**Chapter 3: Implementation**

**Section 1: LTE Detection Module Implementation**

To demonstrate the implementation details of the channel list detection, we take the LTE detections Module as an example. The implementation of Wi-Fi detection module is almost the same, the difference will be highlighted at the end of this section.

**Section 1.1: LTE Data Model**

For each LTE channel, we design a LTE class to represent them.

/\*LTE.java\*/

public class LTE {

private int mPci;

private int mDbm;

private int mAsuLevel;

}

It each scanning, we obtain a list of the LTE channels, we define a LTE List class to hold the information. As me mentioned in chapter 2, we implement the singleton pattern on the LTE List class, the implementation details are as follow:

/\*LTEList.java\*/

private static LTEList sLTEList;

private ArrayList<LTE> mLTEList = new ArrayList<>();

private LTEList(Context context){

getLTEList(context);

}

public static LTEList get(Context context){

if (sLTEList == null){

sLTEList = new LTEList(context);

}

sLTEList.getLTEList(context);

return sLTEList;

}

}

To implement singleton pattern, the constructor of the LTEList class is defined as private, the outer functions are only able to get the instance of the class by the LTEList.get(Context) function and the function always return the same sLTEList object. It ensures that every caller of the class are given the same LTEList.

**Section 1.2: XML File**

A xml file is the file used in android handling UI.

To use fragment, we first need to define a container in the xml file of the activity. A container is used to hold the fragment defined later. A container can hold any fragment, so LTE List Fragment and Wi-Fi List Fragment both can be hold by this container.

/\* activity\_fragment.xml \*/

<FrameLayout

xmlns:android="http://schemas.android.com/apk/res/android"

**android:id="@+id/fragment\_container"**/>

We use a RecyclerView to display the LTE List we have obtain. RecyclerView is a sub-class of ViewGroup. It displays a list of child View objects.

/\*fragment\_list.xml\*/

<**android. support. v7. widget.RecyclerView**

xmlns:android="http://schemas.android.com/apk/res/android"

android:id="@+id/recycler\_view" />

Also we need to define how each of the elements inside the RecyclerView is going to be displayed.

/\*list\_item\_lte.xml\*/

<TextView

**android:id="@+id/list\_pci\_text\_view"**/>

<TextView

**android:id="@+id/list\_dbm\_text\_view"**

android:layout\_below="@+id/list\_pci\_text\_view"/>

<TextView

**android:id="@+id/list\_asulevel\_text\_view"**

android:layout\_below="@+id/list\_dbm\_text\_view" />

Then each LTE channel is displayed as follow:

PCI:

DBM:

ASU:

Figure (2) LTE List Item View

**Section 1.3: Fragment Class**

Then, we create the LTE List Fragment class which is a sub-class of the standard Fragment class from Android:

/\*LTEListFragment.java\*/

public class LTEListFragment extends Fragment {

**private RecyclerView mRecyclerView;**

public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {

View view = inflater.inflate(R.layout.fragment\_list, container, false);

**mRecyclerView = (RecyclerView) view.findViewById(R.id.recycler\_view);**

return view;

}

}

Figure(3) RecyclerView Mechanism

As shown of figure (3), when we use the RecyclerView, a LayoutManager and an Adapter are required.

The LayoutManager is responsible for placing the item views on right position inside the RecyclerView. Also it determines when to reuse the item views which are created previously and no long visible to user. In our application, we use the LinearLayoutManager provided by RecyclerView API.

/\*LTEListFragment.java\*/

public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {

View view = inflater.inflate(R.layout.fragment\_list, container, false);

mRecyclerView = (RecyclerView) view.findViewById(R.id.recycler\_view);

**mRecyclerView.setLayoutManager(new LinearLayoutManager(getActivity()));**

return view;

}

As for Adapter, it plays two roles: provide access to the data set and create the correct layout of each individual items the RecyclerView is going to display. We need to override two methods, onCreateViewHolder, which is used to inflate the view and its view holder, and also onBindViewHolder, which is used to bind the data to the view.

onCreateViewHolder is used to inflate the item layout and create the holder. To override onCreateViewHolder, we need to define the holder we are going to use first. The holder is defined once the RecyclerView is given the data set, how RecyclerView should extract the information from the data set to display.

/\*LTEListFragment.java\*/

private class **Holder** extends RecyclerView.ViewHolder

{

private LTE mLTE;

private TextView mPci;

private TextView mDbm;

private TextView mAsuLevel;

public **Holder** (View itemView) {

super(itemView);

itemView.setOnClickListener(this);

mPci = (TextView) itemView.findViewById(R.id.list\_pci\_text\_view);

mDbm = (TextView) itemView.findViewById(R.id.list\_dbm\_text\_view);

mAsuLevel = (TextView) itemView.findViewById(R.id.list\_asulevel\_text\_view);

}

public void bindLTE(LTE lte) {

**mLTE = lte;**

**mPci.setText**(...**String.valueOf(lte.getmPci())**);

**mDbm.setText**(...**String.valueOf(lte.getmDbm())**);

**mAsuLevel.setText(**…**String.valueOf(lte.getmAsuLevel())));**

}

}

/\*LTEListFragment.java\*/

public **Holder** onCreateViewHolder(ViewGroup parent, int viewType) {

LayoutInflater layoutInflater = LayoutInflater.from(getActivity());

View view = layoutInflater.inflate(R.layout.list\_item\_lte, parent, false);

return new Holder(view);

}

When the layout manager needs to display an item view in the RecyclerView’s visible screen area, it has to call the onBindViewHolder. OnBindViewHolder is responsible for binding the data with the item view we are going to display.

/\*LTEListFragment.java\*/

public void onBindViewHolder(Holder holder,int position) {

**LTE lte = mLTEs.get(position)**;

**holder.bindLTE(lte);**

}

**Section 1.4: Activity**

To use the fragment, we finished above, we need a FragmentManager to help us manage the transaction between fragments, including adding a new fragment, replacing or removing the existing one.

As all of the activities in our application use fragments to display, we build an abstract class to reuse the codes.

/\*SingleFragmentActivity.java\*/

public abstract class SingleFragmentActivity extends FragmentActivity {

protected abstract Fragment createFragment();

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_fragment);

**FragmentManager fm = getSupportFragmentManager();**

**Fragment fragment = fm.findFragmentById(R.id.fragment\_container);**

**if (fragment == null) {**

**fragment = createFragment();**

**fm.beginTransaction()**

**.add(R.id.fragment\_container, fragment)**

**. commit();**

**}**

}

}

To use this abstract class, the activity just need to override the createFragment() function

public class LTEListActivity extends SingleFragmentActivity {

**@Override**

**protected Fragment createFragment() {**

**return new LTEListFragment();**

**}**

}

**Section 2: Wi-Fi Detection Module Implementation**

The implementation of the Wi-Fi detection module is almost the same as LTE detection module. The main difference is the data model we use to represent the Wi-Fi AP.

/\* WiFi.java\*/

public class WiFi {

private String mSSID;

private String mBSSID;

private int mSignalLevel;

private long mTimestamp;

}

**Section 3: Localization Implementation**

When users click on one of the items shown on the LTE List Fragment, we need to get the id of the item. In android, intent is used to pass information between activities. In our implementation, we define a method how outer activity pass information to Localization Activity. In this case, LTE’s pci is passed.

/\*LocationActivity.java\*/

public static Intent newIntent(Context packageContext,int pci){

**Intent i = new Intent(packageContext,LocationActivity.class);**

**i.putExtra(GET\_PCI,pci);**

return i;

}

When Location Activity receives the id, it also needs to pass it to the Location Fragment. In this case, we first define the pci as an argument to call the LocationFragment. And in the Location Activity, we pass the pci received from intent as the argument to call LocationFragmetn.

/\*LocationFragment.java\*/

public static LocationFragment newInstance(int pci) {

Bundle args = new Bundle();

**args.putInt(ARG\_PCI,pci);**

LocationFragment fragment = new LocationFragment();

**fragment.setArguments(args);**

return fragment;

}

/\*LocationActivity.java\*/

@Override

protected Fragment createFragment() {

**int pci = getIntent().getIntExtra(GET\_PCI,0);**

**return LocationFragment.newInstance(pci);**

}

To use localization and mapping services, first we need a GoogleApiClient. Inside the GoogleApiClient, we need to define what we want to do when the GoogleApiClient is connected to the Google Play Service. In our application, we need to update our current location and when the location has changed, we need obtain the signal level in the new location to update the UI. We define a findLocation() method.

/\*LocationFragment.java\*/

private Location mCurrentLocation;

private GoogleApiClient mClient;

private GoogleMap mMap;

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setLtePci(getArguments().getInt(ARG\_PCI));

**mClient = new GoogleApiClient.Builder(getActivity())**

.addApi(LocationServices.API)

.addConnectionCallbacks(new GoogleApiClient.ConnectionCallbacks() {

@Override

**public void onConnected(Bundle bundle) {**

**findLocation();**

**}**

@Override

public void onConnectionSuspended(int i) {

}

}). build ();

getMapAsync(new OnMapReadyCallback() {

@Override

public void onMapReady(GoogleMap googleMap) {

mMap = googleMap;

}

});

}

private void findLocation() {

**LocationRequest request = LocationRequest.create();**

request.setPriority(LocationRequest.PRIORITY\_HIGH\_ACCURACY);

request.setInterval(10);

request.setSmallestDisplacement(2);

LocationServices.FusedLocationApi

.requestLocationUpdates(mClient, request, new LocationListener() {

@Override

**public void onLocationChanged(Location location) {**

**mCurrentLocation = location;**

**updateUI();**

**}**

});

}

}