Outline
• Connecting to the Internet
• Perform network operations
• Manage network usage
• Parsing data

Connect to the Internet

• Permissions in the manifest file
  <uses-permission android:name="android.permission.INTERNET"/>
  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>

• Create an HTTP client
  – Common approach for Android apps to send and receive data
  – HttpURLConnection: support HTTPS, streaming data, configurable timeouts, IPV6, ...

  URL url = new URL("http://www.android.com/");
  HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();

Connect to the Internet

• Check the network connection
  – Before your app attempts to connect to the network
  – Using setActiveNetworkInfo() and isConnected()

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

Perform Network Operations

• Start a separate thread
  – Network operations can involve unpredictable delays
  – E.g., using AsyncTask
    • doInBackground: network connection and operation
    • onPostExecute: update the UI

  1. The app parses the specified URL to the AsyncTask.
  2. The AsyncTask method doInBackground() calls the downloadUrl() method.
  3. The downloadUrl() method takes a URL string as a parameter and uses it to create a HttpURLConnection.
  4. The HttpURLConnection object is used to establish a connection. The code sets the read timeout and connection timeout.
  5. The HttpURLConnection object fetches the web page as an InputStream.
  6. The InputStream is passed to the read() method, which converts the stream to a string.
  7. Finally, the onPostExecute() method displays the string in the main activity's UI.

  1. The app parses the specified URL to the AsyncTask.
  2. The AsyncTask method doInBackground() calls the downloadUrl() method.
  3. The downloadUrl() method takes a URL string as a parameter and uses it to create a HttpURLConnection.
  4. The HttpURLConnection object is used to establish a connection. The code sets the read timeout and connection timeout.
  5. The HttpURLConnection object fetches the web page as an InputStream.
  6. The InputStream is passed to the read() method, which converts the stream to a string.
  7. Finally, the onPostExecute() method displays the string in the main activity's UI.
Use HttpURLConnection
- Connection is initiated when HttpURLConnection.connect() is called
- Apps are not always required to explicitly call connect method
  • Some operations such as getInputStream will implicitly perform the connection when necessary.
- Response header from the server: content type/length, encoding
  • An int representing the three digit HTTP Status-Code:1xx: Informational, 2xx: Success, 3xx: Redirection, 4xx: Client Error, 5xx: Server Error

Set the connection so they may be closed or reused.

While(true){
    try {
        readStream = urlConnection.getInputStream();
        readStream.close();
    }
    catch (IOException e) {
        display error
    }
}

Use HttpURLConnection
- Post content: upload data to a web server
  • Configure the connection for output using setDoOutput(true)
  • Call either setFixedLengthStreamingMode() or setChunkedStreamingMode() when the body length is known in advance, or setChunkedStreamingMode() when it is not.

readStream = urlConnection.openStream();
writeStream = urlConnection.getOutputStream();
writeStream.write(content);...
Manage Network Usage

- Fine-grained control of network resources
  - User specific settings, such as
    - How often your app syncs data?
    - Perform network operations only when on Wi-Fi?
    - Transfer data while roaming?

Manage Network Usage

- Network connection
  - Types of network connection (Wi-Fi or cellular)
  - Check the network state before performing network operations
    - ConnectivityManager and NetworkInfo
      - ConnectivityManager: Answers queries about the state of network connectivity. Notify applications when network connectivity changes
        - Monitor network connections (Wi-Fi, GPRS, UMTS, etc.)
        - Send broadcast intents when network connectivity changes
        - Attempt to "fail over" to another network when connectivity to a network is lost
        - Provide an API that allows applications to query the coarse-grained or fine-grained state of the available networks
      - Provide an API that allows applications to request and select networks for their data traffic

```java
ConnectivityManager connMgr = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
if (networkInfo != null && networkInfo.isConnected()) {
    Log.d(DEBUG_TAG, "Mobile connected:");
} else {
    Log.d(DEBUG_TAG, "Mobile disconnected:");
}
```

Manage Network Usage

- Preference of network usage
  - For example: allow users to upload videos only when the device is connected to a Wi-Fi network; sync (or not) depending on specific criteria such as network availability, time interval, ...

- Required permissions
  - android.permission.INTERNET — Allows applications to open network socket
  - android.permission.ACCESS_NETWORK_STATE — Allows applications to access information about networks
Manage Network Usage

- Preference of network usage
  - MANAGE_NETWORK_USAGE Intent Filter: shows settings for managing the network data usage of a specific application.
  - If your app includes an activity that offers options to control data usage
    ```
    <activity>
      <meta-data android:name="android.intent.category.DEFAULT" android:label="SettingsActivity" />
      <intent-filter>
        <action android:name="android.intent.action.MANAGE_NETWORK_USAGE" />
        <category android:name="android.intent.category.DEFAULT" />
      </intent-filter>
    </activity>
    ```

- Settings
  - A Preference object represents a single setting
  - Key-value pairs are stored in a default SharedPreferences file
  - PreferenceActivity or PreferenceFragment
  - Different types of preferences
    - CheckBoxPreference
    - ListPreference
    - EditTextPreference
  - Define Preferences in XML

- Preference of network usage
  - Implement a preferences activity
    ```java
    @Override
    public void onSharedPreferenceChanged(SharedPreferences sharedPreferences, String key) {
      // Sets refreshDisplay to true so that when the user returns to the main activity, the display refreshes to reflect the new settings.
      NetworkActivity.refreshDisplay = true;
    }
    ```

- Respond to preference changes
  - Get the current network state
    ```java
    public void updateConnectedFlags() {
      ConnectivityManager connMgr = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
      NetworkInfo activeInfo = connMgr.getActiveNetworkInfo();
      if (activeInfo != null && activeInfo.isConnected()) {
        wifiConnected = activeInfo.getType() == ConnectivityManager.TYPE_WIFI;
        mobileConnected = activeInfo.getType() == ConnectivityManager.TYPE_MOBILE;
      } else {
        wifiConnected = false;
        mobileConnected = false;
      }
    }
    ```

- Check the network preference
  ```java
  public void loadPage() {
    if (((sPref.equals(ANY)) && (wifiConnected || mobileConnected)) || ((sPref.equals(WIFI)) && (wifiConnected))) {
      // AsyncTask subclass
      new DownloadXmlTask().execute(URL);
    } else {
      showErrorPage();
    }
  }
  ```
Manage Network Usage

• Parsing data (XML)
  – Choose a parser (XmlPullParser)
    • via `Xml.newPullParser()`
  – Analyze the feed

```
<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>
  <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>
```

• Parsing data (XML)
  – Instantiate the parser via `Xml.newPullParser()`
  • Data fed by an `InputStream`
    • Start parsing with `nextTag()`

```
public List parse(InputStream in) throws XmlPullParserException, IOException {
    try {
        XmlPullParser parser = Xml.newPullParser();
        parser.setFeature(XmlPullParser.FEATURE_PROCESS_NAMESPACES, false);
        parser.setInput(in, null);
        parser.nextTag();
        return readFeed(parser);
    } finally {
        in.close();
    }
}
```

• Parsing data (XML)
  – Read the feed
    • Look for elements tagged "entry"
  • Return a List containing the entries

```
private List readFeed(XmlPullParser parser) throws XmlPullParserException,
IOException {
    List entries = new ArrayList();
    parser.require(XmlPullParser.START_TAG, ns, "feed");
    while (parser.next() != XmlPullParser.END_TAG) {
        if (parser.getEventType() != XmlPullParser.START_TAG) {
            continue;
        }
        String name = parser.getName();
        if (name.equals("entry")) {
            entries.add(readEntry(parser));
        } else {
            skip(parser);
        }
    }
    return entries;
```

• Parsing data (XML)
  – Parse each entry
    • A "read" method for each tag you're interested in. For example, `readEntry()`, `readTitle()`, and so on.

```
private Entry readEntry(XmlPullParser parser) throws XmlPullParserException, IOException {
    parser.require(XmlPullParser.START_TAG, ns, "entry");
    String title = null;
    String summary = null;
    String link = null;
    while (parser.next() != XmlPullParser.END_TAG) {
        if (parser.getEventType() != XmlPullParser.START_TAG) {
            continue;
        }
        String name = parser.getName();
        if (name.equals("title")) { title = readTitle(parser); }
        else if (name.equals("summary")) { summary = readSummary(parser); }
        else if (name.equals("link")) { link = readLink(parser); }
        else { skip(parser); }
    }
    return new Entry(title, summary, link);
```

HW3

1. Read a user input (city's name or zip code)
2. Display the temperature
3. Display the weather icon
4. Show the map of the city

• `openweathermap.org`
  – References
    • https://openweathermap.org/current
    • http://openweathermap.org/weather-conditions
  – Query examples
    • http://api.openweathermap.org/data/2.5/weather?q=boston,ma&APPID=f9a0da7858696d1453d0faa23006c2d9
    • http://api.openweathermap.org/data/2.5/weather?q=12345678901234567890&APPID=f9a0da7858696d1453d0faa23006c2d9
HW3

Main steps

- Get the user input and form the http URL
- Then, download the data from the server
- Analyze/parse the returned data
  - Temperature
  - Icon file name
  - Latitude, longitude
- Download the icon file

Parse JSON data

```java
JSONObject jobj = new JSONObject(inputString);
JSONArray jsonArray = jobj.getJSONArray("coord");
```

Manage Network Usage

Settings

```java
public class SettingsFragment extends PreferenceFragment {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Load the preferences from an XML resource
        addPreferencesFromResource(R.xml.preferences);
    }
}
```

```java
public class SettingsActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Display the fragment as the main content.
        getFragmentManager().beginTransaction().replace(android.R.id.content, new SettingsFragment()).commit();
    }
}
```