DEVELOPPEMENT MOBILE

1 Interroger un WebService

1.1 Objectif

L'objectif de ce TP est de comprendre le mécanisme permettant d'accéder à une source de donnée externe de type WebService REST.

1.2 Contrainte d'accès au WebService

Une source de donnée de type WebService REST présente des données de type objet, liste d'objets ou combinaison d'objets et de listes et ceci sous un format XML ou JSON.
Cela permet de protéger la source des données et permet la communication entre machines à travers le protocole HTTP.

Dans notre cas, nous allons utiliser un service d'exemple : http://jsonplaceholder.typicode.com/

Créez une nouvelle application.
Comme pour tout accès internet, nous aurons besoin de la permission correspondante :

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

Nous souhaitons accéder et lister les publications effectuer par l'utilisateur n°1, la requête est la suivante :
GET http://jsonplaceholder.typicode.com/posts?userId=1
Le téléchargement des données se fait comme une simple requête web, on obtient à l'arrivée une chaine de caractère contenant toute les données (au format **JSON** dans notre cas).

L'annexe 1 propose une implémentation pour télécharger un fichier texte à partir d'une requête GET sur une URL.

1.3 Le traitement des données

Les données doivent ensuite être converties en objet. Nous avons pour cela besoin de l'objet « **Post** » de destination comme modèle de référence et d'un convertisseur.
Plusieurs bibliothèques proposent la conversion depuis le format JSON, nous allons prendre la dépendance Gson et le code de conversion proposés dans **l'annexe 2**.
Android est très peu tolérant sur le blocage de l'écran, les applications doivent donc effectuer toutes les tâches longues à partir de Thread. C'est bien évidemment le cas des accès internet.
Pour effectuer ce travail, nous allons utiliser le mécanisme des AsyncTask. Ces tâches asynchrones sont le mécanisme de Thread proposé par Android, censé être plus « propre » et permettant de maintenir une communication avec l'interface graphique pendant son exécution, sans bloquer l'affichage.

Créez cette tâche, dans le « **doInBackground** », téléchargez les données à partir de la classe « **HttpUtils** » puis tout de suite convertissez les en liste d'objets de type **Post** à l'aide de la classe « **JsonUtils** ».
Il restera à faire l'affichage dans une « **ListView** » depuis le « **onPostExecute** ».

Annexe 1: La classe de téléchargement

```
import android.util.Log;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
public class HttpUtils {
    /**
     * Found on http://developer.android.com/training/basics/networkops/
     connecting.html
     * @param myurl http web url
     * @return web page content
     * @throws IOException
     */
    public static String downloadUrl(String myurl) throws IOException {
        InputStream is = null;
        try {
            // http://172.28.106.1:8080/fr.ensicaen.si.jersey/si/clients
            URL url = new URL(myurl);
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            conn.setReadTimeout(10000); // milliseconds
            conn.setConnectTimeout(15000); // milliseconds
            conn.setRequestProperty("Content-Type", "application/json");
            conn.setRequestProperty("Accept", "application/json");
            conn.setRequestMethod("GET");
            conn.setDoInput(true);
            // Starts the query
            conn.connect();
            int response = conn.getResponseCode();
            Log.i(HttpUtils.class.getSimpleName(), "The response is: " + response);
            is = conn.getInputStream();
            // Convert the InputStream into a string
            //String contentAsString = readIt(is, len);
            String contentAsString = readIt(is);
            return contentAsString;
            // Makes sure that the InputStream is closed after the app is
            // finished using it.
        } finally {
            if (is != null) {
                is.close();
            }
        }
    }
    /**
     * Download a text file line by line until end of file.
     * @return a string with the whole text file.
     */
    public static String readIt(InputStream stream) throws IOException {
        StringBuilder stringBuilder = new StringBuilder();
        BufferedReader reader = new BufferedReader(new InputStreamReader(stream));
        String line;
        while ((line = reader.readLine()) != null) {
            stringBuilder.append(line + '\n');
        }
        return stringBuilder.toString();
    } }
```

Annexe 2: La classe de parsing

Nécessite la librairie GSON, la dépendance à declarer est : **com.google.code.gson:gson:2.3.1**

Voici le code pour parser les objets

```
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.JsonSyntaxException;
import java.lang.reflect.Type;

public class JsonUtils {
    private static JsonUtils instance = null;
    private Gson gson;
    private JsonUtils() {
        gson = new GsonBuilder().create();
    }
    public static JsonUtils getInstance() {
        if (instance == null) {
            instance = new JsonUtils();
        }
        return instance;
    }
    public static List<Post> parsePosts(String json) throws JsonSyntaxException {
        Type listType = new TypeToken<List<Post>>(){}.getType();
        return getInstance().gson.fromJson(json, listType);
    }
}
```