

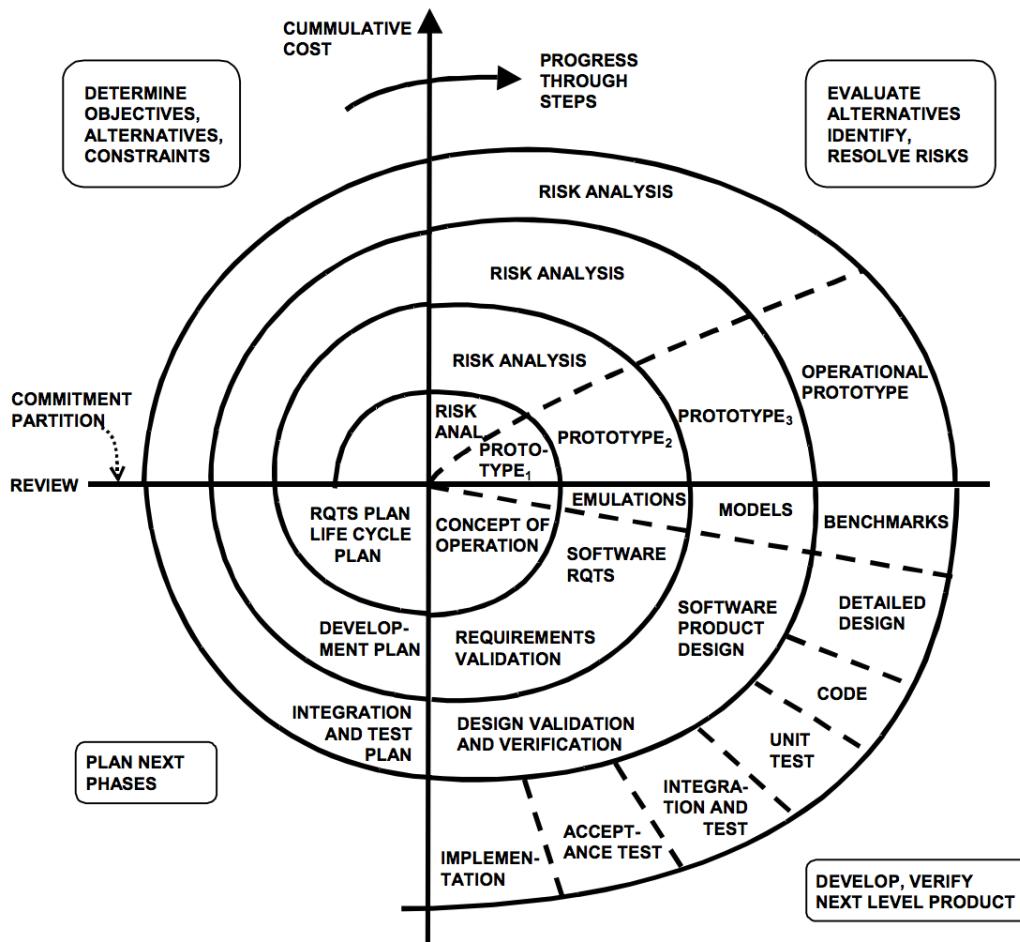


CHAPTER 4

SOFTWARE LIFECYCLE

A software life cycle model (also called process model) is a descriptive and diagrammatic representation of the software life cycle. A life cycle model represents all the activities required to make a software product transit through its life cycle phases. A life cycle model maps the different activities performed on a software product from its inception to retirement.

As we studied different short-coming and industrial approach to projects we found the **SPIRAL MODEL** is found to be most appropriate for our project. So, we have approached spiral model for life cycle of Software development. The Spiral model of software development is shown in figure. The diagrammatic representation of this model appears like a spiral with many loops. The exact number of loops in the spiral is not fixed. Each loop of the spiral represents a phase of the software process.



we see three loops as 3 stages of developments with different and increased models. So, in each quadrant we define 3 levels of development.

FIRST QUADRANT (RISK ANALYSIS AND PROTOTYPE DESIGN)

- **Loop1:** Risk analysis of: prototype1
- **Loop2:** Risk analysis of: prototype2
- **Loop3:** Risk analysis of: prototype3

SECOND QUADRANT (DETERMINE REQUIREMENT ANALYSIS)

- Determine the Requirements Analysis
- **Loop1:** Development of Layout Files: prototype1
- **Loop2:** Development of Java Files: prototype2
- **Loop3:** Development of Android application and access through it: prototype3

THIRD QUADRANT (PLANNING)

- **Loop1:** Software development
- **Loop2:** Development plan
- **Loop3:** Software integration and Test plan

FOURTH QUADRANT (DETAILED DESIGN AND IMPLEMENTATION)

- **Loop1:** Software requirement and requirement validation
- **Loop2:** Software design and design validation
- **Loop3:** Detailed design

The Spiral model development is implemented to this project.

CHAPTER 5

SOFTWARE DESIGN

Software design is the process of defining software methods, functions, objects, and the overall structure and interaction of your code so that the resulting functionality will satisfy your users requirements.

As a part of software design, we first divided the full project to 5 Modules. They are follows:

Module 1: Installing and configuring Android Studio.

Module 2: Development of Layout Files and Java Files.

Module 3: Development of User interface and Synchronization with Wi-fi- Direct.

Module 4: Development of Android Application and Access to Wi-fi -Direct

Module 5: Apply of Business models in all modules and Security Enhancements by Encryption

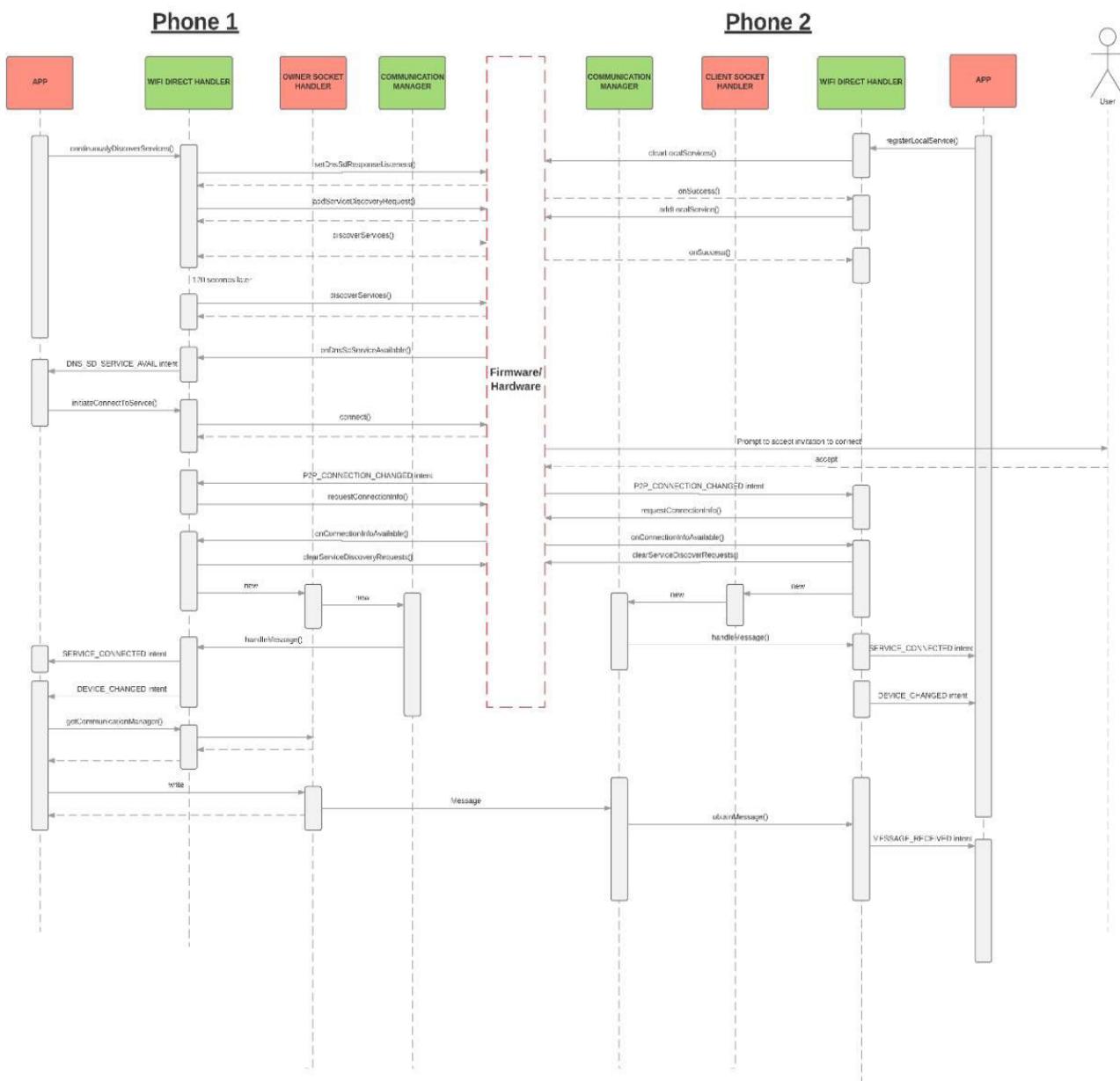
During the design the following points are being taken cared of as:

- The total design should be properly modulated
- **Cohesion** of the project is high
- **Coupling** of the project is low

Detailed Modular Design (Cohesion and Coupling)

CONNECTION SEQUENCE

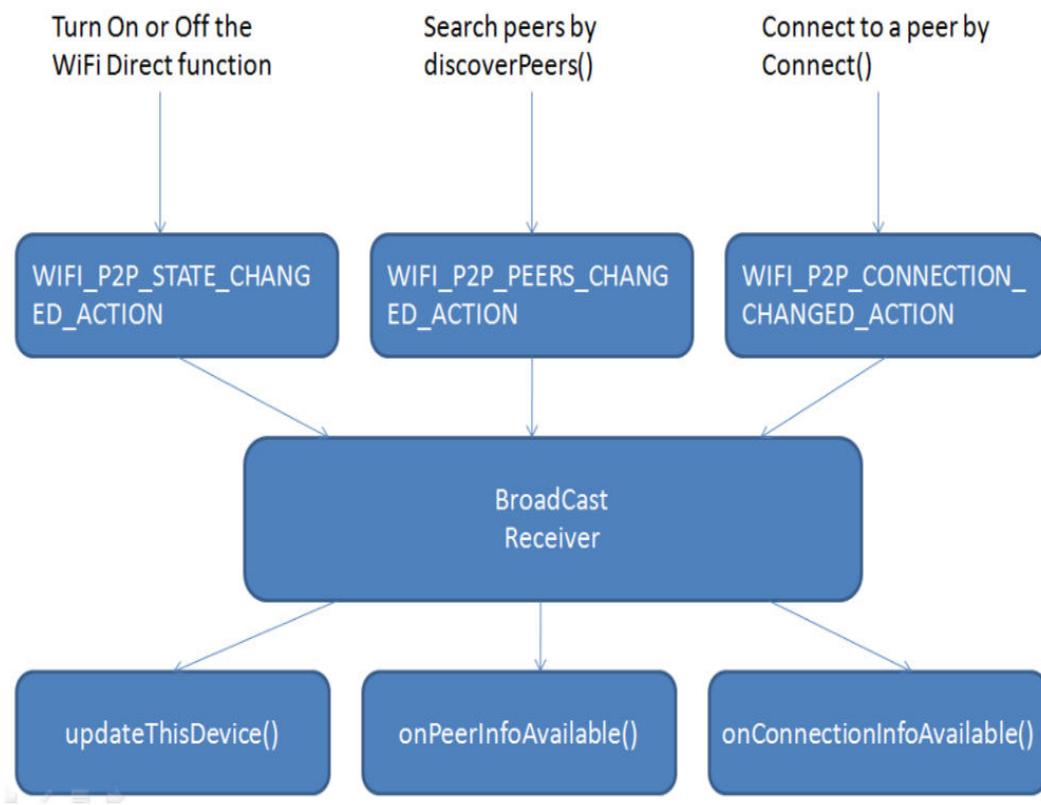
Crash Test Buddies | July 24, 2016



CHAPTER 6

Implementation and Coding

Implementation Details



1. Classes

WifiP2pManager: This is a class which is used to interact with the Wi-Fi hardware on our device which helps in connecting and discovering the peers.

2. Methods

WifiP2pManager class provides some methods for peer-to-peer connection.

initialize(): This method registers the application within a Wi-Fi framework, and it is called before calling any other Wi-Fi P2p methods.

connect(): This method is used to start a Wi-Fi pair connection with a device having a specific

discoverPeers(): This method is used to detect all the available peers that are in the range.

cancelConnect(): This method is used to cancel all the connected Wi-Fi pair connections.

removeGroup(): This method is used to remove the current peer-to-peer group.

Intents

When using the Wi-Fi pair connection, we need to listen for broadcast intents and handle the events occurred.

WIFI_P2P_STATE_CHANGED_ACTION: It indicates a change in the Wi-Fi P2P status

WIFI_P2P_PEERS_CHANGED_ACTION: It indicates a change in the list of available peers

WIFI_P2P_CONNECTION_CHANGED_ACTION: It indicates the state of Wi-Fi P2P

connectivity is changed.

WIFI_P2P_THIS_DEVICE_CHANGED_ACTION: It indicates the device details are changed.

EasyShare with Source code

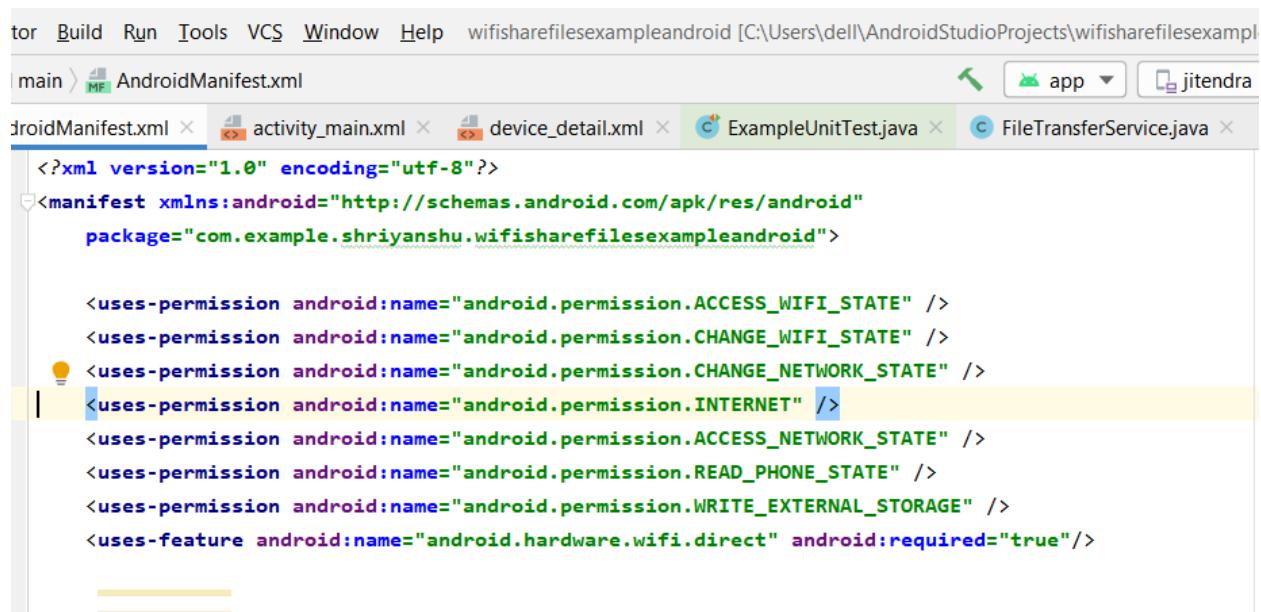
This Android Application which transfers files by creating a Wi-Fi pair connection in Android.

Create an Application **WiFiShareFilesExampleAndroid** in Android Studio
 Add the permissions in the **AndroidManifest.xml** file.

AndroidManifest.xml

To use the Wi-Fi pair connection, we have to add some necessary permissions in this file.

Screenshot-



```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.shriyanshu.wifisharefilesexampleandroid">

    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
    <uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-feature android:name="android.hardware.wifi.direct" android:required="true"/>

```

Now do it programmatically by creating some Java files and XML files in our project.

Set up Wi-Fi P2P Framework:

```

WifiP2pManager mManager;
Channel mChannel;
BroadcastReceiver mReceiver;
...
@Override
protected void onCreate(Bundle savedInstanceState){
    ...
    mManager = (WifiP2pManager) getSystemService(Context.WIFI_P2P_SERVICE);
    mChannel = mManager.initialize(this, getMainLooper(), null);
    mReceiver = new WiFiDirectBroadcastReceiver(mManager, mChannel, this);
    ...
}

```

Check Wi-Fi P2P supported or not:

```

// A BroadcastReceiver that notifies of important Wi-Fi p2p events.

public class WiFiDirectBroadcastReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();

        if (WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION.equals(action)) {

            int state = intent.getIntExtra(WifiP2pManager.EXTRA_WIFI_STATE, -1);
            if (state == WifiP2pManager.WIFI_P2P_STATE_ENABLED) {
                // Wifi P2P is enabled
            } else {
                // Wi-Fi P2P is not enabled
            }
        }
    }
}

```

Register BroadcastReceiver:

```

IntentFilter mIntentFilter;
...
@Override
protected void onCreate(Bundle savedInstanceState){
    ...
    mIntentFilter = new IntentFilter();
    mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION);
    mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION);
    mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION);
    mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION);
    ...
}

```

Discovering peers:

| | |
|--|--|
| <pre> mManager.discoverPeers(channel, new WifiP2pManager.ActionListener() { ... @Override public void onSuccess() { ... } ... @Override public void onFailure(int reasonCode) { ... } }); </pre> | <pre> PeerListListener myPeerListListener; ... if (WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION.equals(action)) { // request available peers from the wifi p2p manager. This is an // asynchronous call and the calling activity is notified with a // callback on PeerListListener.onPeersAvailable() if (mManager != null) { mManager.requestPeers(mChannel, myPeerListListener); } } </pre> |
|--|--|

Connecting to peers:

```
//obtain a peer from the WifiP2pDeviceList
WifiP2pDevice device;
WifiP2pConfig config = new WifiP2pConfig();
config.deviceAddress = device.deviceAddress;
mManager.connect(mChannel, config, new ActionListener() {

    @Override
    public void onSuccess() {
        //success logic
    }

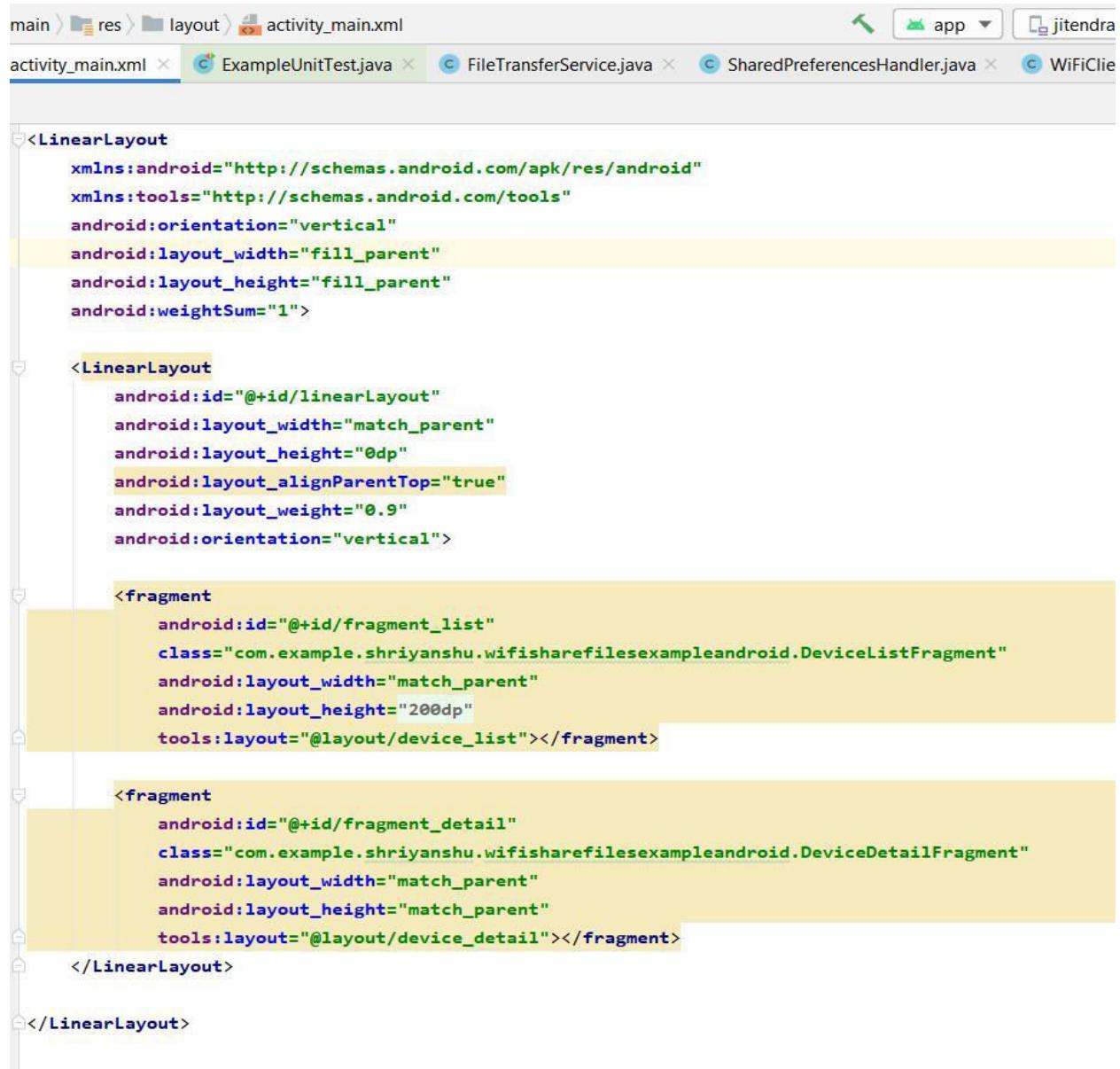
    @Override
    public void onFailure(int reason) {
        //failure logic
    }
});
```

6.1 Layout Files:

1. activity_main.xml

This is our main layout file. Here, we add two fragments inside LinearLayout.

Screenshots



The screenshot shows the Android Studio code editor with the file `activity_main.xml` open. The code defines a vertical linear layout containing two fragments: a list fragment and a detail fragment.

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:weightSum="1">

    <LinearLayout
        android:id="@+id/linearLayout"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_alignParentTop="true"
        android:layout_weight="0.9"
        android:orientation="vertical">

        <fragment
            android:id="@+id/fragment_list"
            class="com.example.shriyanshu.wifisharefilesexampleandroid.DeviceListFragment"
            android:layout_width="match_parent"
            android:layout_height="200dp"
            tools:layout="@layout/device_list"></fragment>

        <fragment
            android:id="@+id/fragment_detail"
            class="com.example.shriyanshu.wifisharefilesexampleandroid.DeviceDetailFragment"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            tools:layout="@layout/device_detail"></fragment>
    </LinearLayout>
</LinearLayout>
```

2. device_detail.xml

Screenshots

The image shows two side-by-side screenshots of the Android Studio code editor. Both screens display the same XML code for the `device_detail.xml` layout file.

```

<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="fill_parent"
    android:visibility="gone">
    <LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
        <LinearLayout
            xmlns:android="http://schemas.android.com/apk/res/android"
            android:orientation="horizontal"
            android:layout_width="match_parent"
            android:layout_height="wrap_content">
            <Button
                android:id="@+id/btn_connect"
                android:layout_width="100dp"
                android:layout_height="wrap_content"
                android:text="Connect"
                android:background="@color/colorPrimary"
                android:layout_margin="5dp"
                android:textColor="#ffffffff"/>
            <Button
                android:id="@+id/btn_disconnect"
                android:layout_width="100dp"
                android:layout_height="wrap_content"
                android:text="Disconnect"
                android:background="@color/colorPrimary"
                android:layout_margin="5dp"
                android:textColor="#ffffffff"/>
            <Button
                android:id="@+id/btn_start_client"
                android:layout_width="130dp"
                android:layout_height="wrap_content"
                android:text="Launch Gallery"
                android:layout_width="150dp"
                android:layout_height="wrap_content"
                android:visibility="gone"
                android:background="@color/colorPrimary"
                android:layout_margin="5dp"
                android:textColor="#ffffffff"/>
        </LinearLayout>
        <TextView
            android:id="@+id/tv_deviceAddress"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />
        <TextView
            android:id="@+id/tvDeviceInfo"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />
        <TextView
            android:id="@+id/tv_groupOwner"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />
        <TextView
            android:id="@+id/tv_groupIp"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />
    </LinearLayout>
    <LinearLayout
        android:layout_width="match_parent"
        android:id="@+id/l1_statusBar"
        android:orientation="vertical"
        android:layout_gravity="bottom"
        android:layout_height="wrap_content"
        android:layout_marginBottom="5dp"
        >
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_gravity="center"
            android:layout_margin="5dp"
            android:id="@+id/tv_statusText"
            >
            </TextView>
        </LinearLayout>
    </FrameLayout>

```

The code defines a `FrameLayout` with a horizontal orientation. It contains a `LinearLayout` with a vertical orientation. Inside this are two nested `LinearLayout`s. The innermost `LinearLayout` contains three `Button` elements. The middle `LinearLayout` contains four `TextView` elements. The outermost `FrameLayout` contains one `LinearLayout` at the bottom, which in turn contains one `TextView`. Several attributes like `layout_width`, `layout_height`, and `text` are used throughout the code.

3. device_list.xml

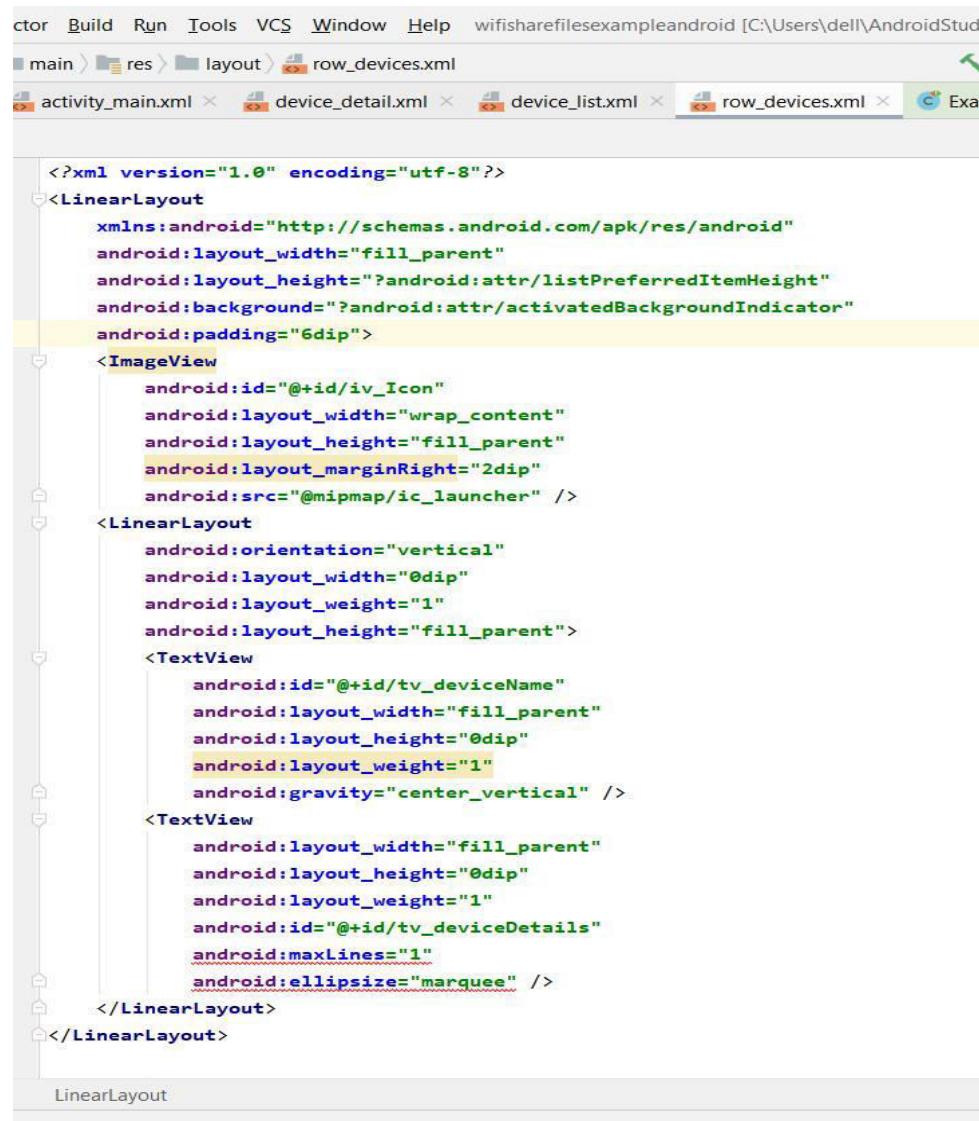
Screenshots

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout
3     xmlns:android="http://schemas.android.com/apk/res/android"
4     android:orientation="vertical"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     android:paddingTop="3dp">
8
9     <TextView
10        android:layout_width="fill_parent"
11        android:layout_height="wrap_content"
12        android:gravity="center_vertical"
13        android:text="ME" />
14     <View
15        android:layout_width="fill_parent"
16        android:layout_height="1dp"
17        android:gravity="center_vertical"
18        android:background="@android:color/holo_blue_light" />
19
20     <!-- Self information -->
21     <LinearLayout
22         xmlns:android="http://schemas.android.com/apk/res/android"
23         android:layout_width="fill_parent"
24         android:layout_height="?android:listPreferredItemHeight"
25         android:background="?android:attr/activatedBackgroundIndicator"
26         android:padding="3dp">
27         <ImageView
28             android:id="@+id/iv_Icon"
29             android:layout_width="wrap_content"
30             android:layout_height="fill_parent"
31             android:layout_marginRight="2dp"
32             android:src="@mipmap/ic_launcher" />
33         <LinearLayout
34             android:orientation="vertical"
35             android:layout_width="0dp"
36             android:layout_weight="1"
37             android:layout_height="fill_parent">
38             <TextView
39                 android:id="@+id/tv_myName"
40                 android:layout_width="fill_parent"
41                 android:layout_height="0dp"
42                 android:layout_weight="1"
43                 android:gravity="center_vertical" />
```

```
39         android:id="@+id/tv_myName"
40         android:layout_width="fill_parent"
41         android:layout_height="0dp"
42         android:layout_weight="1"
43         android:gravity="center_vertical" />
44     <TextView
45         android:id="@+id/tv_myStatus"
46         android:layout_width="fill_parent"
47         android:layout_height="0dp"
48         android:layout_weight="1"
49         android:maxLines="1"
50         android:ellipsize="marquee" />
51     </LinearLayout>
52 </LinearLayout>
53
54 <TextView
55     android:layout_width="fill_parent"
56     android:layout_height="wrap_content"
57     android:gravity="center_vertical"
58     android:text="PEERS" />
59
60 <View
61     android:layout_width="fill_parent"
62     android:layout_height="1dp"
63     android:gravity="center_vertical"
64     android:background="@android:color/holo_blue_light" />
65
66 <!-- Available peers -->
67 <ListView
68     android:id="@+id/android:list"
69     android:layout_width="match_parent"
70     android:layout_height="match_parent"
71     android:layout_weight="1"
72     android:drawSelectorOnTop="false" />
73
74 <TextView
75     android:id="@+id/android:empty"
76     android:layout_width="match_parent"
77     android:layout_height="match_parent"
78     android:layout_gravity="center"
79     android:text="No devices found. Turn on P2P and perform dis-
```

4. row_devices.xml

Screenshots



The screenshot shows the Android Studio interface with the file `row_devices.xml` open in the editor. The code defines a linear layout with various views and attributes.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="?android:attr/listPreferredItemHeight"
    android:background="?android:attr/activatedBackgroundIndicator"
    android:padding="6dip">
    <ImageView
        android:id="@+id/iv_Icon"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_marginRight="2dip"
        android:src="@mipmap/ic_launcher" />
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="0dip"
        android:layout_weight="1"
        android:layout_height="fill_parent">
        <TextView
            android:id="@+id/tv_deviceName"
            android:layout_width="fill_parent"
            android:layout_height="0dip"
            android:layout_weight="1"
            android:gravity="center_vertical" />
        <TextView
            android:layout_width="fill_parent"
            android:layout_height="0dip"
            android:layout_weight="1"
            android:id="@+id/tv_deviceDetails"
            android:maxLines="1"
            android:ellipsize="marquee" />
    </LinearLayout>
</LinearLayout>
```

5. menu_main.xml

Screenshot

The screenshot shows the Android Studio interface with the code editor open to the `menu_main.xml` file. The file contains XML code defining a menu with two items: "P2P On/Off" and "Discover". The code is color-coded for syntax highlighting.

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"  
      xmlns:app="http://schemas.android.com/apk/res-auto"  
      xmlns:tools="http://schemas.android.com/tools"  
      tools:context=".MainActivity">  
    <item android:id="@+id/action_directEnable"  
          android:title="P2P On/Off"  
          android:orderInCategory="100"  
          app:showAsAction="never"/>  
    <item android:id="@+id/action_directDiscover"  
          android:title="Discover"  
          android:orderInCategory="100"  
          app:showAsAction="never"/>  
</menu>
```

6.2Java Files

WifiDirectBroadcastReceiver.java

This class extends with BroadcastReceiver. Here, this class notifies important Wi-Fi P2p events.

Initialize

```
private WifiP2pManager manager;
private Channel channel;
private MainActivity activity;
```

Add the Code in onReceive() Method:

```

public class WiFiDirectBroadcastReceiver extends BroadcastReceiver {
    private WifiP2pManager manager;
    private Channel channel;
    private MainActivity activity;
    public WiFiDirectBroadcastReceiver(WifiP2pManager manager, Channel channel, MainActivity activity) {
        super();
        this.manager = manager;
        this.channel = channel;
        this.activity = activity;
    }
    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if (WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION.equals(action)) {
            int state = intent.getIntExtra(WifiP2pManager.EXTRA_WIFI_STATE, -1);
            if (state == WifiP2pManager.WIFI_P2P_STATE_ENABLED) {
                activity.setWifiP2pEnabled(true);
            } else {
                activity.setWifiP2pEnabled(false);
                activity.resetData();
            }
        } else if (WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION.equals(action)) {
            if (manager != null) {
                manager.requestPeers(channel, (PeerListListener) activity.getFragmentManager()
                        .findFragmentById(R.id.fragment_list));
            }
        } else if (WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION.equals(action)) {
            if (manager == null) {
                return;
            }
            NetworkInfo networkInfo = (NetworkInfo) intent
                    .getParcelableExtra(WifiP2pManager.EXTRA_NETWORK_INFO);
            if (networkInfo.isConnected()) {
                DeviceDetailFragment fragment = (DeviceDetailFragment) activity
                        .getFragmentManager().findFragmentById(R.id.fragment_detail);
                manager.requestConnectionInfo(channel, fragment);
            } else {
                activity.resetData();
            }
        } else if (WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION.equals(action)) {
            DeviceListFragment fragment = (DeviceListFragment) activity.getFragmentManager()
                    .findFragmentById(R.id.fragment_list);
            fragment.updateThisDevice((WifiP2pDevice) intent.getParcelableExtra(
                    WifiP2pManager.EXTRA_WIFI_P2P_DEVICE));
        }
    }
}

```

MainActivity.java

This is our Main Activity. Here, it adds the necessary intent values by creating an intent-filter, registers the broadcast receiver, obtains the instance of WifiP2pManager class, registers the application by using initialize() method, also discover peers and connects with the available devices.

Initialize

```

public class MainActivity extends AppCompatActivity implements ChannelListener, DeviceActionListener {

    public static final String TAG = "WiFiShareFilesExampleAndroid";
    private WifiP2pManager manager;
    private boolean isWifiP2pEnabled = false;
    private boolean retryChannel = false;
    private final IntentFilter intentFilter = new IntentFilter();
    private Channel channel;
    private BroadcastReceiver receiver = null;

```

Add the code in onCreate(...){..}:

```

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // add necessary intent values to be matched.
    intentFilter.addAction(WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION);
    intentFilter.addAction(WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION);
    intentFilter.addAction(WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION);
    intentFilter.addAction(WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION);

    manager = (WifiP2pManager) getSystemService(Context.WIFI_P2P_SERVICE);
    channel = manager.initialize(srcContext: this, getMainLooper(), listener: null);
}

```

Now, register the receiver in onResume() method

```

@Override
public void onResume() {
    super.onResume();
    receiver = new WiFiDirectBroadcastReceiver(manager, channel, activity: this);
    registerReceiver(receiver, intentFilter);
}

@Override
public void onPause() {
    super.onPause();
    unregisterReceiver(receiver);
}

```

Add the code for discover peers

```

final DeviceListFragment fragment = (DeviceListFragment) getFragmentManager()
    .findFragmentById(R.id.fragment_list);
fragment.onInitiateDiscovery();
manager.discoverPeers(channel, new WifiP2pManager.ActionListener() {

    @Override
    public void onSuccess() {
        Toast.makeText( context: MainActivity.this, text: "Discovery Initiated",
            Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onFailure(int reasonCode) {
        Toast.makeText( context: MainActivity.this, text: "Discovery Failed : " + reasonCode,
            Toast.LENGTH_SHORT).show();
    }
});
return true;
default:
    return super.onOptionsItemSelected(item);
}

```

Now call connect() method for connecting with an available peer.

```

@Override
public void connect(WifiP2pConfig config) {
    manager.connect(channel, config, new ActionListener() {

        @Override
        public void onSuccess() {
            // WiFiDirectBroadcastReceiver will notify us. Ignore for now.
        }

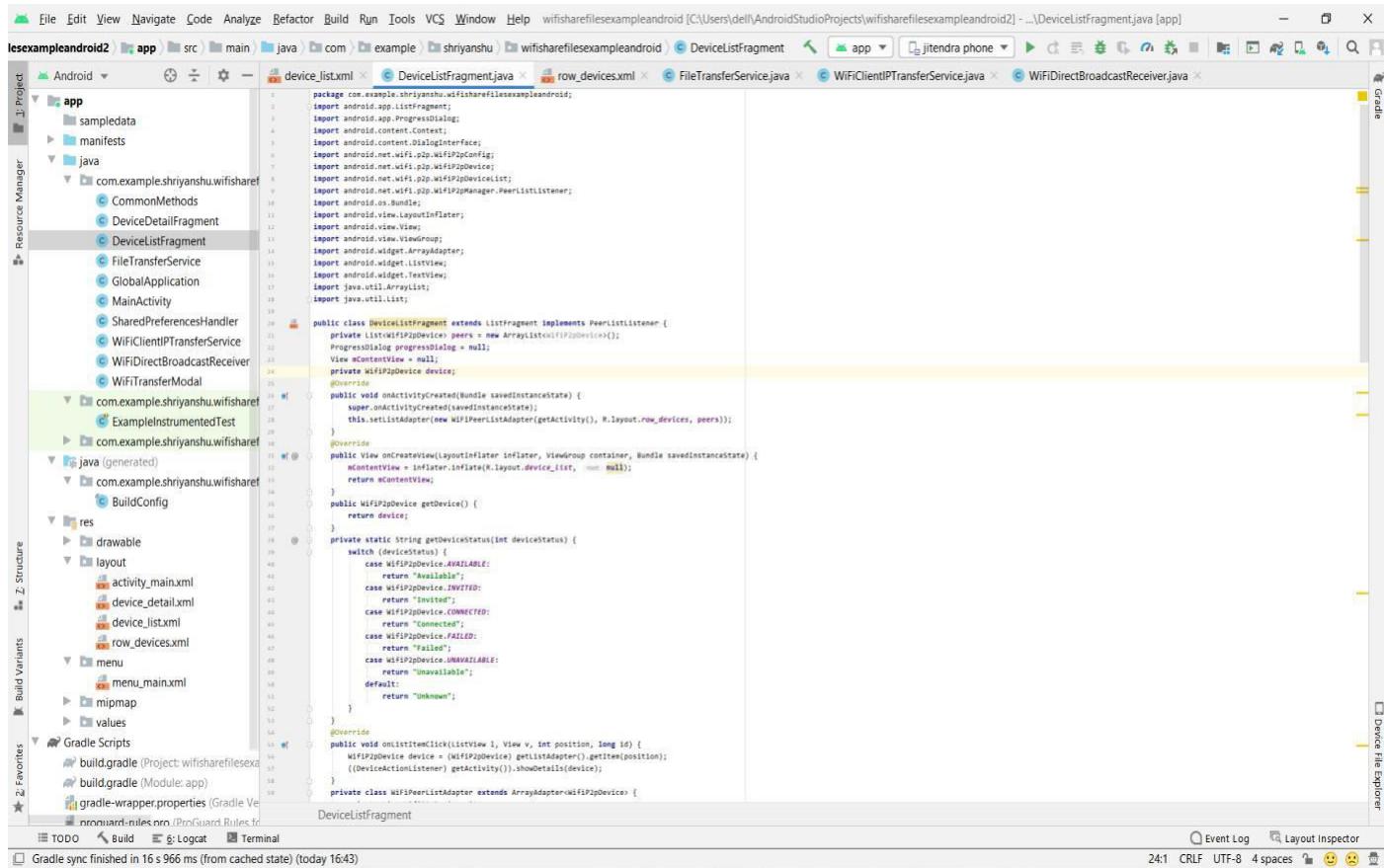
        @Override
        public void onFailure(int reason) {
            Toast.makeText( context: MainActivity.this, text: "Connect failed. Retry.",
                Toast.LENGTH_SHORT).show();
        }
    });
}

```

DeviceDetailFragment.java

This fragment class manages with a particular peer and “allows the interaction with the device” means when a connection is established successfully, we can transfer the data to the device by setting up a network connection.

Added the code here by creating a server socket that accepts the connection.



The screenshot shows the Android Studio interface with the DeviceListFragment.java file open in the code editor. The code implements a PeerListListener and overrides the onActivityCreated and onCreateView methods to handle a list of devices. It also includes a static method to get device status based on their IDs.

```

package com.example.shriyanshu.wifisharefilesexampleandroid;
import android.app.ListFragment;
import android.app.ProgressDialog;
import android.content.Context;
import android.content.DialogInterface;
import android.net.wifi.p2p.WifiP2pConfig;
import android.net.wifi.p2p.WifiP2pDevice;
import android.net.wifi.p2p.WifiP2pDeviceList;
import android.net.wifi.p2p.WifiP2pManager.PeerListListener;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;
import java.util.ArrayList;
import java.util.List;
public class DeviceListFragment extends ListFragment implements PeerListListener {
    private ArrayList<WifiP2pDevice> peers = new ArrayList<WifiP2pDevice>();
    ProgressDialog progressDialog = null;
    View mContentView = null;
    private WifiP2pDevice device;
    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        this.setListAdapter(new WiFiPeerListAdapter(getActivity(), R.layout.row_devices, peers));
    }
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        mContentView = inflater.inflate(R.layout.device_list, null);
        return mContentView;
    }
    public WiFiPeerDevice getDevice() {
        return device;
    }
    private static String getDeviceStatus(int deviceStatus) {
        switch (deviceStatus) {
            case WiFiPeerDevice.AVAILABLE:
                return "Available";
            case WiFiPeerDevice.INVITED:
                return "Invited";
            case WiFiPeerDevice.CONNECTED:
                return "Connected";
            case WiFiPeerDevice.FAILED:
                return "Failed";
            case WiFiPeerDevice.UNAVAILABLE:
                return "Unavailable";
            default:
                return "Unknown";
        }
    }
    @Override
    public void onListItemClick(ListView l, View v, int position, long id) {
        WiFiPeerDevice device = (WiFiPeerDevice) getListAdapter().getItems(position);
        ((DeviceClickListener) getActivity()).showDetails(device);
    }
    private class WiFiPeerListAdapter extends ArrayAdapter<WiFiPeerDevice> {
        DeviceListFragment()
    }
}

```

CHAPTER 7

SOFTWARE TESTING AND DEBUGGING

7.1 Software Testing

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding software bugs (errors or other defects).

7.1.1 Unit testing

Unit testing, also known as component testing, refers to tests that verify the functionality of a specific section of code, usually at the function level. In an object-oriented environment, this is usually at the class level, and the minimal unit tests include the constructors and destructors.

We verified every Class and methods for functionality under unit testing.

7.1.2 Integration testing

Integration testing is any type of software testing that seeks to verify the interfaces between components against a software design. Software components may be integrated in an iterative way or all together ("big bang"). Normally the former is considered a better practice since it allows interface issues to be localized more quickly and fixed.

Under this testing, we integrated the system and the functionality is compared with the software design, And the system performance is checked.

Phase 1: Module 1 is tested by integrating all the classes

Phase 2: Module 2 is tested by integrating all the classes

Phase 3: Module 2 is tested by integrating all the classes

7.1.3 System testing

System testing tests a completely integrated system to verify that it meets its requirements.

Under this testing strategy, we tested the whole system.

All the modules integrated in previous phases were tested.

7.2 Approaches for Testing

7.2.1 White-box testing

White-box testing is when the tester has access to the internal data structures and algorithms including the code that implement these.

7.2.2 Black-box testing

Black-box testing treats the software as a "black box"—without any knowledge of internal implementation. Black-box testing methods include: equivalence partitioning, boundary value analysis, all-pairs testing, fuzz testing, model-based testing, exploratory testing and specification based testing.

7.3 Debugging

Debugging is a methodical process of finding and reducing the number of bugs, or defects, in a computer program or a piece of electronic hardware, thus making it behave as expected. Debugging tends to be harder when various subsystems are tightly coupled, as changes in one may cause bugs to emerge in another. Many books have been written about debugging (see below: Further reading), as it involves numerous aspects, including interactive debugging, control flow, integration testing, log files, monitoring (application, system), memory dumps, profiling, Statistical Process Control, and special design tactics to improve detection while simplifying changes.

Approaches for debugging applied

7.3.1 Brute force method

Brute force method is common method but it is least efficient method. In this approach, print statements are inserted throughout the program.

```

final DeviceListFragment fragment = (DeviceListFragment) getFragmentManager()
        .findFragmentById(R.id.fragment_list);
fragment.onInitiateDiscovery();
manager.discoverPeers(channel, new WifiP2pManager.ActionListener() {

    @Override
    public void onSuccess() {
        Toast.makeText( context: MainActivity.this, text: "Discovery Initiated",
                Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onFailure(int reasonCode) {
        Toast.makeText( context: MainActivity.this, text: "Discovery Failed : " + reasonCode,
                Toast.LENGTH_SHORT).show();
    }
});
return true;
default:
    return super.onOptionsItemSelected(item);
}

```

7.3.2 Backtracking

In this approach, beginning from the statement at which an error symptom has been observed, the source code is traced backwards until the error is discovered. it is a fairly common approach.

The screenshot shows the Android Studio interface with the project 'wifisharefilesexampleandroid' open. The code editor displays the `WiFiDirectBroadcastReceiver` class, which extends `BroadcastReceiver`. The code handles various WiFi P2P actions such as state changes, peer changes, and connection changes. The code editor has syntax highlighting and code completion features. The left sidebar shows the project structure with files like `activity_main.xml`, `device_detail.xml`, and `menu_main.xml`. The bottom navigation bar includes tabs for `TODO`, `Build`, `Logcat`, `Terminal`, `Event Log`, and `Layout Inspector`.

```
import android.net.wifi.p2p.WifiP2pManager;
import android.net.wifi.p2p.WifiP2pManager.Channel;
import android.net.wifi.p2p.WifiP2pManager.PeerListListener;

public class WiFiDirectBroadcastReceiver extends BroadcastReceiver {
    private WifiP2pManager manager;
    private Channel channel;
    private MainActivity activity;
    public WiFiDirectBroadcastReceiver(WifiP2pManager manager, Channel channel,
                                       MainActivity activity) {
        super();
        this.manager = manager;
        this.channel = channel;
        this.activity = activity;
    }
    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if (WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION.equals(action)) {
            int state = intent.getIntExtra(WifiP2pManager.EXTRA_WIFI_STATE, -1);
            if (state == WifiP2pManager.WIFI_P2P_STATE_ENABLED) {
                activity.setIsWifip2pEnabled(true);
            } else {
                activity.setIsWifip2pEnabled(false);
                activity.resetData();
            }
        } else if (WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION.equals(action)) {
            if (manager != null) {
                manager.requestPeers(channel, (PeerListListener) activity.getFragmentManager()
                        .findFragmentById(R.id.fragment_list));
            }
        } else if (WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION.equals(action)) {
            if (manager == null) {
                return;
            }
            NetworkInfo networkInfo = (NetworkInfo) intent
                    .getParcelableExtra(WifiP2pManager.EXTRA_NETWORK_INFO);
            if (networkInfo.isConnected()) {
                DeviceDetailFragment fragment = (DeviceDetailFragment) activity
                        .getFragmentManager().findFragmentById(R.id.fragment_detail);
                manager.requestConnectionInfo(channel, fragment);
            } else {
                activity.resetData();
            }
        } else if (WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION.equals(action)) {
            DeviceListFragment fragment = (DeviceListFragment) activity.getFragmentManager()
                    .findFragmentById(R.id.fragment_list);
            fragment.updateThisDevice((WifiP2pDevice) intent.getParcelableExtra(
                    WifiP2pManager.EXTRA_WIFI_P2P_DEVICE));
        }
    }
}
```

Fig- The error is traced from the top to lower hierarchy of the files.

CHAPTER 8

CONCLUSION AND FUTURE SCOPE

Our project has reached the milestone as we designed to meet the requirements, The Android Application provides user friendly interface for users and Android application is in Good working conditions.

The primary goal of this project is to give an idea about file Sending among android mobile and Tablet with Wi-fi-Direct. This project has given us a depth information about java technology, j2ee and ANDROID technology and its applications in day today life. This project offers several benefits due to the use of simple and easy to understand interface. This project is developed using java and android platform, which is easy to read, learn, scales well, performs well for being interpreted, and is incredibly well- documented.

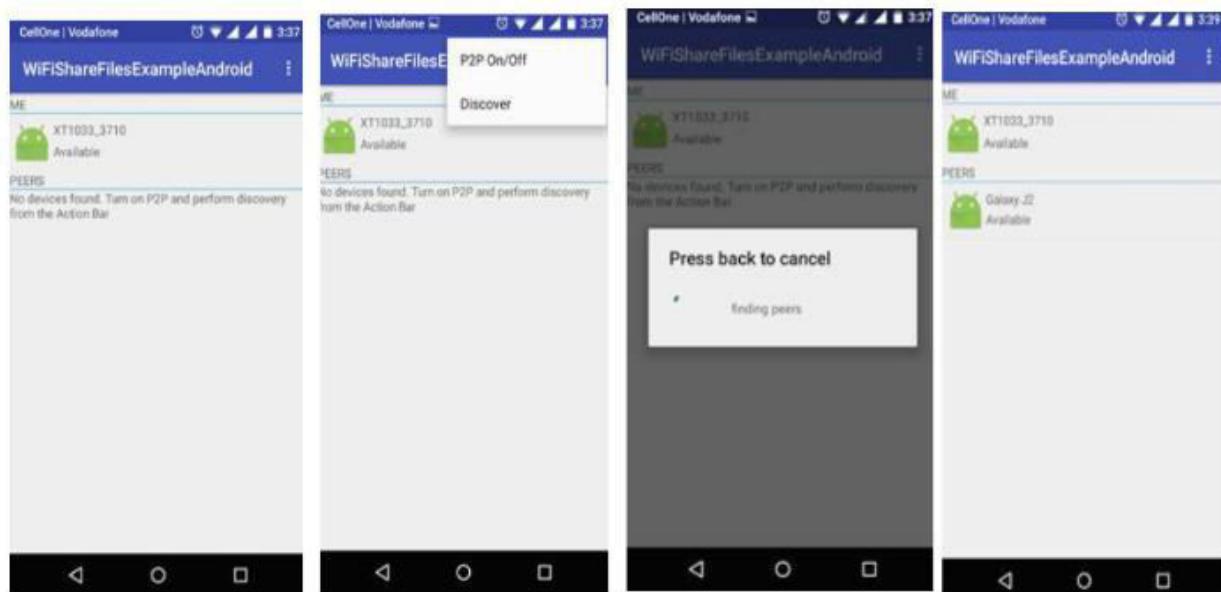
As the final result meet to our expectation of File sharing. We are expecting to bring this project to market with real-time cloud file sharing platform. The file sharing application and the platform is ready.

Future Scope

- It can extend into a cloud-based productivity platform like *Google Drive* and *Dropbox*.
- Transferring large files quicker by seeding process as seen in *Spotify* and *BitTorrent*.
- It can work as a built-in capability of an OS to share files among computers with the same OS.
- It can extend into a centralized File system like *IBM Aspera*.

6.1 Screenshots

6.1.1 Screenshot of User interface of Android Application

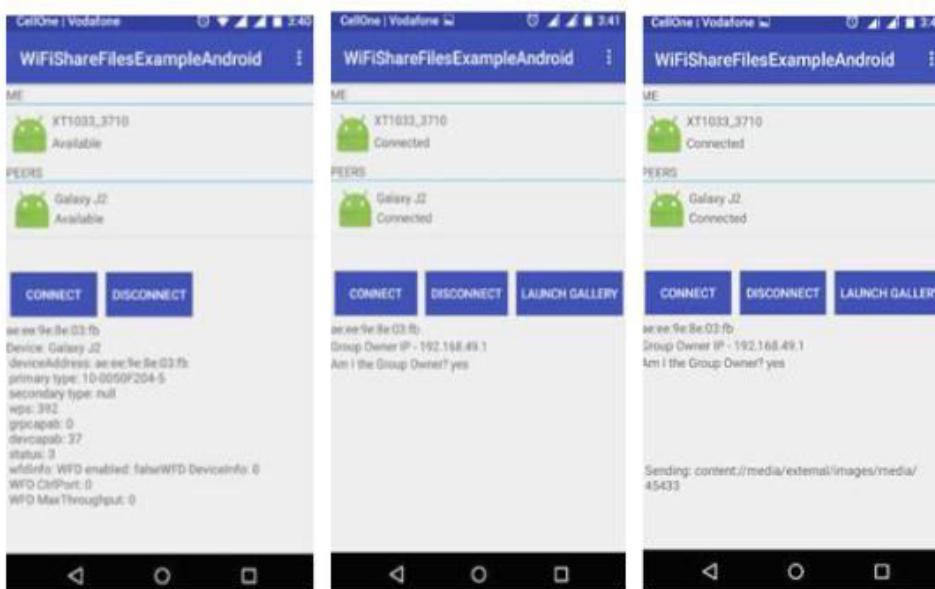


Before Discover

Perform Discover

Finding Peers

Peer Available



Connecting Device

Device Connected

Status after Sending the data successfully

6.1.2 Screenshots of Source code

```

<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:weightSum="1">

    <LinearLayout
        android:id="@+id/linearLayout"
        android:layout_width="match_parent"
        android:layout_height="8dp"
        android:layout_alignParentTop="true"
        android:layout_weight="0.9"
        android:orientation="vertical">

        <fragment
            android:id="@+id/fragment_list"
            class="com.example.shriyanshu.wifisharefilesexampleandroid.DeviceListFragment"
            android:layout_width="match_parent"
            android:layout_height="@dimen/phone_list_height"
            tools:layout="@layout/device_list"></fragment>

        <fragment
            android:id="@+id/fragment_detail"
            class="com.example.shriyanshu.wifisharefilesexampleandroid.DeviceDetailFragment"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            tools:layout="@layout/device_detail"></fragment>
    </LinearLayout>
</LinearLayout>

```

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help wifishareexampleandroid [C:\Users\dell\AndroidStudioProjects\wifishareexampleandroid2] - ...src\main\res\layout\device_detail.xml [app]

device_list.xml menu_main.xml device_detail.xml activity_main.xml DeviceListFragment.java Open Edit Run/Debug configurations' dialog Service.java WiFiClientPTransferServ + llz

Code Split Design

Project

Resource Manager

Java

com.example.shriyanshu.wifishare

- CommonMethods
- DeviceDetailFragment
- DeviceListFragment
- FileTransferService
- GlobalApplication
- MainActivity
- SharedPreferencesHandler
- WiFiClientPTransferService
- WiFiDirectBroadcastReceiver
- WiFiTransferModal
- com.example.shriyanshu.wifishare
- ExampleInstrumentedTest
- java (generated)
- com.example.shriyanshu.wifishare
- BuildConfig
- res
- drawable
- layout
- activity_main.xml
- device_list.xml
- row_devices.xml
- menu
- menu_main.xml
- values
- gradle Scripts
- build.gradle (Project: wifisharefilesx)
- build.gradle (Module: app)
- gradle-wrapper.properties (Gradle Version)
- monostandard-index.html (OneClient Rules)

z Favorites

z Build Variants

z Structure

z Build

z Logcat

z Terminal

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="fill_parent"
    android:visibility="gone">
    <LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
        <LinearLayout
            xmlns:android="http://schemas.android.com/apk/res/android"
            android:orientation="horizontal"
            android:layout_width="match_parent"
            android:layout_height="wrap_content">
            <Button
                android:id="@+id/btn_connect"
                android:layout_width="100dp"
                android:layout_height="wrap_content"
                android:text="@string/connect_peer_button"
                android:background="@color/colorPrimary"
                android:layout_margin="5dp"
                android:textColor="#ffffffff"/>
            <Button
                android:id="@+id/btn_disconnect"
                android:layout_width="100dp"
                android:layout_height="wrap_content"
                android:text="@string/disconnect_peer_button"
                android:background="@color/colorPrimary"
                android:layout_margin="5dp"
                android:textColor="#ffffffff"/>
        </LinearLayout>
    </LinearLayout>
</FrameLayout>
```

Event Log Layout Inspector

393 CRLF 4 spaces

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help wifishareexampleandroid [C:\Users\dell\AndroidStudioProjects\wifishareexampleandroid2] - ...app\src\main\res\layout\device_list.xml [app]

device_list.xml menu_main.xml device_detail.xml activity_main.xml DeviceListFragment.java row_devices.xml FileTransferService.java WiFiClientPTransferServ + llz

Code Split Design

Project

Resource Manager

Java

com.example.shriyanshu.wifishare

- CommonMethods
- DeviceDetailFragment
- DeviceListFragment
- FileTransferService
- GlobalApplication
- MainActivity
- SharedPreferencesHandler
- WiFiClientPTransferService
- WiFiDirectBroadcastReceiver
- WiFiTransferModal
- com.example.shriyanshu.wifishare
- ExampleInstrumentedTest
- java (generated)
- com.example.shriyanshu.wifishare
- BuildConfig
- res
- drawable
- layout
- activity_main.xml
- device_detail.xml
- device_list.xml
- row_devices.xml
- menu
- menu_main.xml
- values
- gradle Scripts
- build.gradle (Project: wifisharefilesx)
- build.gradle (Module: app)
- gradle-wrapper.properties (Gradle Version)
- monostandard-index.html (OneClient Rules)

z Favorites

z Build Variants

z Build

z Logcat

z Terminal

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingTop="10dp">
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="center_vertical"
        android:text="My Device List" />
    <View
        android:layout_width="fill_parent"
        android:layout_height="1dp"
        android:gravity="center_vertical"
        android:background="#android:color/white_blink_light" />
    <!-- Self Information -->
    <LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="android:attr/listPreferredItemHeight"
        android:background="#android:attr/activatedBackgroundIndicator"
        android:padding="10dp">
        <ImageItem
            android:id="@+id/icon_Icon"
            android:layout_width="wrap_content"
            android:layout_height="fill_parent"
            android:layout_marginRight="10dp"
            android:src="@mipmap/ic_launcher" />
        <LinearLayout
            android:orientation="vertical"
            android:layout_width="0dp"
            android:layout_weight="1"
            android:layout_height="fill_parent">
            <TextItem
                android:id="@+id/rv_myName"
                android:layout_width="fill_parent"
                android:layout_height="0dp"
                android:layout_weight="1"
                android:layout_marginBottom="5dp" />
```

Event Log Layout Inspector

64 CRLF 4 spaces

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="?android:attr/listPreferredItemHeight"
    android:background="?android:attr/activatedBackgroundIndicator"
    android:padding="6dp">
    <ImageView
        android:id="@+id/sv_Icon"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_marginRight="2dp"
        android:src="@mipmap/ic_launcher" />
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="0dp"
        android:layout_weight="1"
        android:layout_height="fill_parent">
        <TextView
            android:id="@+id/tv_deviceName"
            android:layout_width="fill_parent"
            android:layout_height="0dp"
            android:layout_weight="1"
            android:gravity="center_vertical" />
        <TextView
            android:layout_width="fill_parent"
            android:layout_height="0dp"
            android:layout_weight="1"
            android:id="@+id/tv_deviceDetails"
            android:maxLines="1"
            android:ellipsize="marquee" />
    </LinearLayout>
</LinearLayout>
```

The screenshot shows the Android Studio interface with the following details:

- Project Tree:** The project structure is visible on the left, showing packages like `app`, `samplerdata`, `manifests`, and `java`. Under `java`, there are several files including `DeviceDetailFragment`, `DeviceListFragment`, `FileTransferService`, `GlobalApplication`, `MainActivity`, `SharedPreferencesHandler`, `WifiClientPTTransferService`, `WifiDirectBroadcastReceiver`, and `WifiTransferModal`. A test file `ExampleInstrumentedTest` is also present.
- Code Editor:** The main window displays the `DeviceDetailFragment.java` file. The code is as follows:

```
1 package com.example.shriyanshu.wifisharefilesexampleandroid;
2
3 import android.app.Fragment;
4 import android.app.ProgressDialog;
5 import android.content.Context;
6 import android.content.Intent;
7 import android.net.Uri;
8 import android.net.wifi.WpsInfo;
9 import android.net.wifi.p2p.WifiP2pConfig;
10 import android.net.wifi.p2p.WifiP2pDevice;
11 import android.net.wifi.p2p.WifiP2pInfo;
12 import android.net.wifi.p2p.WifiP2pManager.ConnectionInfoListener;
13 import android.os.AsyncTask;
14 import android.os.Bundle;
15 import android.os.Environment;
16 import android.os.Handler;
17 import android.util.Log;
18 import android.view.LayoutInflater;
19 import android.view.View;
20 import android.view.ViewGroup;
21 import android.widget.TextView;
22 import android.widget.Toast;
23 import com.example.shriyanshu.wifisharefilesexampleandroid.DeviceListFragment.DeviceActionListener;
24 import java.io.File;
25 import java.io.FileOutputStream;
26 import java.io.IOException;
27 import java.io.InputStream;
28 import java.io.ObjectInputStream;
29 import java.io.OutputStream;
30 import java.net.ServerSocket;
31 import java.net.Socket;
32
33 import static com.example.shriyanshu.wifisharefilesexampleandroid.R.id.tv_deviceInfo;
34 public class DeviceDetailFragment extends Fragment implements ConnectionInfoListener {
35     protected static final int CHOOSE_FILE_RESULT_CODE = 20;
36     private View view;
37     private WifiP2pDevice device;
38     private WifiP2pInfo info;
39     ProgressDialog progressDialog = null;
40     private static ProgressDialog mProgressDialog;
41     public static String WiFiServerIP = "";
42     public static String WiFiClientsIP = "";
43 }
```

The code includes imports for various Android components and Java IO classes. It defines a `DeviceDetailFragment` that extends `Fragment` and implements `ConnectionInfoListener`. It uses static variables for WiFi server and client IP addresses. The `onCreateView` method is partially visible at the bottom.



The screenshot shows the Android Studio interface with the following details:

- Project Structure:** The project is named "exampleandroid2" and contains an "app" module. The "app" module has an "src/main/java/com/example/shriyanshu/wifishare" package where the "DeviceDetailFragment" class is located.
- Code Editor:** The editor displays the "DeviceDetailFragment.java" file. The code implements the "View" interface and extends "AppCompatActivity". It includes methods for handling device connection and configuration.
- Toolbars and Menus:** Standard Android Studio toolbars and menus like File, Edit, View, Navigate, Code, Analyze, Refactor, Build, Run, Tools, VCS, Window, Help, and Device Manager are visible at the top.
- Side Panels:** The Resource Manager, Structure, and Logcat panels are visible on the left and bottom right respectively.

```
private void setupViewPager(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
    ProgressDialog progressDialog = null;
    private static ProgressDialog mProgressDialog;
    public static String WiFiserverIP = "";
    public static String WiFiClientIP = "";
    static Boolean ClientCheck = false;
    public static String GroupHeaderAddress = "";
    static long ActualFileLength = 0;
    static int Percentage = 0;
    public static String FolderName = "WiFiShareFilesExampleAndroid";

    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        setContentView = inflater.inflate(R.layout.device_detail, null);
        setContentView.findViewById(R.id.btn_connect).setOnClickListener(v -> {
            WiFiP2pConfig config = new WiFiP2pConfig();
            config.deviceAddress = device.deviceAddress;
            config.wps.setupInfo.PBC;
            if (progressDialog != null && progressDialog.isShowing()) {
                progressDialog.dismiss();
            }
            progressDialog = ProgressDialog.show(getActivity(), "Press back to cancel", "Connecting to :" + device.deviceAddress, indeterminate: true, cancelable: true
            );
            ((DeviceActionListener) getActivity()).connect(config);
        });
        setContentView.findViewById(R.id.btn_disconnect).setOnClickListener(
            v -> {
                ((DeviceActionListener) getActivity()).disconnect();
            });
        setContentView.findViewById(R.id.btn_start_client).setOnClickListener(
            v -> {

```

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help wifisharefilesexampleandroid [C:\Users\dell\AndroidStudioProjects\wifisharefilesexampleandroid2] - DeviceListFragment.java [app]

```

device_list.xml  menu_main.xml  device_detail.xml  activity_main.xml  DeviceListFragment.java  row_devices.xml  CommonMethods.java  DeviceDetailFragment.java
1 package com.example.shriyanshu.wifisharefilesexampleandroid;
2 import android.app.ListFragment;
3 import android.app.ProgressDialog;
4 import android.content.Context;
5 import android.net.IWIFI_P2P_DEVICE_INTERFACE;
6 import android.net.wifi.p2p.WifiP2pConfig;
7 import android.net.wifi.p2p.WifiP2pDevice;
8 import android.net.wifi.p2p.WifiP2pDeviceList;
9 import android.net.wifi.p2p.WifiP2pManager.PeerListListener;
10 import android.os.Bundle;
11 import android.os.Handler;
12 import android.view.LayoutInflater;
13 import android.view.View;
14 import android.view.ViewGroup;
15 import android.widget.AdapterView;
16 import android.widget.ListView;
17 import android.widget.TextView;
18 import java.util.ArrayList;
19 import java.util.List;
20
21 public class DeviceListFragment extends ListFragment implements PeerListListener {
22     private LayoutInflater mInflater = new LayoutInflater(getActivity());
23     ProgressDialog progressDialog = null;
24     ViewGroup container = null;
25     private WiFip2pDevice device;
26     Handler mHandler;
27
28     public void onCreate(Bundle savedInstanceState) {
29         super.onCreate(savedInstanceState);
30         this.setListAdapter(new WiFip2pListAdapter(getActivity(), R.layout.row_devices, peers));
31     }
32
33     public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
34         container.addView(inflater.inflate(R.layout.device_list, null));
35         return container;
36     }
37
38     public WiFip2pDevice getDevice() {
39         return device;
40     }
41
42     private static String getDeviceStatus(int deviceStatus) {
43         switch (deviceStatus) {
44             case WiFip2pDevice.DISCOVERABLE:
45                 return "Available";
46             case WiFip2pDevice.INVITED:
47                 return "Invited";
48             case WiFip2pDevice.CONNECTED:
49                 return "Connected";
50             case WiFip2pDevice.FAILED:
51                 return "Failed";
52             case WiFip2pDevice.UNAVAILABLE:
53                 return "Unavailable";
54             default:
55                 return "Unknown";
56         }
57     }
58
59     @Override
60     public void onListItemClick(ListView l, View v, int position, long id) {
61         WiFip2pDevice device = (WiFip2pDevice) getListAdapter().getItem(position);
62         (new IntentListener()).getActivity().showDetails(device);
63     }
64
65     private class WiFip2pListAdapter extends ArrayAdapter<WiFip2pDevice> {
66
67         public WiFip2pListAdapter(Context context, int resource, List<WiFip2pDevice> devices) {
68             super(context, resource, devices);
69         }
70
71         @Override
72         public View getView(int position, View convertView, ViewGroup parent) {
73             ViewHolder holder;
74             if (convertView == null) {
75                 convertView = mInflater.inflate(R.layout.row_devices, parent, false);
76                 holder = new ViewHolder();
77                 convertView.setTag(holder);
78             } else {
79                 holder = (ViewHolder) convertView.getTag();
80             }
81
82             WiFip2pDevice device = getItem(position);
83             holder.name.setText(device.getDeviceName());
84             holder.ipAddress.setText(device.getIpAddress());
85             holder.status.setText(getDeviceStatus(device.getDeviceStatus()));
86
87             return convertView;
88         }
89
90         static class ViewHolder {
91             TextView name;
92             TextView ipAddress;
93             TextView status;
94         }
95     }
96 }

```

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help wifisharefilesexampleandroid [C:\Users\dell\AndroidStudioProjects\wifisharefilesexampleandroid2] - FileTransferService.java [app]

```

device_list.xml  menu_main.xml  device_detail.xml  activity_main.xml  DeviceListFragment.java  FileTransferService.java  row_devices.xml  CommonMethods.java  DeviceDetailFragment.java
1 package com.example.shriyanshu.wifisharefilesexampleandroid;
2
3 import android.app.IntentService;
4 import android.content.ContentResolver;
5 import android.content.Context;
6 import android.content.Intent;
7 import android.net.IWIFI_P2P_SERVICE_INTERFACE;
8 import android.os.Handler;
9 import android.widget.Toast;
10 import java.io.FileNotFoundException;
11 import java.io.IOException;
12 import java.io.InputStream;
13 import java.io.ObjectOutputStream;
14 import java.io.OutputStream;
15 import java.net.InetSocketAddress;
16 import java.net.Socket;
17
18 public class FileTransferService extends IntentService {
19
20     Handler mHandler;
21
22     public static final int SOCKET_TIMEOUT = 5000;
23     public static final String ACTION_SEND_FILE = "com.example.shriyanshu.wifisharefilesexampleandroid.SEND_FILE";
24     public static final String EXTRAS_FILE_PATH = "file_url";
25     public static final String EXTRAS_GROUP_OWNER_ADDRESS = "go_host";
26     public static final String EXTRAS_GROUP_OWNER_PORT = "go_port";
27
28     public static final int PORT = 8888;
29     public static final String netaddress = "inetaddress";
30     public static final int bytesize = 512;
31     public static final String Extension = "extension";
32     public static final String filength = "filength";
33
34     public FileTransferService(String name) {
35         super(name);
36     }
37
38     public FileTransferService() {
39         super("FileTransferService");
40     }
41
42     @Override
43     public void onCreate() {
44         // TODO Auto-generated method stub
45         super.onCreate();
46         mHandler = new Handler();
47     }
48
49     public void transferFile() {
50         Intent intent = new Intent(ACTION_SEND_FILE);
51         intent.putExtra(EXTRAS_FILE_PATH, "http://192.168.43.133/test.pdf");
52         intent.putExtra(EXTRAS_GROUP_OWNER_ADDRESS, "192.168.43.133");
53         intent.putExtra(EXTRAS_GROUP_OWNER_PORT, 8888);
54         sendBroadcast(intent);
55     }
56
57     protected void onHandleIntent(Intent intent) {
58         String fileurl = intent.getStringExtra(EXTRAS_FILE_PATH);
59         String host = intent.getStringExtra(EXTRAS_GROUP_OWNER_ADDRESS);
60         int port = intent.getIntExtra(EXTRAS_GROUP_OWNER_PORT, 8888);
61
62         try {
63             Socket socket = new Socket(host, port);
64             OutputStream os = socket.getOutputStream();
65             InputStream is = socket.getInputStream();
66
67             byte[] buffer = new byte[bytesize];
68             int length;
69             while ((length = is.read(buffer)) > 0) {
70                 os.write(buffer, 0, length);
71             }
72             os.close();
73             is.close();
74             socket.close();
75         } catch (IOException e) {
76             e.printStackTrace();
77         }
78     }
79
80 }

```

The screenshot shows the Android Studio interface with the following details:

- Project Structure:** The left sidebar shows the project structure under "filesexampleandroid2".
- Editor:** The main editor window displays the code for `GlobalApplication.java`. The code defines a global application context and provides a static method to get it.

```
1 package com.example.shriyanshu.wifisharefilesexampleandroid;
2
3 import android.app.Application;
4 import android.content.Context;
5
6 public class GlobalApplication extends Application {
7     private static Context GlobalContext;
8
9     @Override
10    public void onCreate() {
11        // TODO Auto-generated method stub
12        super.onCreate();
13        if(GlobalApplication.GlobalContext == null){
14            GlobalApplication.GlobalContext = getApplicationContext();
15        }
16    }
17
18    public static Context getGlobalAppContext() {
19        return GlobalApplication.GlobalContext;
20    }
21 }
```

- Toolbars and Status Bar:** The top bar includes standard Android Studio icons for File, Edit, View, Navigate, Code, Analyze, Refactor, Build, Run, Tools, VCS, Window, Help, and various tabs like device_list.xml, menu_main.xml, device_detail.xml, activity_main.xml, DeviceListFragment.java, FileTransferService.java, GlobalApi, Run 'app' with Coverage, and services.xml.
- Bottom Bar:** Includes tabs for Event Log, Layout Inspector, and Terminal, along with status indicators for CRLF, UTF-8, 4 spaces, and a selected configuration.

The screenshot shows the Android Studio interface with the following details:

- Project Bar:** File, Edits, View, Navigate, Code, Analyze, Refactor, Build, Run, Tools, VCS, Window, Help, wifisharefilesexampleandroid [C:\Users\ dell\AndroidStudioProjects\wifisharefilesexampleandroid2] - SharedPreferencesHandler.java [app]
- Toolbars:** Top toolbar with various icons for file operations.
- Left Sidebar:** Project Manager, Resource Manager, Build Variants, and Favorites.
- Code Editor:** The main window displays the `SharedPreferencesHandler.java` file under the `app` module's `java` package. The code implements methods for getting and setting shared preferences.

```
1 package com.example.shriyanhu.wifisharefilesexampleandroid;
2
3 import android.content.Context;
4 import android.content.SharedPreferences;
5 import android.content.SharedPreferences.Editor;
6 import android.preference.PreferenceManager;
7
8 public class SharedPreferencesHandler {
9
10     public static SharedPreferences getSharedPreferences(Context ctx) {
11         return PreferenceManager.getDefaultSharedPreferences(ctx);
12     }
13
14     public static void setStringValues(Context ctx, String key,
15             String DataToSave) {
16         Editor editor = getSharedPreferences(ctx).edit();
17         editor.putString(key, DataToSave);
18         editor.commit();
19     }
20
21     public static String getStringValues(Context ctx, String key) {
22         return getSharedPreferences(ctx).getString(key, null);
23     }
24
25 }
26
```

- Bottom Status Bar:** Shows build statistics, event log, and layout inspector buttons.

The screenshot shows the Android Studio interface with the project navigation bar at the top. The main area displays the code for `WiFiClientIPTransferService.java`. The code implements an `IntentService` for handling file transfers over WiFi. It includes imports for various Android classes and interfaces, and defines methods for handling intents and establishing network connections.

```
package com.example.shriyanshu.wifisharefilesexampleandroid;

import android.app.IntentService;
import android.content.ContentResolver;
import android.content.Context;
import android.os.Handler;
import java.io.IOException;
import java.io.InputStream;
import java.io.ObjectOutputStream;
import java.io.OutputStream;
import java.net.InetSocketAddress;
import java.net.Socket;

public class WiFiClientIPTransferService extends IntentService {
    public WiFiClientIPTransferService(String name) {
        super(name);
        // TODO Auto-generated constructor stub
    }
    public WiFiClientIPTransferService() {
        super( name: "WiFiClientIPTransferService");
    }
    Handler mHandler;
    @Override
    protected void onHandleIntent(Intent intent) {
        Context context = GlobalApplication.getGlobalApplicationContext();
        if (intent.getAction().equals(FileTransferService.ACTION_SEND_FILE)) {
            String host = intent.getExtras().getString(FileTransferService.EXTRAS_GROUP_OWNER_ADDRESS);
            String inetAddress = intent.getExtras().getString(FileTransferService.inetAddress);
            CommonMethods.e(tag: "Localip Received while first connect", msg: "host address"+ host);

            Socket socket = new Socket();
            int port = intent.getExtras().getInt(FileTransferService.EXTRAS_GROUP_OWNER_PORT);

            try {
                
```

The screenshot shows the Android Studio interface with the following details:

- Project Structure:** The project is named "wifisharefilesexampleandroid". The "app" module contains Java files like FileTransferService.java, SharedPreferencesHandler.java, GlobalApplication.java, MainActivity.java, SharedPreferencesHandle.java, WiFiClientPTTransferService.java, WiFiDirectBroadcastReceiver.java, WiFiTransferModal.java, and several XML files for layouts and values.
- Code Editor:** The main focus is on the WiFiDirectBroadcastReceiver.java file. The code handles WiFi P2P state changes and peer list updates. It uses BroadcastReceiver, WiFiP2pManager, and MainActivity.
- Toolbars and Status Bar:** The top bar shows the file path and the message "File sync finished in 16 s 966 ms (from cached state) (yesterday 1643)". The bottom status bar shows "294 CRLF UTF-8 4 spaces" and icons for Event Log, Layout Inspector, and Terminal.

```
package com.example.shriyanshu.wifisharefilesexampleandroid;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.net.NetworkInfo;
import android.net.wifi.p2p.WifiP2pDevice;
import android.net.wifi.p2p.WifiP2pManager;
import android.net.wifi.p2p.WifiP2pManager.Channel;
import android.net.wifi.p2p.WifiP2pManager.PeerListListener;

public class WiFiDirectBroadcastReceiver extends BroadcastReceiver {
    private WifiP2pManager manager;
    private Channel channel;
    private MainActivity activity;
    public WiFiDirectBroadcastReceiver(WifiP2pManager manager, Channel channel,
                                       MainActivity activity) {
        super();
        this.manager = manager;
        this.channel = channel;
        this.activity = activity;
    }
    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if (WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION.equals(action)) {
            int state = intent.getIntExtra(WifiP2pManager.EXTRA_WIFI_STATE, -1);
            if (state == WifiP2pManager.WIFI_P2P_STATE_ENABLED) {
                activity.setIsWiFiP2pEnabled(true);
            } else {
                activity.setIsWiFiP2pEnabled(false);
                activity.resetData();
            }
        } else if (WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION.equals(action)) {
            if (manager != null) {
                manager.getPeers(new PeerListListener() {
                    @Override
                    public void onPeersFound(List<WifiP2pDevice> peers, int numPeers) {
                        // Handle peer list update
                    }
                });
            }
        }
    }
}
```

The screenshot shows the Android Studio interface with the following details:

- Project Tree:** The left sidebar displays the project structure under "com.example.shriyanshu.wifishare". It includes modules like app, sampledata, manifests, and java. The java module contains packages com.example.shriyanshu.wifishare and com.example.shriyanshu.wifishareexample. The com.example.shriyanshu.wifishareexample package contains classes SharedPreferencesHandler.java, GlobalApplication.java, WiFiClientPTTransferService.java, WiFiDirectBroadcastReceiver.java, and WiFiTransferModal.java.
- Code Editor:** The main window shows the source code for WiFiTransferModal.java. The code defines a class WiFiTransferModal that implements Serializable. It has private fields for fileName, fileLength, and InetAddress, and a constructor that takes an InetAddress. It also has methods to get and set the InetAddress and to get the file length.
- Toolbars and Status Bar:** The top bar shows standard file menu options (File, Edit, View, Navigate, Code, Analyze, Refactor, Build, Run, Tools, VCS, Window, Help) and the current file path (wifisharefilesexampleandroid [C:\Users\deffl\AndroidStudioProjects\wifisharefilesexampleandroid2] -> WiFiTransferModal.java [app]). The bottom status bar indicates "Gradle sync finished in 16 s 966 ms (from cached state) (yesterday 1643)" and shows icons for Event Log, Layout Inspector, and Terminal.

The screenshot shows the Android Studio interface with the following details:

- Project Structure:** The project is named "wifisharefilesexampleandroid". It contains an "app" module with Java files like SharedPreferencesHandler.java, GlobalApplication.java, WiFiClientPTTransferService.java, WiFiDirectBroadcastReceiver.java, WiFiTransferModal.java, and ExampleInstrumentedTest.java.
- Code View:** The current file is ExampleInstrumentedTest.java. The code is as follows:

```
1 package com.example.shriyanшу.wifisharefilesexampleandroid;
2
3 import android.content.Context;
4
5 import androidx.test.platform.app.InstrumentationRegistry;
6 import androidx.test.ext.junit.runners.AndroidJUnit4;
7
8 import org.junit.Test;
9 import org.junit.runner.RunWith;
10
11 import static org.junit.Assert.*;
12
13 /**
14 * Instrumented test, which will execute on an Android device.
15 *
16 * Testing documentation
17 */
18 @RunWith(AndroidJUnit4.class)
19 public class ExampleInstrumentedTest {
20     @Test
21     public void useAppContext() {
22         // Context of the app under test.
23         Context appContext = InstrumentationRegistry.getInstrumentation().getTargetContext();
24         assertEquals("com.example.shriyanшу.wifisharefilesexampleandroid", appContext.getPackageName());
25     }
26 }
```

The code is annotated with Javadoc-style comments and uses static imports from the org.junit package. The `useAppContext()` method tests the application's context.

The screenshot shows the Android Studio interface with the project 'wifisharefilesexampleandroid2' open. The left sidebar displays the project structure, including the app module with Java files like MainActivity.java, DeviceListFragment.java, FileTransferService.java, SharedPreferencesHandler.java, GlobalApplication.java, WiFiClientPTTransferService.java, WiFiDirectBroadcastReceiver.java, WiFiTransferModal.java, and others. Below these are com.example.shriyanshu.wifishare and com.example.shriyanshu.wifisharetest packages. The build.gradle and build.gradle (Module: app) files are also visible. The bottom navigation bar includes tabs for TODO, Build, Logcat, and Terminal.

```
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help wifisharefilesexampleandroid [C:/Users/dell/AndroidStudioProjects/wifisharefilesexampleandroid2] - .../MainActivity.java [app]
```

```
src/main/java/com/example/shriyanshu/wifishare
```

```
    MainActivity.java DeviceListFragment.java FileTransferService.java SharedPreferencesHandler.java GlobalApplication.java WiFiClientPTTransferService.java WiFiDirectBroadcastReceiver.java WiFiTransferModal.java
```

```
    com.example.shriyanshu.wifishare
```

```
        ExampleInstrumentedTest
```

```
    com.example.shriyanshu.wifisharetest
```

```
java (generated)
```

```
com.example.shriyanshu.wifishare
```

```
    BuildConfig
```

```
res
```

```
    drawable
```

```
    layout
```

```
        activity_main.xml
```

```
        device_detail.xml
```

```
        device_list.xml
```

```
        row_devices.xml
```

```
    menu
```

```
        menu_main.xml
```

```
        mipmap
```

```
        values
```

```
Gradle Scripts
```

```
    build.gradle (Project: wifisharefilesexampleandroid2)
```

```
    build.gradle (Module: app)
```

```
    gradle-wrapper.properties (Gradle Version: 4.4)
```

```
    .gradle/wrapper/gradle-wrapper.jar
```

```
    .gradle/wrapper/maven-wrapper.jar
```

```
    .gradle/wrapper/native-wrapper.jar
```

```
    .gradle/wrapper/resources.jar
```

```
    .gradle/wrapper/tools.jar
```

```
    .gradle/wrapper/wrapper.jar
```

```
    .gradle/wrapper/wrapper.properties
```

```
MainActivity
```

```
    @Override
```

```
    public void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_main);
```

```
        // add necessary intent values to be matched.
```

```
        IntentFilter intentFilter = new IntentFilter();
```

```
        intentFilter.addAction(WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION);
```

```
        intentFilter.addAction(WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION);
```

```
        intentFilter.addAction(WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION);
```

```
        intentFilter.addAction(WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION);
```

```
        manager = (WifiP2pManager) getSystemService(Context.WIFI_P2P_SERVICE);
```

```
        channel = manager.initialize(getApplicationContext(), null);
```

```
}
```

```
    /** register the BroadcastReceiver with the intent values to be matched */
```

```
    @Override
```

```
    public void onResume() {
```

```
        super.onResume();
```

```
        receiver = new WiFiDirectBroadcastReceiver(manager, channel, this);
```

```
        registerReceiver(receiver, intentFilter);
```

```
}
```

```
    @Override
```

```
    public void onPause() {
```

```
        super.onPause();
```

```
        unregisterReceiver(receiver);
```

```
}
```

```
    /**
```

```
     * Remove all peers and clear all fields. This is called on
```

```
     * BroadcastReceiver receiving a state change event.
```

```
     */
```

```
    public void resetData() {
```

```
        DeviceListFragment fragmentList = (DeviceListFragment) getFragmentManager()
```

```
            .findFragmentById(R.id.fragment_main);
```

```

    * Remove all peers and clear all fields. This is called on
    * BroadcastReceiver receiving a state change event.
    */
    public void reloadData() {
        DeviceListFragment fragmentList = (DeviceListFragment) getSupportFragmentManager()
            .findFragmentById(R.id.fragment_list);
        DeviceDetailFragment fragmentDetails = (DeviceDetailFragment) getSupportFragmentManager()
            .findFragmentById(R.id.fragment_detail);

        if (fragmentList != null) {
            fragmentList.clearPeers();
        }

        if (fragmentDetails != null) {
            fragmentDetails.resetViews();
        }
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.menu_main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.action_directEnable:
                if (manager != null & channel != null) {

                    // Since this is the system wireless settings activity, it's
                    // not going to send us a result. We will be notified by
                    // WiFiDeviceBroadcastReceiver instead.

                    startActivity(new Intent(Settings.ACTION_WIRELESS_SETTINGS));
                }
                break;
        }
        return super.onOptionsItemSelected(item);
    }
}

```

```

    startActivity(new Intent(Settings.ACTION_WIRELESS_SETTINGS));
} else {
    //Log.e(TAG, "channel or manager is null");
}

return true;

case R.id.action_directDiscover:
    if (!isWifiP2pEnabled) {
        Toast.makeText(context: MainActivity.this, "Enable P2P from action bar button above or system settings",
            Toast.LENGTH_SHORT).show();
    }
    return true;
}

final DeviceListFragment fragment = (DeviceListFragment) getSupportFragmentManager()
    .findFragmentById(R.id.fragment_list);
fragment.onInitiateDiscovery();
manager.discoverPeers(channel, new WifiP2pManager.ActionListener() {
    ...
}

@Override
public void onSuccess() {
    Toast.makeText(context: MainActivity.this, text: "Discovery Initiated",
        Toast.LENGTH_SHORT).show();
}

@Override
public void onFailure(int reasonCode) {
    Toast.makeText(context: MainActivity.this, text: "Discovery Failed : " + reasonCode,
        Toast.LENGTH_SHORT).show();
}
});

return true;
default:
    return super.onOptionsItemSelected(item);
}
}

```

The screenshot shows the Android Studio interface with the project structure on the left and the code editor on the right. The code editor displays the `MainActivity.java` file, which contains Java code for managing a P2P connection. The code includes methods for handling connection attempts and disconnecting from devices.

```
// we will try once more
if (manager != null && !retryChannel) {
    Toast.makeText(context, "Channel lost. Trying again", Toast.LENGTH_LONG).show();
    reloadData();
    retryChannel = true;
    manager.initialize(getApplicationContext(), listener: this);
} else {
    Toast.makeText(context, "Severe! Channel is probably lost permanently. Try Disable/Re-Enable P2P.", Toast.LENGTH_LONG).show();
}

@Override
public void cancelDisconnect() {
    if (manager != null) {
        final DeviceListFragment fragment = (DeviceListFragment) getSupportFragmentManager()
            .findFragmentById(R.id.fragment_list);
        if (fragment.getDevice() == null
            || fragment.getDevice().status == WifiP2pDevice.CONNECTED) {
            disconnect();
        } else if (fragment.getDevice().status == WifiP2pDevice.AVAILABLE
            || fragment.getDevice().status == WifiP2pDevice.INVITED) {

            manager.cancelConnect(channel, new ActionListener() {

                @Override
                public void onSuccess() {
                    Toast.makeText(context, "Aborting connection", Toast.LENGTH_SHORT).show();
                }

                @Override
                public void onFailure(int reasonCode) {
                    Toast.makeText(context, "Error", Toast.LENGTH_SHORT).show();
                }
            });
        }
    }
}
```

[GitHub Link](https://github.com/Nehaadnekar/File-Sharing-App-MINI-PROJECT) : <https://github.com/Nehaadnekar/File-Sharing-App-MINI-PROJECT>

References

1. Solo-learn :- www.sololearn.com
2. Google Scholar :- www.scholar.google.com
3. W3schools :- www.w3schools.com
4. JavaTpoint :-www.javatpoint.com
5. Youtube :- www.youtube.com
- 6- StackOverFlow :- www.stackoverflow.com

Other Links -

- <https://nevnonprojects.com/secure-file-sharing-using-access-control/>
- <https://freeprojectsforall.com/an046-secure-file-sharing-using-access-control/>
- https://ijarcce.com/wp-content/uploads/2012/03/IJARCCE5G_a_shailes_secured.pdf
- <http://developer.android.com/guide/topics/connectivity/wifip2p.html>
- http://anrg.usc.edu/ee579_2012/Group09/#wifidirect
- Beginning Java forum, <http://www.coderanch.com/>
- How to switch between intents, <http://www.androidhive.info> /2011/08/ 23rd November, 2013.
- [http://developer.android.com/develop/index.html,](http://developer.android.com/develop/index.html)