NETWORKING

COMP 41690

DAVID COYLE > D.COYLE@UCD.IE

CHECK THE NETWORK CONNECTION

Remember, the device may be out of range of a network, or the user may have disabled both Wi-Fi and mobile data access.

Before your app attempts to connect to the network, it should check to see whether a network connection is available using getActiveNetworkInfo() and isConnected().

```
public void myClickHandler(View view) {
    ...
    ConnectivityManager connMgr = (ConnectivityManager)
        getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
    if (networkInfo != null && networkInfo.isConnected()) {
        // fetch data
    } else {
        // display error
    }
    ...
}
```

https://developer.android.com/training/basics/network-ops/managing.html

CHECK THE NETWORK CONNECTION

Remember, the device may be out of range of a network, or the user may have disabled both Wi-Fi and mobile data access.

Before your app attempts to connect to the network, it should check to see whether a network connection is available using getActiveNetworkInfo() and isConnected().

```
public void myClickHandler(View view) {
    ...
    ConnectivityManager connMgr = (ConnectivityManager)
        getSystemService
    NetworkInfo networkI
    if (networkInfo != n
        // fetch data
    } else {
        // display error
    }
    ...
}
```

```
public boolean isOnline() {
    ConnectivityManager connMgr = (ConnectivityManager)
            getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
    return (networkInfo != null && networkInfo.isConnected());
}
```

https://developer.android.com/training/basics/network-ops/managing.html

CONNECTING TO A NETWORK

Android includes multiple networking support classes.

- java.net – (Socket, URL)
- org.apache – (HttpRequest, HttpResponse)
- android.net – (URI, AndroidHttpClient, AudioStream)

Connections are typically made using HTTP requests, via:

- Socket
- HttpURLConnection
- AndroidHttpClient

SOCKET

Part of the java.net package

See **NetworkingSockets** example

```
@Override
protected String doInBackground(Void... params) {
    Socket socket = null;
    String data = "";

    try {
        socket = new Socket(HOST, 80);
        PrintWriter pw = new PrintWriter(new OutputStreamWriter(
                socket.getOutputStream()), true);
        pw.println(HTTP_GET_COMMAND);

        data = readStream(socket.getInputStream());

    }
```

HTTP CLIENTS

Most network-connected Android apps will use HTTP to send and receive data.

Android includes two HTTP clients:

- HttpURLConnection
- AndroidHttpClient

Both support HTTPS, streaming uploads and downloads, configurable timeouts, IPv6 and connection pooling.

Which is best?

AndroidHttpClient has fewer bugs on Eclair and Froyo. It is the best choice for these releases.

For Gingerbread forward, HttpURLConnection is the best choice.

Its simple API and small size makes it great fit for Android. Transparent compression and response caching reduce network use, improve speed and save battery.

http://android-developers.blogspot.com.es/2011/09/androids-http-clients.html

HTTP CLIENTS

Most network-connected Android apps will use HTTP to send and receive data.

Android includes two HTTP clients:

- HttpURLConnection
- AndroidHttpClient

Both support HTTPS, streaming uploads and downloads, configurable timeouts, IPv6 and connection pooling.

Android developers' words:

"DefaultHttpClient and its sibling AndroidHttpClient are extensible HTTP clients suitable for web browsers. They have large and flexible APIs. Their implementation is stable and they have few bugs.

But the large size of this API makes it difficult for us to improve it without breaking compatibility. The Android team is not actively working on Apache HTTP Client."

http://android-developers.blogspot.com.es/2011/09/androids-http-clients.html

HTTPURLCONNECTION

An URLConnection for HTTP used to send and receive data over the web.

- Data may be of any type and length.
- This class may be used to send and receive streaming data whose length is not known in advance.
- Always preform network operations on a separate thread to the UI thread.

See **NetworkingURL** example

```
HttpURLConnection httpUrlConnection = null;

try {
    httpUrlConnection = (HttpURLConnection) new URL(URL)
            .openConnection();

    InputStream in = new BufferedInputStream(
            httpUrlConnection.getInputStream());

    data = readStream(in);

}
```

USING HTTPURLCONNECTION

1. Obtain a new HttpURLConnection by calling URL.openConnection() and casting the result to HttpURLConnection.
2. Prepare the request. The primary property of a request is its URI. Request headers may also include metadata such as credentials, preferred content types, and session cookies.
3. Optionally upload a request body. Instances must be configured with setDoOutput(true) if they include a request body. Transmit data by writing to the stream returned by getOutputStream().
4. Read the response. Response headers typically include metadata such as the response body's content type and length, modified dates and session cookies. The response body may be read from the stream returned by getInputStream(). If the response has no body, that method returns an empty stream.
5. Disconnect. Once the response body has been read, the HttpURLConnection should be closed by calling disconnect(). Disconnecting releases the resources held by a connection so they may be closed or reused.

UPLOADING CONTENT

To upload data to a web server, configure the connection for output using setDoOutput(true).

For best performance, you should either:

- Call setFixedLengthStreamingMode(int) when the body length is known in advance,

or

- Call setChunkedStreamingMode(int) when it is not.

Otherwise HttpURLConnection will be forced to buffer the complete request body in memory before it is transmitted, wasting (and possibly exhausting) heap and increasing latency.

UPLOADING CONTENT

```
 HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
try {
  urlConnection.setDoOutput(true);
  urlConnection.setChunkedStreamingMode(0);

  OutputStream out = new BufferedOutputStream(urlConnection.getOutputStream());
  writeStream(out);

  InputStream in = new BufferedInputStream(urlConnection.getInputStream());
  readStream(in);
 finally {
  urlConnection.disconnect();
 }
}
```

SECURE COMMUNICATION

Calling openConnection() on a URL with the "https" scheme will return an HttpsURLConnection.

See HttpsURLConnection for more details.

https://developer.android.com/reference/javax/net/ssl/HttpsURLConnection.html

WEATHER APP UPDATES

ForeCastFragment.java | AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.davidcoyle.sunshine.app" >

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Sunshine"
```

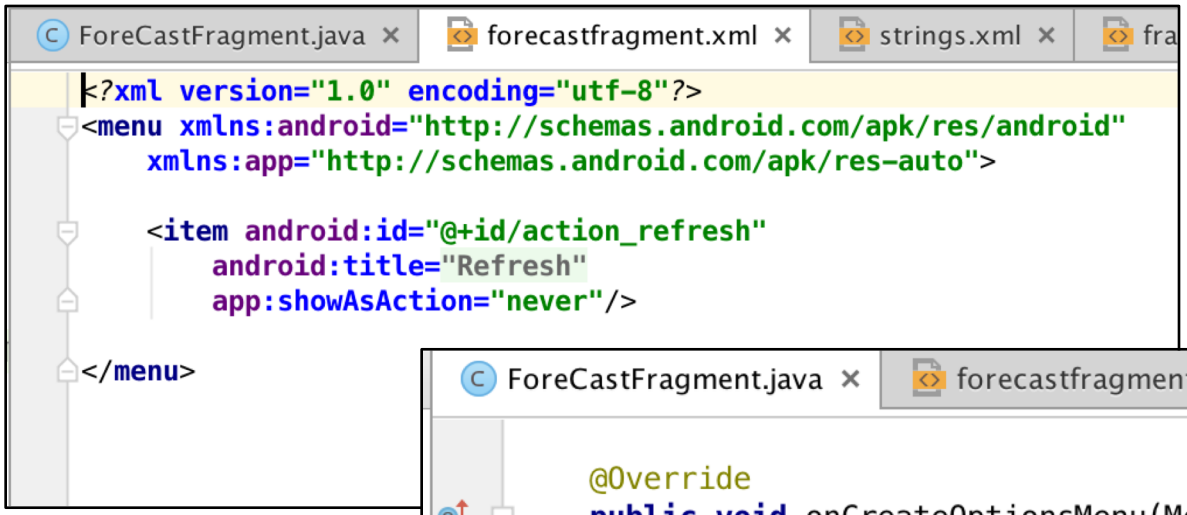
ForeCastFragment.java | forecastfragment.xml | strings.xml | fra

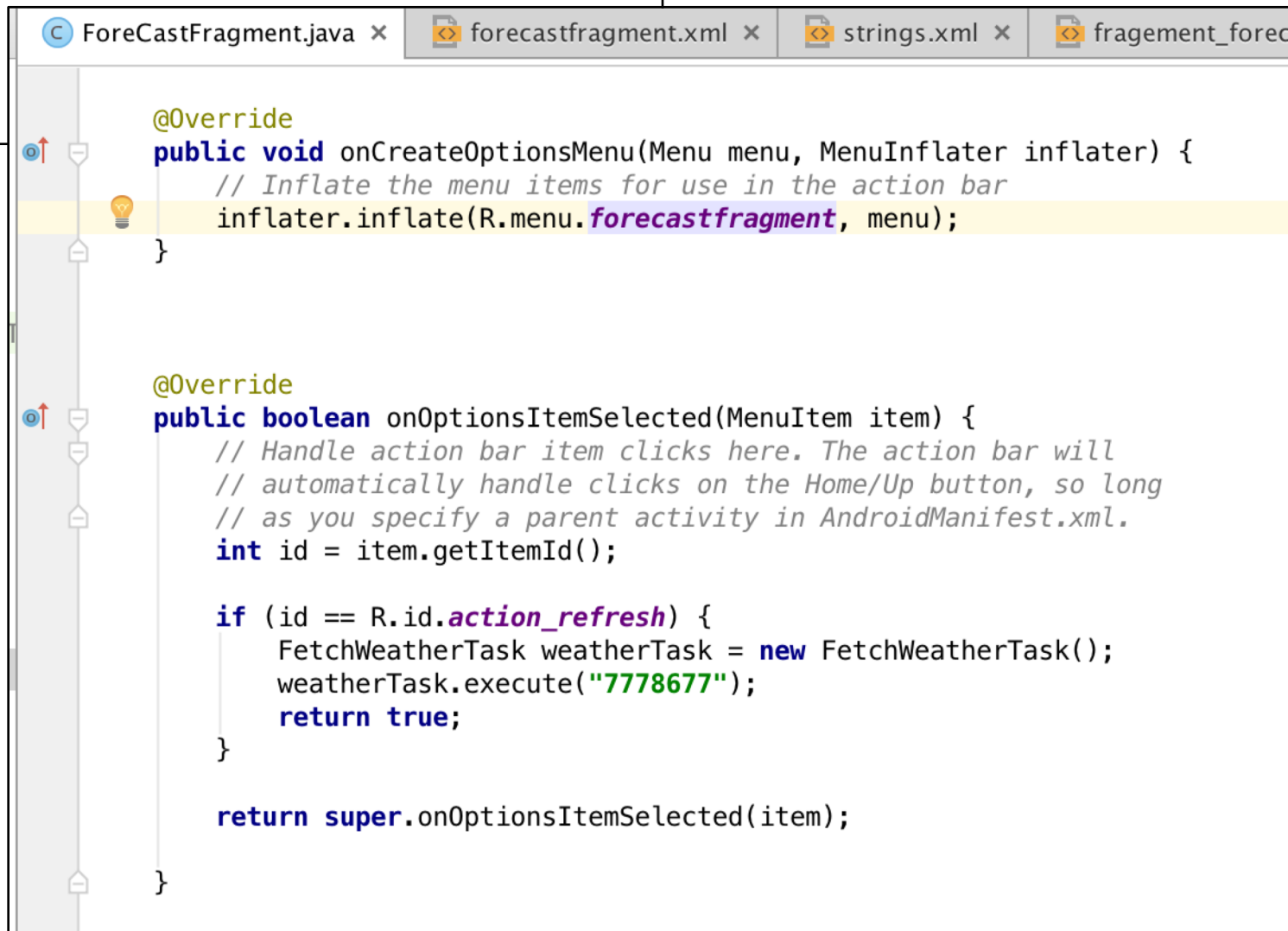
```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <item android:id="@+id/action_refresh"
        android:title="Refresh"
        app:showAsAction="never"/>

</menu>
```

https://developer.android.com/guide/topics/ui/actionbar.html


ForeCastFragment.java
forecastfragment.xml
strings.xml
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto">
<item android:id="@+id/action_refresh"
android:title="Refresh"
app:showAsAction="never"/>
</menu>


ForeCastFragment.java
forecastfragment.xml
strings.xml
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
// Inflate the menu items for use in the action bar
inflater.inflate(R.menu.forecastfragment, menu);
}
@Override
public boolean onOptionsItemSelected(MenuItem item) {
// Handle action bar item clicks here. The action bar will
// automatically handle clicks on the Home/Up button, so long
// as you specify a parent activity in AndroidManifest.xml.
int id = item.getItemId();
if (id == R.id.action_refresh) {
FetchWeatherTask weatherTask = new FetchWeatherTask();
weatherTask.execute("7778677");
return true;
}
return super.onOptionsItemSelected(item);
}

The sequence of events is as follows:

1. When users click refresh in the options menu

2. The app passes the id string to the AsyncTask subclass FetchWeatherTask.

ForeCastFragment.java

```
public class FetchWeatherTask extends AsyncTask<String, Void, String[]> {

    private final String LOG_TAG = FetchWeatherTask.class.getSimpleName();


    /* The date/time conversion code is going to be moved outside the asynctask later,
    * so for convenience we're breaking it out into its own method now.
    */
    private String getReadableDateString(long time){...}

    /**...*/
    private String formatHighLows(double high, double low) {...}


    /**...*/
    private String[] getWeatherDataFromJson(String forecastJsonStr, int numDays)
            throws JSONException {...}


    @Override
    protected String[] doInBackground(String... params) {...}


    @Override
    protected void onPostExecute(String[] results) {...}
```

The sequence of events is as follows:

1. When users click refresh in the options menu

2. The app passes the id string to the AsyncTask subclass FetchWeatherTask.

3. The AsyncTask method doInBackground() makes a URL string as a parameter and uses it to create a URL object.

4. The URL object is used to establish an HttpURLConnection.

ForeCastFragment.java ×

```
        }

        // These two need to be declared outside the try/catch
        // so that they can be closed in the finally block.
        HttpURLConnection urlConnection = null;
        BufferedReader reader = null;

        // Will contain the raw JSON response as a string.
        String forecastJsonStr = null;

        String format = "json";
        String units = "metric";
        int numDays = 10;

        try {
            // Construct the URL for the OpenWeatherMap query
            // Possible parameters are avaiable at OWM's forecast API page, at
            // http://openweathermap.org/API#forecast

            final String FORECAST_BASE_URL =
                    "http://api.openweathermap.org/data/2.5/forecast/daily?";
            final String QUERY_PARAM = "id";
            final String FORMAT_PARAM = "mode";
            final String UNITS_PARAM = "units";
            final String DAYS_PARAM = "cnt";

            Uri builtUri = Uri.parse(FORECAST_BASE_URL).buildUpon()
                    .appendQueryParameter(QUERY_PARAM, params[0])
                    .appendQueryParameter(FORMAT_PARAM, format)
                    .appendQueryParameter(UNITS_PARAM, units)
                    .appendQueryParameter(DAYS_PARAM, Integer.toString(numDays))
                    .build();

            URL url = new URL(builtUri.toString());

            Log.v(LOG_TAG, "Built URI " + builtUri.toString());


            //URL url = new URL("http://http://api.openweathermap.org/data/2.5/forecast/daily?id=52490

            // Create the request to OpenWeatherMap, and open the connection
            urlConnection = (HttpURLConnection) url.openConnection();
            urlConnection.setRequestMethod("GET");
            urlConnection.connect();
```

The sequence of events is as follows:

1. When users click refresh in the options menu

2. The app passes the id string to the AsyncTask subclass FetchWeatherTask.

3. The AsyncTask method doInBackground() makes a URL string as a parameter and uses it to create a URL object.

4. The URL object is used to establish an HttpURLConnection.

5. Once the connection has been established, the HttpURLConnection object fetches the web page content as an InputStream.

6. The InputStream is read/buffered and converted to a string.

ForeCastFragment.java ×

```
            // Read the input stream into a String
            InputStream inputStream = urlConnection.getInputStream();
            StringBuffer buffer = new StringBuffer();
            if (inputStream == null) {
                // Nothing to do.
                return null;
            }
            reader = new BufferedReader(new InputStreamReader(inputStream));

            String line;
            while ((line = reader.readLine()) != null) {
                // Since it's JSON, adding a newline isn't necessary (it won't affect parsing)
                // But it does make debugging a *lot* easier if you print out the completed
                // buffer for debugging.
                buffer.append(line + "\n");
            }

            if (buffer.length() == 0) {
                // Stream was empty.  No point in parsing.
                return null;
            }
            forecastJsonStr = buffer.toString();

            //Log.v(LOG_TAG, "Forecast JSON String: " + forecastJsonStr);

        } catch (IOException e) {
            Log.e(LOG_TAG, "Error ", e);
            // If the code didn't successfully get the weather data, there's no point in attemping
            // to parse it.
            return null;
        } finally {
            if (urlConnection != null) {
                urlConnection.disconnect();
            }
            if (reader != null) {
                try {
                    reader.close();
                } catch (final IOException e) {
                    Log.e(LOG_TAG, "Error closing stream", e);
                }
            }
        }
```

The sequence of events is as follows:

1. When users click refresh in the options menu

2. The app passes the id string to the AsyncTask subclass FetchWeatherTask.

3. The AsyncTask method doInBackground() makes a URL string as a parameter and uses it to create a URL object.

4. The URL object is used to establish an HttpURLConnection.

5. Once the connection has been established, the HttpURLConnection object fetches the web page content as an InputStream.

6. The InputStream is read/buffered and converted to a string.

7. The string is passed to getWeatherDataFromJson()

JSON PARSING

ForeCastFragment.java

```
private String[] getWeatherDataFromJson(String forecastJsonStr, int numDays)
        throws JSONException {

    // These are the names of the JSON objects that need to be extracted.
    final String OWM_LIST = "list";
    final String OWM_WEATHER = "weather";
    final String OWM_TEMPERATURE = "temp";
    final String OWM_MAX = "max";
    final String OWM_MIN = "min";
    final String OWM_DESCRIPTION = "main";

    JSONObject forecastJson = new JSONObject(forecastJsonStr);
    JSONArray weatherArray = forecastJson.getJSONArray(OWM_LIST);

    //...
    Time dayTime = new Time();
    dayTime.setToNow();
    // we start at the day returned by local time. Otherwise this is a mess.
    int julianStartDay = Time.getJulianDay(System.currentTimeMillis(), dayTime.gmtoff);
    // now we work exclusively in UTC
    dayTime = new Time();

    String[] resultStrs = new String[numDays];
    for(int i = 0; i < weatherArray.length(); i++) {
        // For now, using the format "Day, description, hi/low"
        String day; String description; String highAndLow;

        // Get the JSON object representing the day
        JSONObject dayForecast = weatherArray.getJSONObject(i);

        //...
        long dateTime;
        // Cheating to convert this to UTC time, which is what we want anyhow
        dateTime = dayTime.setJulianDay(julianStartDay+i);
        day = getReadableDateString(dateTime);

        // description is in a child array called "weather", which is 1 element long.
        JSONObject weatherObject = dayForecast.getJSONArray(OWM_WEATHER).getJSONObject(0);
        description = weatherObject.getString(OWM_DESCRIPTION);

        //...
        JSONObject temperatureObject = dayForecast.getJSONObject(OWM_TEMPERATURE);
        double high = temperatureObject.getDouble(OWM_MAX);
        double low = temperatureObject.getDouble(OWM_MIN);

        highAndLow = formatHighLows(high, low);
        resultStrs[i] = day + " - " + description + " - " + highAndLow;
    }

    //...
    return resultStrs;

}
```

http://jsonviewer.stack.hu/

The sequence of events is as follows:

1. When users click refresh in the options menu

2. The app passes the id string to the AsyncTask subclass FetchWeatherTask.

3. The AsyncTask method doInBackground() makes a URL string as a parameter and uses it to create a URL object.

4. The URL object is used to establish an HttpURLConnection.

5. Once the connection has been established, the HttpURLConnection object fetches the web page content as an InputStream.

6. The InputStream is read/buffered and converted to a string.

7. The string is passed to getWeatherDataFromJson()

8. Finally, the AsyncTask's onPostExecute() method displays the string in the main activity's UI.

```java
@Override
protected void onPostExecute(String[] results) {
    //super.onPostExecute(strings);

    if (results != null) {
        mForecastAdapter.clear();
        //mForecastAdapter.addAll(result);
        for (String dayForeCastStr: results){
            mForecastAdapter.add(dayForeCastStr);
        }
    }
}
```

QUESTIONS?

Contact:

d.coyle@ucd.ie

Please ask in the Discussion Forum.

Next classes:

Saving data