# Connexion au réseau

# Se connecter au réseau

- Les connections et les opérations « réseau » nécessitent des autorisations à inclure dans le fichier de configuration :

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

# Vérification de la connexion au réseau

- Avant d'essayer d'établir une connexion au réseau, il faut vérifier si une connexion est disponible, utilisation de :
  - getActiveNetworkInfo()
  - 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
    }
    ...
}
```

# Réalisation de la connexion au réseau dans un thread séparé

- Les connexions au réseau sont souvent liées à des délais relativement plus lents que le reste des opérations d'une application.

- Il est recommandé d'effectuer ces opérations dans des threads séparés par rapport à celui de l'interface utilisateur.

- La classe AsyncTask offre un moyen simple pour effectuer une tâche dans un thread autre que celui de l'interface utilisateur.

```
public class HttpExampleActivity extends Activity {
    private static final String DEBUG_TAG = "HttpExample";
    private EditText urlText;
    private TextView textView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        urlText = (EditText) findViewById(R.id.myUrl);
        textView = (TextView) findViewById(R.id.myText);
    }

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

```
    // Uses AsyncTask to create a task away from the main UI thread. This task takes a
    // URL string and uses it to create an HttpUrlConnection. Once the connection
    // has been established, the AsyncTask downloads the contents of the webpage as
    // an InputStream. Finally, the InputStream is converted into a string, which is
    // displayed in the UI by the AsyncTask's onPostExecute method.


     private class DownloadWebpageTask extends AsyncTask<String, Void, String> {
        @Override
        protected String doInBackground(String... urls) {

            // params comes from the execute() call: params[0] is the url.
            try {
                return downloadUrl(urls[0]);
            } catch (IOException e) {
                return "Unable to retrieve web page. URL may be invalid.";
            }
        }
        // onPostExecute displays the results of the AsyncTask.
        @Override
        protected void onPostExecute(String result) {
            textView.setText(result);
       }
    }
    ...
}
```

# Connexion et chargement des données

- Choix du client HTTP:
  - La plupart des applications Android connectées utilisent *http* pour envoyer et recevoir des données
  - Android inclut deux clients *HTTP* :
    - HttpURLConnection → disponible via la classe *HttpURLConnection*
      - Cette classe est utilisée pour envoyer et recevoir des données sur le web.
      - Les données peuvent être de n'importe quel type et longueur.
    - Apache HttpClient

  - Les concepteurs d'Android recommandent d'utiliser HttpURLConnection pour les applications basées sur les versions *Gingerbread* (*Android 2.3*)et plus.

# Connexion et chargement des données

- Utilisation de HttpURLConnection pour réaliser un GET et télécharger des données.
- Appel de connect(),
- Appel de getInputStream() pour avoir un InputStream
  - Retourne le code lié au statut de la connexion, utile pour avoir des informations complémentaires par rapport à la connexion.
    - Le code 200 indique le succès de la connexion.

```
// Given a URL, establishes an HttpUrlConnection and retrieves the web page content
//as an InputStream, which it returns as a string.

private String downloadUrl(String myurl) throws IOException {
    InputStream is = null;
 // Only display the first 500 characters of the retrieved web page content.
    int len = 500;

 try {
      URL url = new URL(myurl);
      HttpURLConnection conn = (HttpURLConnection) url.openConnection();
      conn.setReadTimeout(10000 /* milliseconds */);
      conn.setConnectTimeout(15000 /* milliseconds */);
      conn.setRequestMethod("GET");
      conn.setDoInput(true);
      // Starts the query
      conn.connect();
      int response = conn.getResponseCode();
      Log.d(DEBUG_TAG, "The response is: " + response);
      is = conn.getInputStream();

       // Convert the InputStream into a string
      String contentAsString = readIt(is, len);
      return contentAsString;

//Makes sure that the InputStream is closed after the app is finished using it.
    } finally {
        if (is != null) {
```

```
// Uses AsyncTask to create a task away from the main UI thread. This task takes a  URL string and uses it
//to create an HttpUrlConnection. Once the connection has been established, the AsyncTask downloads
//the contents of the webpage a an InputStream. Finally, the InputStream is converted into a string,
//which is displayed in the UI by the AsyncTask's onPostExecute method.
    private class DownloadWebpageTask extends AsyncTask<String, Void, String> {
        @Override
        protected String doInBackground(String... urls) {

            // params comes from the execute() call: params[0] is the url.
            try {
                return downloadUrl(urls[0]);
            } catch (IOException e) {
                return "Unable to retrieve web page. URL may be invalid.";
            }
        }
        // onPostExecute displays the results of the AsyncTask.
        @Override
        protected void onPostExecute(String result) {
            textView.setText(result);
        }
    }
    ...
}
```

# Convertir un InputStream à String

- Un InputStream est un fichier de bytes où il est commode de le convertir en un autre type de données.

```
InputStream is = null;
...
Bitmap bitmap = BitmapFactory.decodeStream(is);
ImageView imageView = (ImageView) findViewById(R.id.image_view);
imageView.setImageBitmap(bitmap);
```

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

# Parser des données XML

- XML est un format de description de données très utilisé sur Internet.
  - Exemple :
    - Les sites internet qui changent fréquemment leurs contenus, comme les sites des news et les blogues.
    - Ces sites fournissement un flux XML pour les applications externes.

- Télécharger et parser des données XML est une tâche très courante pour les applications connectées.

- Choix du parseur
  - Les concepteurs d'Android recommande XmlPullParse, pour parser des documents XML sur.
  - Deux implémentations sont disponibles :
    - XmlParser via XmlPullParserFactory.newPullParser().
    - ExpatPullParser, via Xml.newPullParser().

# Analyse du flux XML

- Instancier le parseur et démarrer le processus d'analyse du flux.
- Appel aux méthodes nextTag() et readFeed(), qui extrait et analyse les données qui intéressent l'application.

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom" xmlns:creativeCommons="http://backend.userland.com/
creativeCommonsRssModule" ...">
<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>
```

# Analyse du flux XML

```
public class StackOverflowXmlParser {
    // We don't use namespaces
    private static final String ns = null;

    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();
        }
    }
 ...
}
```

# Lecture du flux

- La méthode readFeed() analyse le flux.
- Cette méthode identifie les éléments à retourner comme un point de départ pour une analyse récursive du flux.
- Les autres éléments sont ignorés.
- Quand tout le fichier est analysé la méthode readFeed() retourne une liste qui contient des éléments cibles avec leurs éléments fils.

# Lecture du flux

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

# Lecture du flux

- Création des méthodes suivantes :
  - Une méthode de lecture pour chaque éléments à utiliser dans l'application.
    - Par exemple, readEntry(), readTitle(), etc..
      - Le parseur lit les éléments du fichier source. Quand il rencontre un élément entry, title, link ou summary, appelle la méthode de lecture correspondante. Sinon, il garde l'élément.
  - Les méthodes pour extraire les données de chaque type différent d'élément et pour que le parseur avance vers l'élément suivant.
    - Par exemple:
      - Pour les éléments title et summary, le parseur appelle readText().
  - Une méthode skip()

```
public static class Entry {
    public final String title;
    public final String link;
    public final String summary;

    private Entry(String title, String summary, String link) {
        this.title = title;
        this.summary = summary;
        this.link = link;
    }
}

// Parses the contents of an entry. If it encounters a title, summary, or link tag, hands them off
// to their respective "read" methods for processing. Otherwise, skips the tag.
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);
}
```

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

// Processes link tags in the feed.
private String readLink(XmlPullParser parser) throws IOException, XmlPullParserException {
    String link = "";
    parser.require(XmlPullParser.START_TAG, ns, "link");
    String tag = parser.getName();
    String relType = parser.getAttributeValue(null, "rel");
    if (tag.equals("link")) {
        if (relType.equals("alternate")){
            link = parser.getAttributeValue(null, "href");
            parser.nextTag();
        }
    }
    parser.require(XmlPullParser.END_TAG, ns, "link");
    return link;
}
```

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

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

# Ignorer des éléments

```
private void skip(XmlPullParser parser) throws XmlPullParserException, IOException {
    if (parser.getEventType() != XmlPullParser.START_TAG) {
        throw new IllegalStateException();
    }
    int depth = 1;
    while (depth != 0) {
        switch (parser.next()) {
        case XmlPullParser.END_TAG:
            depth--;
            break;
        case XmlPullParser.START_TAG:
            depth++;
            break;
        }
    }
}
```