# Software Engineering Large Practical: Accessing remote data

Stephen Gilmore
(Stephen.Gilmore@ed.ac.uk)
School of Informatics

October 12, 2016

# Contents

- More about system permissions
- Accessing remote data
- XML parsing

# Android system permissions

- By default, no Android application has permission to perform any operations that would adversely impact other applications, the operating system, or the user.
- This includes reading or writing the user's private data, reading or writing another application's files, performing network access, keeping the device awake, and so on.
- Applications statically declare the permissions they require, and the Android system prompts the user for consent.

---

`developer.android.com/guide/topics/security/permissions.html`

# Semantics of permissions

- When an Android application requests a permission to perform some action or access some service the question which is being asked is "*May I . . . ?*" not "*Can I . . . ?*".
- That is, even if the application has permission to access a service, there is no guarantee that the service is available.

| Permitted | Available | Result |
|---|---|---|
| No | No | Refused |
| No | Yes | Refused |
| Yes | No | Fail |
| Yes | Yes | Success |

# Granting permissions

The way in which Android asks the user to grant permissions depends on the system version, and the system version targeted by your app:

- If the device is running Android 6.0 "Marshmallow" (API level 23) or higher, and the app's targetSdkVersion is 23 or higher, the app requests permissions from the user *at run-time*.
- If the device is running Android 5.1 "Lollipop" (API level 22) or lower, or the app's targetSdkVersion is 22 or lower, the system asks the user to grant the permissions *when the user installs the app*.

---

`developer.android.com/guide/topics/security/permissions.html`

# Normal and dangerous permissions

System permissions are divided into several protection levels. The two most important protection levels to know about are *normal* and *dangerous* permissions:

- **Normal permissions** cover areas where your app needs to access data or resources outside the app's sandbox, but where there's very little risk to the user's privacy or the operation of other apps.
- **Dangerous permissions** cover areas where the app wants data or resources that involve the user's private information, or could potentially affect the user's stored data or the operation of other apps.

---

`developer.android.com/guide/topics/security/permissions.html`

# *Normal* permissions examples

- ACCESS_NETWORK_STATE
- ACCESS_WIFI_STATE
- BLUETOOTH
- CHANGE_WIFI_STATE
- INTERNET
- MODIFY_AUDIO_SETTINGS
- NFC
- SET_ALARM
- SET_TIME_ZONE
- SET_WALLPAPER
- VIBRATE
- ...

---

`developer.android.com/reference/android/Manifest.permission.html`

# *Dangerous* permissions examples

- READ_CALENDAR
- WRITE_CALENDAR
- CAMERA
- READ_CONTACTS
- WRITE_CONTACTS
- ACCESS_FINE_LOCATION
- ACCESS_COARSE_LOCATION
- RECORD_AUDIO
- READ_PHONE_STATE
- CALL_PHONE
- SEND_SMS
- READ_SMS
- READ_EXTERNAL_STORAGE
- WRITE_EXTERNAL_STORAGE
- ...

---

`developer.android.com/reference/android/Manifest.permission.html`

# Getting permission to access the internet

- Note that to perform any network operations, an application manifest must request the permissions: `android.permission.INTERNET` and `android.permission.ACCESS_NETWORK_STATE`.
- As before, permission is requested with the `uses-permission` element in your app manifest (`AndroidManifest.xml`).

```
1 <uses-permission android:name="android.permission.INTERNET" />
2 <uses-permission android:name="android.permission.
      ACCESS_NETWORK_STATE" />
```

`developer.android.com/training/basics/network-ops/connecting.html`

# Activity HttpExampleActivity (1/3)

```
1  public class HttpExampleActivity extends Activity {
2      private static final String DEBUG_TAG = "HttpExample";
3      private EditText urlText;
4      private TextView textView;
5
6      @Override
7      public void onCreate(Bundle savedInstanceState) {
8          super.onCreate(savedInstanceState);
9          setContentView(R.layout.main);
10         urlText = (EditText) findViewById(R.id.myUrl);
11         textView = (TextView) findViewById(R.id.myText);
12     }
```

developer.android.com/training/basics/network-ops/connecting.html

# Activity HttpExampleActivity (2/3)

```
13      // When user clicks button, calls AsyncTask.
14      // Before attempting to fetch the URL, makes sure that there is a
            network connection.
15      public void myClickHandler(View view) {
16          // Gets the URL from the UI's text field.
17          String stringUrl = urlText.getText().toString();
18          ConnectivityManager connMgr = (ConnectivityManager)
19              getSystemService(Context.CONNECTIVITY_SERVICE);
20          NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
21          if (networkInfo != null && networkInfo.isConnected()) {
22              new DownloadWebpageTask().execute(stringUrl);
23          } else {
24              textView.setText("No network connection available.");
25          }
26      }
```

developer.android.com/training/basics/network-ops/connecting.html

# Activity HttpExampleActivity (3/3)

```
27      private class DownloadWebpageTask extends AsyncTask<String, Void,
            String> {
28          @Override
29          protected String doInBackground(String... urls) {
30              // params comes from the execute() call: params[0] is the url
31              try {
32                  return downloadUrl(urls[0]);
33              } catch (IOException e) {
34                  return "Unable to retrieve web page.";
35              }
36          }
37          // onPostExecute displays the results of the AsyncTask.
38          @Override
39          protected void onPostExecute(String result) {
40              textView.setText(result);
41          }
42      }
43      ...
44  }
```

developer.android.com/training/basics/network-ops/connecting.html

# Asynchronous tasks

- `android.os.AsyncTask<Params, Progress, Result>` performs background operations and publishes results on the UI thread.
- It is designed to be a helper class around `Thread` and `Handler` and does not constitute a generic threading framework.
- AsyncTasks should ideally be used for short operations (a few seconds at the most.)
- An asynchronous task is defined by three generic types, called `Params`, `Progress` and `Result`, and four steps, called `onPreExecute`, `doInBackground`, `onProgressUpdate` and `onPostExecute`.

---

`developer.android.com/reference/android/os/AsyncTask.html`

# Method `downloadUrl` (1/2)

```
1  private String downloadUrl(String myurl) throws IOException {
2      InputStream is = null;
3      int len = 500;
4
5      try {
6          URL url = new URL(myurl);
7          HttpURLConnection conn = (HttpURLConnection) url.
               openConnection();
8          conn.setReadTimeout(10000 /* milliseconds */);
9          conn.setConnectTimeout(15000 /* milliseconds */);
10         conn.setRequestMethod("GET");
11         conn.setDoInput(true);
12         // Starts the query
13         conn.connect();
```

developer.android.com/training/basics/network-ops/connecting.html

# Method `downloadUrl` (2/2)

```
14          int response = conn.getResponseCode();
15          Log.d(DEBUG_TAG, "The response is: " + response);
16          is = conn.getInputStream();
17
18          // Convert the InputStream into a string
19          String contentAsString = readIt(is, len);
20          return contentAsString;
21
22      } catch (...) { ... TODO ... }
23
24      // Make sure that the InputStream is closed
25      finally {
26          if (is != null) {
27              is.close();
28          }
29      }
30  }
```

---

`developer.android.com/training/basics/network-ops/connecting.html`

# Method `readIt`

```
1  // Reads an InputStream and converts it to a String.
2  public String readIt(InputStream stream, int len) throws IOException,
       UnsupportedEncodingException {
3      Reader reader = null;
4      reader = new InputStreamReader(stream, "UTF-8");
5      char[] buffer = new char[len];
6      reader.read(buffer);
7      return new String(buffer);
8  }
```

developer.android.com/training/basics/network-ops/connecting.html

# Parsing XML on Android

The `XmlPullParser` interface defines an efficient and maintainable way to parse XML on Android. Android has had two implementations of this interface:

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

Either choice is fine. The following uses `ExpatPullParser`, via `Xml.newPullParser()`.

---

`developer.android.com/training/basics/network-ops/xml.html`

# Sample input

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <feed xmlns="http://www.w3.org/2005/Atom" xmlns:creativeCommons="
       http://backend.userland.com/creativeCommonsRssModule" ...">
3  <title type="text">newest questions tagged android</title>
4      <entry>
5          <id>http://stackoverflow.com/q/9439999</id>
6          <re:rank scheme="http://stackoverflow.com">0</re:rank>
7          <title type="text">Where is my data file?</title>
8           ...
9          <link rel="alternate" href="http://stackoverflow.com/questions
               /9439999/where-is-my-data-file" />
10          ...
11         <summary type="html">
12             <p>I have an Application that requires a data file ...</p>
13         </summary>
14     </entry>
15      ...
16 </feed>
```

---

`developer.android.com/training/basics/network-ops/xml.html`

# Instantiate the parser

```
1  public class StackOverflowXmlParser {
2      // We don't use namespaces
3      private static final String ns = null;
4
5      public List parse(InputStream in) throws XmlPullParserException,
           IOException {
6          try {
7              XmlPullParser parser = Xml.newPullParser();
8              parser.setFeature(XmlPullParser.
                   FEATURE_PROCESS_NAMESPACES, false);
9              parser.setInput(in, null);
10             parser.nextTag();
11             return readFeed(parser);
12         } catch( ... ) { ... /* TODO */ ... } finally {
13             in.close();
14         }
15     }
16   ...
17 }
```

developer.android.com/training/basics/network-ops/xml.html

# Reading the XML feed

```
1  private List readFeed(XmlPullParser parser) throws XmlPullParserException
       , IOException {
2      List entries = new ArrayList(); // TODO: Use generics
3
4      parser.require(XmlPullParser.START_TAG, ns, "feed");
5      while (parser.next() != XmlPullParser.END_TAG) {
6          if (parser.getEventType() != XmlPullParser.START_TAG) {
7              continue;
8          }
9          String name = parser.getName();
10         // Starts by looking for the entry tag
11         if (name.equals("entry")) {
12             entries.add(readEntry(parser));
13         } else {
14             skip(parser);
15         }
16     }
17     return entries;
18 }
```

developer.android.com/training/basics/network-ops/xml.html

# A class for storing entries

```
1  public static class Entry {
2      public final String title;
3      public final String link;
4      public final String summary;
5
6      private Entry(String title, String summary, String link) {
7          this.title = title;
8          this.summary = summary;
9          this.link = link;
10     }
11 }
```

developer.android.com/training/basics/network-ops/xml.html

```
1  private Entry readEntry(XmlPullParser parser) throws
       XmlPullParserException, IOException {
2      parser.require(XmlPullParser.START_TAG, ns, "entry");
3      String title = null;    String summary = null; String link = null;
4      while (parser.next() != XmlPullParser.END_TAG) {
5          if (parser.getEventType() != XmlPullParser.START_TAG)
6              continue;
7          String name = parser.getName();
8          if (name.equals("title")) {
9              title = readTitle(parser);
10         } else if (name.equals("summary")) {
11             summary = readSummary(parser);
12         } else if (name.equals("link")) {
13             link = readLink(parser);
14         } else {
15             skip(parser);
16         }
17     }
18     return new Entry(title, summary, link);
19 }
```

developer.android.com/training/basics/network-ops/xml.html

```
1  // Processes title tags in the feed.
2  private String readTitle(XmlPullParser parser) throws IOException,
       XmlPullParserException {
3      parser.require(XmlPullParser.START_TAG, ns, "title");
4      String title = readText(parser);
5      parser.require(XmlPullParser.END_TAG, ns, "title");
6      return title;
7  }
```

Documentation: "[`parser.require`] tests if the current event is of the given type and if the namespace and name do match. null will match any namespace and any name. If the test is not passed, an exception is thrown. The exception text indicates the parser position, the expected event and the current event that is not meeting the requirement."

---

`developer.android.com/training/basics/network-ops/xml.html`

```
1  // Processes link tags in the feed.
2  private String readLink(XmlPullParser parser) throws IOException,
       XmlPullParserException {
3      String link = "";
4      parser.require(XmlPullParser.START_TAG, ns, "link");
5      String tag = parser.getName();
6      String relType = parser.getAttributeValue(null, "rel");
7      if (tag.equals("link")) {
8          if (relType.equals("alternate")){
9              link = parser.getAttributeValue(null, "href");
10             parser.nextTag();
11         }
12     }
13     parser.require(XmlPullParser.END_TAG, ns, "link");
14     return link;
15 }
```

developer.android.com/training/basics/network-ops/xml.html

```
1  // Processes summary tags in the feed.
2  private String readSummary(XmlPullParser parser) throws IOException,
       XmlPullParserException {
3      parser.require(XmlPullParser.START_TAG, ns, "summary");
4      String summary = readText(parser);
5      parser.require(XmlPullParser.END_TAG, ns, "summary");
6      return summary;
7  }
```

developer.android.com/training/basics/network-ops/xml.html

```
1  // For the tags title and summary, extracts their text values.
2  private String readText(XmlPullParser parser) throws IOException,
       XmlPullParserException {
3      String result = "";
4      if (parser.next() == XmlPullParser.TEXT) {
5          result = parser.getText();
6          parser.nextTag();
7      }
8      return result;
9  }
```

Documentation: `parser.next` "Get next parsing event - element content will be coalesced and only one TEXT event must be returned for whole element content (comments and processing instructions will be ignored and entity references must be expanded or exception must be thrown if entity reference cannot be expanded). If element content is empty (content is "") then no TEXT event will be reported.

---

`developer.android.com/training/basics/network-ops/xml.html`

# Concluding remarks (1/2)

- Android APIs sometimes throw exceptions on failure and sometimes return null on failure: check the documentation to find out if a method can return null.
- There is no real reason to omit parameters to generic classes in modern Java code. Prefer `List<String>` to `List`.

# Concluding remarks (2/2)

Careful, defensive programming cares about catching the correct exceptions and handling them in a meaningful way. The user should not see low-level errors from apps.