UNIT – VI

Introduction to Android

Networking API

1. Overview of HTTP access on Android

1.1. Available official API's

Android contains the standard Java network java.net package which can be used to access network resources. The base class for HTTP network access in the java.net package is the HttpURLConnection class.

1.2. Required permissions

To access the Internet your application requires the android.permission.INTERNET permission.

To check the network state your application requires the android.permission.ACCESS_NETWORK_STATE permission.

1.3. Open source libraries

Performing network operations on Android can be cumbersome. You have to open and close a connections, enable caches and ensure to perform the network operation in a background thread.

To simplify these operations several popular Open Source libraries are available. The most popular ones are the following:

- Volley
- OkHttp

2. Good practices for network access under Android

Within an Android application you should avoid performing long running operations on the user interface thread. This includes file and network access.

As of Android 3.0 (Honeycomb) the system is configured to crash with a NetworkOnMainThreadException exception, if network is accessed in the user interface thread.

A typical setup for performing network access in a productive Android application is using a service. While it is possible to do network access from an activity or a fragment, using a service typical leads to a better overall design because you code in the activity becomes simpler.

3. Java and HTTP access

Java provides a general-purpose, lightweight HTTP client API to access resources via the HTTP or HTTPS protocol. The main classes to access the Internet are the java.net.URL class and the java.net.HttpURLConnection class.

The URL class can be used to define a pointer to a web resource while the HttpURLConnection class can be used to access a web resource.

HttpURLConnection allows you to create an InputStream.

Once you have accessed an InputStream you can read it similarly to an InputStream from a local file.

In the latest version HttpURLConnection supports the transparent response compression (via the header Accept-Encoding: gzip, Server Name Indication (extension of SSL and TLS) and a response cache.

The API is relatively straight forward. For example to retrieve the webpage www.vogella.com you can use the following example.

```
try {
  URL url = newURL("http://www.vogella.com");
HttpURLConnection con = (HttpURLConnection) url
    .openConnection();
readStream(con.getInputStream());
  } catch (Exception e) {
e.printStackTrace();
}
```

```
privatevoidreadStream(InputStream in) {
BufferedReader reader = null;
try {
reader = newBufferedReader(newInputStreamReader(in));
    String line = "";
while ((line = reader.readLine()) != null) {
System.out.println(line);
    }
  } catch (IOException e) {
e.printStackTrace();
  } finally {
if (reader != null) {
try {
reader.close();
```

```
      } catch (IOException e) {
e.printStackTrace();
        }
    }
  }
}
```

4. Check the network availability

Obviously the network on an Android device is not always available. You can check the network is currently available via the following code.

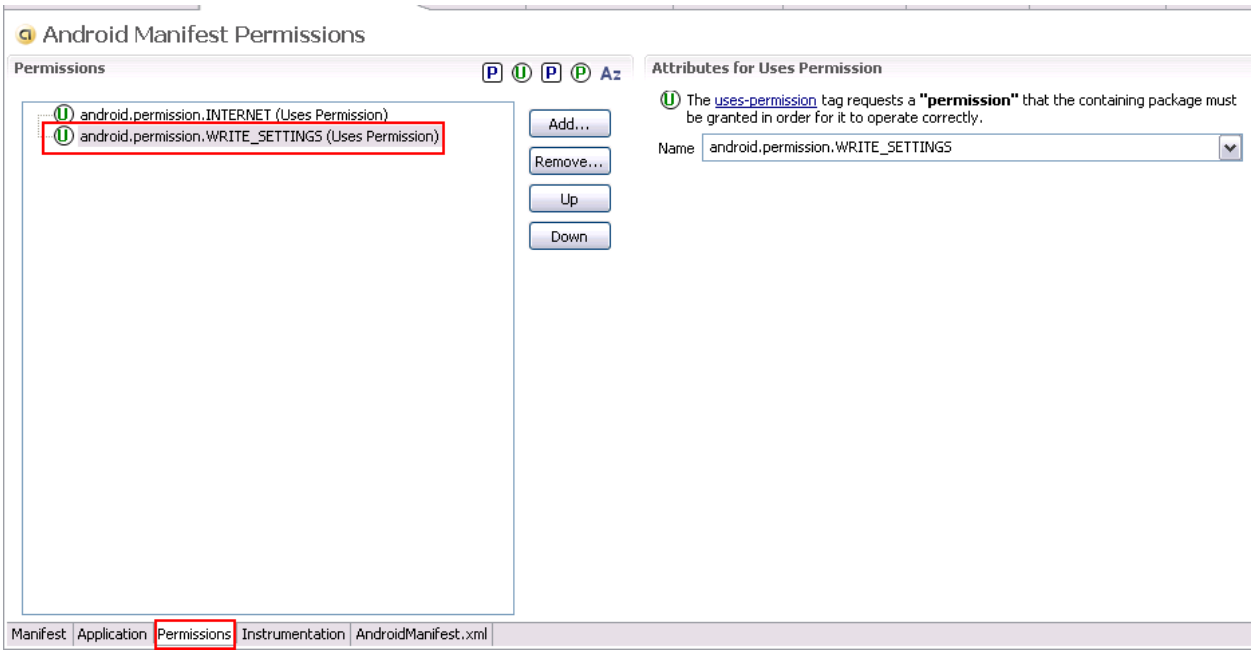
```
publicbooleanisNetworkAvailable() {
ConnectivityManager cm = (ConnectivityManager)
getSystemService(Context.CONNECTIVITY_SERVICE);
NetworkInfonetworkInfo = cm.getActiveNetworkInfo();
// if no network is available networkInfo will be null
// otherwise check if we are connected
if (networkInfo != null &&networkInfo.isConnected()) {
return true;
    }
return false;
}
```

5. Proxy

This chapter is only relevant for you if you are testing with the Android simulator behind a proxy. You can set the proxy via the Settings class. For example you could add the following line to your onCreate method in your activity.

```
Settings.System.putString(getContentResolver(),
Settings.System.HTTP_PROXY, "myproxy:8080");
```

To change the proxy settings you have to have the android.permission.WRITE_SETTINGS permission in your AndroidManifest.xml file.



Parsing XML Data

Extensible Markup Language (XML) is a set of rules for encoding documents in machine-readable form. XML is a popular format for sharing data on the internet. Websites that frequently update their content, such as news sites or blogs, often provide an XML feed so that external programs can keep abreast of content changes. Uploading and parsing XML data is a common task for network-connected apps. This lesson explains how to parse XML documents and use their data.

Choose a Parser

We recommend XmlPullParser, which is an efficient and maintainable way to parse XML on Android. Historically Android has had two implementations of this interface:

- KXmlParser via XmlPullParserFactory.newPullParser().
- ExpatPullParser, via Xml.newPullParser().

Either choice is fine. The example in this section uses ExpatPullParser, via Xml.newPullParser()

Analyze the Feed

The first step in parsing a feed is to decide which fields you're interested in. The parser extracts data for those fields and ignores the rest.

Here is an excerpt from the feed that's being parsed in the sample app. Each post to StackOverflow.com appears in the feed as an entry tag that contains several nested tags:

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom"
xmlns:creativeCommons="http://backend.userland.com/creativeCommo
nsRssModule" ...">
<title type="text">newest questions tagged android - Stack
Overflow</title>
...
    <entry>
    ...
    </entry>
    <entry>
        <id>http://stackoverflow.com/q/9439999</id>
        <re:rank scheme="http://stackoverflow.com">0</re:rank>
        <title type="text">Where is my data file?</title>
        <category
scheme="http://stackoverflow.com/feeds/tag?tagnames=android&sort
=newest/tags" term="android"/>
        <category
scheme="http://stackoverflow.com/feeds/tag?tagnames=android&sort
=newest/tags" term="file"/>
        <author>
            <name>cliff2310</name>
            <uri>http://stackoverflow.com/users/1128925</uri>
        </author>
        <link rel="alternate"
href="http://stackoverflow.com/questions/9439999/where-is-my-
data-file" />
        <published>2012-02-25T00:30:54Z</published>
        <updated>2012-02-25T00:30:54Z</updated>
        <summary type="html">
            <p>I have an Application that requires a data
file...</p>

        </summary>
    </entry>
    <entry>
    ...
    </entry>
...
</feed>
```

The sample app extracts data for the entry tag and its nested tags title, link, and summary.

Instantiate the Parser

The next step is to instantiate a parser and kick off the parsing process. In this snippet, a parser is initialized to not process namespaces, and to use the provided InputStream as its input. It starts the parsing process with a call to nextTag() and invokes the readFeed() method, which extracts and processes the data the app is interested in:

```
publicclassStackOverflowXmlParser{
    // We don't use namespaces
    privatestaticfinalString ns =null;

    publicList
parse(InputStreamin)throwsXmlPullParserException,IOException{
        try{
            XmlPullParser parser =Xml.newPullParser();

parser.setFeature(XmlPullParser.FEATURE_PROCESS_NAMESPACES,false
);
            parser.setInput(in,null);
            parser.nextTag();
            returnreadFeed(parser);
        }finally{
            in.close();
        }
    }
 ...
}
```

Read the Feed

The readFeed() method does the actual work of processing the feed. It looks for elements tagged "entry" as a starting point for recursively processing the feed. If a tag isn't an entry tag, it skips it. Once the whole feed has been recursively processed, readFeed() returns a List containing the entries (including nested data members) it extracted from the feed. This List is then returned by the parser.

```
privateListreadFeed(XmlPullParser
parser)throwsXmlPullParserException,IOException{
    List entries =newArrayList();

    parser.require(XmlPullParser.START_TAG, ns,"feed");
    while(parser.next()!=XmlPullParser.END_TAG){
        if(parser.getEventType()!=XmlPullParser.START_TAG){
```

```
            continue;
        }
        String name =parser.getName();
        // Starts by looking for the entry tag
        if(name.equals("entry")){
            entries.add(readEntry(parser));
        }else{
            skip(parser);
        }
    }
    return entries;
}
```

Telephony API

The Android Telephony package

Programmers will need ways to not just retrieve telephony data, but also to dial a phone number, intercept outgoing phone calls, or send/receive SMS messages. This is achieved with a combination of classes provided in the android.telephony package, along with inbuilt Intents and phone services.

The most important of these is the TelephonyManager class, which can be used to retrieve metadata about the SIM card, the physical device and the network provider. Programmers will need to use the Android Context, through the getSystemService() method, with a constant as shown below, to obtain an instance of the TelephonyManager class.

```
// ..creating TelephonyManager from Context
finalTelephonyManagermytelMgr =
(TelephonyManager)this.getSystemService(Context.TELEPHONY_SERVICE);
```

With this handle, telephony data can be retrieved. Sample code to retrieve the phone's call state, using the getCallState() method, is shown below.

```
publicString getCurrentCallState(finalTelephonyManagermytelMgr)
{
        intcallState = mytelMgr.getCallState();
        String callStateString = "NOTKNOWN";
        switch(callState) {
            caseTelephonyManager.CALL_STATE_IDLE:
                callStateString = "IDLE";
                break;
            caseTelephonyManager.CALL_STATE_OFFHOOK:
                callStateString = "OFFHOOK";
                break;
            caseTelephonyManager.CALL_STATE_RINGING:
```

```
                callStateString = "RINGING";
                break;
        }
}
```

Additionally, it is often important to know the change in the call or service state of the phone, in an app. For example, you may want to mute your application when a call arrives. This is done by attaching a listener, called PhoneStateListener to the TelephonyManager.

Finally, it is important to note that information retrieval via TelephonyManager is permission-protected. For the application to access this information, the android.permission.READ_PHONE_STATE permission has to be set in the app's manifest.xml. Figure 2 shows some metrics retrieved using TelephonyManager.

The **android.telephony.TelephonyManager** class provides information about the telephony services such as subscriber id, sim serial number, phone network type etc. Moreover, you can determine the phone state etc.

activity_main.xml

```
<RelativeLayout xmlns:androclass="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   tools:context=".MainActivity" >

   <TextView
      android:id="@+id/textView1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentLeft="true"
      android:layout_alignParentTop="true"
      android:layout_marginLeft="38dp"
      android:layout_marginTop="30dp"
      android:text="Phone Details:" />

</RelativeLayout>
```

MainActivity.java

```
package com.javatpoint.telephonymanager;

import android.os.Bundle;
import android.app.Activity;
```

```
import android.content.Context;
import android.telephony.TelephonyManager;
import android.view.Menu;
import android.widget.TextView;

public class MainActivity extends Activity {
  TextView textView1;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        textView1=(TextView)findViewById(R.id.textView1);

        //Get the instance of TelephonyManager
        TelephonyManager  tm=(TelephonyManager)getSystemService(Context.TELEPHONY_SE
RVICE);

        //Calling the methods of TelephonyManager the returns the information
        String IMEINumber=tm.getDeviceId();
        String subscriberID=tm.getDeviceId();
        String SIMSerialNumber=tm.getSimSerialNumber();
        String networkCountryISO=tm.getNetworkCountryIso();
        String SIMCountryISO=tm.getSimCountryIso();
        String softwareVersion=tm.getDeviceSoftwareVersion();
        String voiceMailNumber=tm.getVoiceMailNumber();

        //Get the phone type
        String strphoneType="";

   int phoneType=tm.getPhoneType();

        switch (phoneType)
        {
              case (TelephonyManager.PHONE_TYPE_CDMA):
                    strphoneType="CDMA";
                       break;
              case (TelephonyManager.PHONE_TYPE_GSM):
                    strphoneType="GSM";
                       break;
              case (TelephonyManager.PHONE_TYPE_NONE):
                     strphoneType="NONE";
                        break;
         }

        //getting information if phone is in roaming
```

```
        boolean isRoaming=tm.isNetworkRoaming();

        String info="Phone Details:\n";
        info+="\n IMEI Number:"+IMEINumber;
        info+="\n SubscriberID:"+subscriberID;
        info+="\n Sim Serial Number:"+SIMSerialNumber;
        info+="\n Network Country ISO:"+networkCountryISO;
        info+="\n SIM Country ISO:"+SIMCountryISO;
        info+="\n Software Version:"+softwareVersion;
        info+="\n Voice Mail Number:"+voiceMailNumber;
        info+="\n Phone Network Type:"+strphoneType;
        info+="\n In Roaming? :"+isRoaming;

        textView1.setText(info);//displaying the information in the textView
    }


}
```

AndroidManifest.xml

```
<uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

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

Web View

If you want to deliver a web application (or just a web page) as a part of a client application, you can do it using WebView. The WebView class is an extension of Android's View class that allows you to display web pages as a part of your activity layout. It does *not* include any features of a fully developed web browser, such as navigation controls or an address bar. All that WebView does, by default, is show a web page.

A common scenario in which using WebView is helpful is when you want to provide information in your application that you might need to update, such as an end-user agreement or a user guide. Within your Android application, you can create an Activity that contains a WebView, then use that to display your document that's hosted online. Another scenario in which WebView can help is if your application provides data to the user that always requires an Internet connection to retrieve data, such as email. In this case, you might find that it's easier to build a Web your Android application that shows a web page with all the user data, rather than performing a network request, then parsing the data and rendering it in an Android layout. Instead, you can design a web page that's tailored for Android devices and then implement a WebView in your Android application that loads the web page.

This document shows you how to get started with WebView and how to do some additional things, such as handle page navigation and bind JavaScript from your web page to client-side code in your Android application.

Adding a WebView to your application

To add a WebView to your Application, simply include the <WebView> element in your activity layout. For example, here's a layout file in which the WebView fills the screen:

```
<?xml version="1.0" encoding="utf-8"?>
<WebView
 xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/webview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
/>
```

To load a web page in the WebView, use loadUrl(). For example:

```
WebViewmyWebView=(WebView)findViewById(R.id.webview);
myWebView.loadUrl("http://www.example.com");
```

To get Internet access, request the INTERNET permission in your manifest file. For example:

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

```
    ...
</manifest>
```