

Direct

A LIBRARY TO SIMPLIFY THE ANDROID WI-FI P2P API

TYLER MCBRIDE | 100888344

<https://github.com/tylerjmcbride/Direct>

Contents

INTRODUCTION	2
API.....	3
DirectBroadcastReceiver.....	3
Direct	5
Host.....	7
Host.....	7
startService	7
Client.....	8
Client	8
startDiscovery	8
stopDiscovery.....	8
connect	9
disconnect.....	9
send.....	10
CONCLUSION	11

INTRODUCTION

Direct is a library that provides a simplified interface to wrap around the Wi-Fi Direct API. In essence, this interface acts as a facade around the Wi-Fi Direct API by hiding its implementation details.

Despite support of Wi-Fi Direct being released on Android 4.0 in October 2011, the Wi-Fi Direct API continues to be very complex and difficult to understand. Many developers steer clear of Wi-Fi Direct due to its complex nature and confusing documentation. This interface attempts to reduce these deterrents from preventing the use of Wi-Fi Direct in future android development.

Wi-Fi Direct has peer-to-peer functionality, hence initially being called Wi-Fi Peer-to-Peer; however, this library will be designed to support a client server architecture instead. In order to establish connections and send data amongst these connections, Wi-Fi Direct provides an abundance of details that are of little use to the developer.

This library abstracts this abundance of details in order to enable the developer to implement Wi-Fi Direct without having to be aware of the underlying functionality; thus, reducing the both the technical load on the developer and the potential for bugs. Ultimately, this library frees the developer of the specific details of the Wi-Fi Direct API.

API

This section will explore in depth the underlying functionality that each class within the library provides. Before reading the following documentation, it is important to understand that all of the Wi-Fi P2P framework methods are one way communication; in other words, any method called with the Wi-Fi P2P framework is only a request, and it's success only reflects the success of the hardware receiving said request, not that the request has been fulfilled. The state of the hardware is only available through the [DirectBroadcastReceiver](#).

For example, calling [connect](#) from the [WifiP2pManager](#) will only send the hardware the request to connect to the respective [WifiP2pDevice](#). The only way to determine if the connection has been establish is through the [DirectBroadcastReceiver](#) receiving a [WIFI_P2P_CONNECTION_CHANGED_ACTION](#) intent.

DirectBroadcastReceiver

`abstract class DirectBroadcastReceiver extends BroadcastReceiver`

The [DirectBroadcastReceiver](#) class extends [BroadcastReceiver](#), which receives and handles broadcasts intents. The [BroadcastReceiver](#) requires the extending class to implement [onReceive\(\)](#); however, this method generally promotes the If-Then-Else code smell, so the [DirectBroadcastReceiver](#) splits the [onReceive\(\)](#) method into several more intuitive methods namely:

- `stateChanged(boolean wifiEnabled)`
 - This method is called when the [WIFI_P2P_STATE_CHANGED_ACTION](#) intent has been broadcasted.
- `discoveryChanged(boolean discoveryEnabled)`
 - This method is called when the [WIFI_P2P_DISCOVERY_CHANGED_ACTION](#) intent has been broadcasted.
- `peersChanged()`
 - This method is called when the [WIFI_P2P_PEERS_CHANGED_ACTION](#) intent has been broadcasted.
- `connectionChanged(WifiP2pInfo p2pInfo, NetworkInfo info, WifiP2pGroup p2pGroup)`
 - This method is called when the [WIFI_P2P_CONNECTION_CHANGED_ACTION](#) intent has been broadcasted.
- `thisDeviceChanged(WifiP2pDevice thisDevice)`

- This method is called when the `WIFI_P2P_THIS_DEVICE_CHANGED_ACTION` intent has been broadcasted.

Direct

`abstract class Direct`

The [Direct](#) abstract class contains common behaviour and variables that are inherited by both the [Host](#) and [Client](#) classes. In particular this class creates an instance of [IntentFilter](#) with listens for the following intent actions:

- [WIFI_P2P_STATE_CHANGED_ACTION](#)
 - This action indicates whether Wi-Fi P2P is enabled or disabled.
- [WIFI_P2P_DISCOVERY_CHANGED_ACTION](#)
 - This action indicates whether the peer discovery has either been started or stopped.
- [WIFI_P2P_PEERS_CHANGED_ACTION](#)
 - This action indicates that the available peer list has changed. The peer list will be changed when peers are lost, found, or updated. This will be exclusively used by the host to unregister clients who have been lost.
- [WIFI_P2P_CONNECTION_CHANGED_ACTION](#)
 - This action indicates that the Wi-Fi P2P connectivity has changed. This will be used to get a handle on the:
 - [WifiP2pInfo](#)
 - This class represents Wi-Fi P2P group connection information. This class contains the field [groupFormed](#) which indicates whether a Wi-Fi P2P group has been successfully formed. This class also contains the field [groupOwnerAddress](#) which may be used to retrieve the host IP address, which is necessary for the client to register with the host.
 - [NetworkInfo](#)
 - This class represents the current network connection. This class is used to call the method [isConnected\(\)](#) to determine whether the current device has established a connection and is able to perform data transactions.
 - [WifiP2pGroup](#)
 - This class represents the current Wi-Fi P2P group. This group consists of the group owner and one or more clients. In particular, this class will be used call the method [getOwner\(\)](#) in to retrieve the host [WifiP2pDevice](#).
- [WIFI_P2P_THIS_DEVICE_CHANGED_ACTION](#)
 - This action indicates that the Wi-Fi P2P device has changed. This will be used to get a handle on the current [WifiP2pDevice](#).

These actions are received by the abstract class [DirectBroadcastReceiver](#) which is extended by an [anonymous class](#) within both the [Host](#) and [Client](#) constructors and registered with the application context. While this class is used by both the [Host](#) and [Client](#) classes, they have different [anonymous class](#) implementations. The reason why these classes are anonymously extended is that they require access to private members of both the [Host](#) and [Client](#) classes.

Apart from creating the [IntentFilter](#), the [Direct](#) class initializes the [Channel](#) which is the channel that connects the given application to the Wi-Fi P2P framework. This class will also use the respective [Application](#) to retrieve the application context in order to create a [Handler](#) in order for asynchronous methods to post a [Runnable](#) to the main thread.

Host

`class Host extends Direct`

This class is responsible for hosting services.

Host

`Host(Application application, String service, String instance)`

This constructor will create an instance of an anonymous class inheriting from [DirectBroadcastReceiver](#) and register said instance with the application context. This constructor will create an instance of `Map<String, String> record` to store within an instance of [WifiP2pDnsSdServiceInfo](#).

The [WifiP2pDnsSdServiceInfo](#) will eventually be passed to clients who are discovering the respective service. In particular, an entry will be put into the `Map<String, String> record` with the `SERVICE_NAME_TAG` key with the value as the respective `service` that will be hosted. This constructor will also create an instance of [HostRegistrar](#) to handle the registration of clients and [ObjectReceiver](#) to receive data from said clients.

startService

`void startService(ObjectCallback dataCallback, ResultCallback callback)`

This method will begin by clearing all local services, or in other words, stopping any previously existing service that the host may be providing.

Afterwards, the method will start the [ObjectReceiver](#) and [HostRegistrar](#), or in other words, initialize their respective [ServerSockets](#). The host now has a handle on the [HostRegistrar](#)'s [ServerSocket](#) port as it has been initialized. This port number will be put into the `Map<String, String> record` with the `REGISTRAR_PORT_TAG` as the key. Then the host instance of [WifiP2pDnsSdServiceInfo](#) will be updated to reflect the updated `record`.

Once the above has been successfully completed, this method will add the local service accompanied by the [WifiP2pDnsSdServiceInfo](#) for service discovery. This is important as the clients will use the [WifiP2pDnsSdServiceInfo](#) to look at the `Map<String, String> record` on discovery.

Client

```
class Client extends Direct
```

This class is responsible for discovering services, and connecting to said services.

Client

```
Client(Application application, String service, String instance)
```

This constructor will create an instance of an anonymous class inheriting from [DirectBroadcastReceiver](#) and register said instance with the application context. The constructor will finally set the [DnsSdServiceResponseListener](#) and [DnsSdTxtRecordListener](#) to be reused with each service request, more on this will be covered in the explanation of the [startDiscovery](#) method. This constructor will also create an instance of [ClientRegistrar](#) to handle the registration with the [HostRegistrar](#).

startDiscovery

```
void startDiscovery(final ResultCallback callback)
```

This method will create a new service request instance and send it to the Wi-Fi P2P framework. If successful, this method will then initiate service discovery. Service discovery is a process that involves scanning for requested services for the purpose of establishing a connection to a peer that supports an available service.

The service discovery notifies the library through the use of both a [DnsSdServiceResponseListener](#) and [DnsSdTxtRecordListener](#). For the purpose of this library, only the [DnsSdTxtRecordListener](#) is used. This is because the [DnsSdTxtRecordListener](#) retrieves the `Map<String, String> record` which contains two entries that are useful to the client. These two entries are [SERVICE_NAME_TAG](#) and [REGISTRAR_PORT_TAG](#). The [SERVICE_NAME_TAG](#) entry used to filter the discovered services, this entry will contain the unique identifier of the application. The [REGISTRAR_PORT_TAG](#) will contain the host registrar port, which in combination with the IP address of the host will be used to both register and unregister.

The services discovered which contain the proper [SERVICE_NAME_TAG](#) will be stored in `Map<WifiP2pDevice, Integer> nearbyHostDevices`, which is a map which the key is the respective host [WifiP2pDevice](#) and value is the respective registrar port stored in the [REGISTRAR_PORT_TAG](#).

stopDiscovery

```
void stopDiscovery(final ResultCallback callback)
```

This method will remove the service request created in `public void startDiscovery(final ResultCallback callback)` if said service request is not null.

`Map<WifiP2pDevice, Integer> nearbyHostDevices` will be cleared and all peer discovery will be ceased.

connect

```
void connect(WifiP2pDevice hostDevice, ObjectCallback dataCallback,
ResultCallback callback)
```

This method will attempt to connect to the given `hostDevice` based on a `WifiP2pConfig` consisting of the MAC address from said host device. This method will only attempt to establish this connection if the given host device is contained within `Map<WifiP2pDevice, Integer> nearbyHostDevices`, this is to prevent the client from connecting to host devices that are not running with the proper unique identifier.

This method will not actually establish the connection but rather sends a connection request to the Wi-Fi P2P framework. In the event of a successful connection between the client and host device, a [WIFI P2P CONNECTION CHANGED ACTION](#) intent will be broadcasted and notify the client of a change in connectivity.

When a successful connection is broadcasted, the client will first create an instance of [ObjectReceiver](#), which is essentially a [ServerSocket](#) listening for objects to be received from the host. At this point, the client has access to the host device IP address through the [WifiP2pGroup](#). Given the host IP address and registrar port, the client then connects to the [HostRegistrar](#) through the [ClientRegistrar](#) and sends a [Handshake](#) object consisting of the client MAC address and port the client [ObjectReceiver](#) is listening on. This is to notify the host of the established connection and to provide the host with means of sending objects to the client [ObjectReceiver](#).

The host will then send a [Handshake](#) object in return consisting of the MAC address and port of the host [ObjectReceiver](#) is listening on. This is to provide the client of the established connection and to provide the host with means of sending objects to the client [ObjectReceiver](#).

disconnect

```
void disconnect(ResultCallback callback)
```

This method will begin by unregistering the client from the host, this is done by sending an [Adieu](#) object to the host through the [ClientRegistrar](#) to the [HostRegistrar](#). This will notify the host that the client is disconnecting to prevent the host from continuing to send objects to the client [ObjectReceiver](#). This is required as the Wi-Fi P2P framework has no reliable functionality to detect client disconnects. The registrar is required to unregister before disconnecting, for when the client leaves the P2P group the [ClientRegistrar](#) will no longer be permitted to connect to the [HostRegistrar](#)'s [ServerSocket](#).

After unregistering, this method will then remove the current P2P group, and through reflection, this method will attempt to delete the persistent P2P group, as P2P groups are by default persisted in the Wi-Fi P2P framework. By reflection I mean that the method to within the Wi-Fi P2P framework to delete these persistent groups is not visible and must be accessed through reflection. This will effectively disconnect the client from the host.

As usual, this method will not actually remove the P2P group but rather sends a request to the Wi-Fi P2P framework. In the event of a successful disconnect between the client and host device, a [WIFI_P2P_CONNECTION_CHANGED_ACTION](#) intent will be broadcasted and notify the client of a change in connectivity. When a successful disconnect is broadcasted, all host information will be cleared and the client [ObjectReceiver](#) will be stopped.

send

```
void send(Serializable object, ResultCallback callback)
```

With the host IP address and the host [ObjectReceiver](#) port obtained from the registration handshake, this method will effectively send the host the respective **object** through [ObjectTransmitter](#). The **object** is required to implement [Serializable](#) as communication between the [ObjectTransmitter](#) and the [ObjectReceiver](#) make use of [ObjectInputStream](#) and [ObjectOutputStream](#).

CONCLUSION

I intend this library to be used by many other developers who wish to implement the Wi-Fi Direct functionality and potentially become a community to further progress the strength and reliability of the library through collaborative efforts.