Chapitre 06

Internet

Accès à l'internet

- Il faut ajouter dans le fichier « AndroidManifest.xml » les autorisations nécessaires pour permettre l'accès à l'internet, comme suit :

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

public static final String INTERNET
Added in API level 1
Allows applications to open network sockets.
Constant Value: "android.permission.INTERNET"

- Si nous voulons vérifier l'état de la connexion à internet, il est nécessaire d'autoriser cette action dans le fichier « AndroidManifest.xml » comme suit :

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

public static final String ACCESS_NETWORK_STATE
Added in API level 1
Allows applications to access information about networks
Constant Value: "android.permission.ACCESS_NETWORK_STATE"

- Ces deux permissions entrent dans la catégorie des permissions « normales » :

- Une permission normale est soumise à une « PROTECTION_NORMAL ». L'utilisateur n'a pas besoin de l'accorder. Elle n'est pas considérée comme une permission à risque.

- Cette permission est accordée de facto à l'installation de l'application.

- Une application qui demande à accéder aux contacts par exemple nécessite un accord préalable.

- Elle est considérée comme étant une permission dangereuse vu le caractère privé des données.

- La liste des permissions « normales » depuis l'API21 est :

  https://developer.android.com/guide/topics/permissions/normal-permissions.html

- Vérifier l'état de la connexion :

On commence par récupérer le service « CONNECTIVITY_SERVICE », responsable de la connectivité réseau de notre appareil, à travers une instance du type « ConnectivityManager » :

```
ConnectivityManager connMgr =
        (ConnectivityManager)
    getSystemService(Context.CONNECTIVITY_SERVICE);
```

Pour cette instance, nous récupérons des informations sur l'état d'activité du réseau.

```
Network network = connMgr.getActiveNetwork();
```

Nous testons par exemple l'accès à l'internet :

- nc.hasCapability(NetworkCapabilities.NET_CAPABILITY_INTERNET)

Examiner l'exemple « **C06 : TestInternet** »

Effectuer les tests en activant puis en désactivant le réseau, en utilisant par exemple le mode avion.

Afficher des pages web

- On peut utiliser un TextView (examiner l'exemple « **C04: StringsDemo** », chapitre 4) pour afficher une page « html ». Or cette page est bien limitée, vu qu'il n'est pas possible d'afficher une image.

- Android offre un mécanisme permettant d'afficher une page HTML dans une activité. On utilise pour cela une vue du type WebView.

- Commencer par ajouter une telle vue dans le fichier « XML » associé à votre activité :

```
<WebView
    android:id="@+id/webview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" />
```

- Par la suite, on récupère la vue dans le code Java et on charge la page web en question :

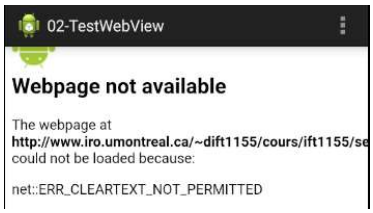
  webview.loadUrl("http://www.iro.umontreal.ca/~dift1155");

- Il faudra la permission permettant à l'activité de se connecter à l'internet :

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

  Faire le test avec et sans la permission (la retirer du fichier AndroidManifest.xml).

- « Google » force l'utilisation du protocole sécurisé « https ». Si on utilise au contraire le protocole « http » pour afficher une page web, on obtient ce message d'erreur :



  https://developer.android.com/training/articles/security-config#CleartextTrafficPermitted

« Starting with Android 9.0 (API level 28), cleartext support is disabled by default. »

Si on veut quand même transmettre la page en claire, on dispose de plusieurs options :

https://stackoverflow.com/questions/51591208/noclassdeffounderror-exception-in-android-stuidio

Une des solutions consiste à ajouter dans le fichier « AndroidManifest.xml » de l'application, la section « application », le passage suivant :

```
android:usesCleartextTraffic="true"
```

- Si on veut activer JavaScript dans la vue :

  webview.getSettings().setJavaScriptEnabled(true);

- Gestion des liens dans une page web

  Quand vous cliquez un lien, Android lance le navigateur approprié pour afficher la page en question.

  Vous pouvez modifier ce comportement et permettre à votre activité (webview) de gérer ce type d'affichage.

  webview.setWebViewClient(new myWebViewClient());

  On définit donc une classe dans laquelle nous allons redéfinir la méthode « ShouldOverrideUrlLoading ».

```
private class myWebViewClient extends WebViewClient {

    // API < 24
    @SuppressWarnings("deprecation")
    @Override
    public boolean shouldOverrideUrlLoading(WebView view, String url) {
          view.loadUrl(url);
          return true;
    }

    // API 24+
    @TargetApi(Build.VERSION_CODES.N)
    @Override
    public boolean shouldOverrideUrlLoading(WebView view,\
                                            WebResourceRequest request) {
          view.loadUrl(request.getUrl().toString());
     return true;
    }
}
```


- On peut aussi gérer le comportement du bouton retour associé à un navigateur afin de revenir à la page web visualisée précédemment, comme suit :

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
   if ((keyCode == KeyEvent.KEYCODE_BACK) && webview.canGoBack()) {
      webview.goBack(); // voir les différentes possibilités
      return true;
   }
   return super.onKeyDown(keyCode, event);
}
```

Examiner l'exemple « **C06 : TestWebView** »

Tester le comportement de la touche « retour ».

Fabriquer une page localement

- On peut fabriquer la page à afficher et la charger par la suite dans une instance de « WebView » avec la commande « loadData » comme suit :

  webview.loadData(Contenu,Typemime,Encodage);

  Les arguments de la méthode « loadData » sont du type « string » :

  - « Contenu » : représente le code HTML de la page web à afficher.

  - « Typemime » : le format des données reliées à la page, dans notre exemple nous allons utiliser le format « text/html ». La liste est disponible ici :

    http://www.iana.org/assignments/media-types/media-types.xhtml

    http://fr.wikipedia.org/wiki/Type_MIME

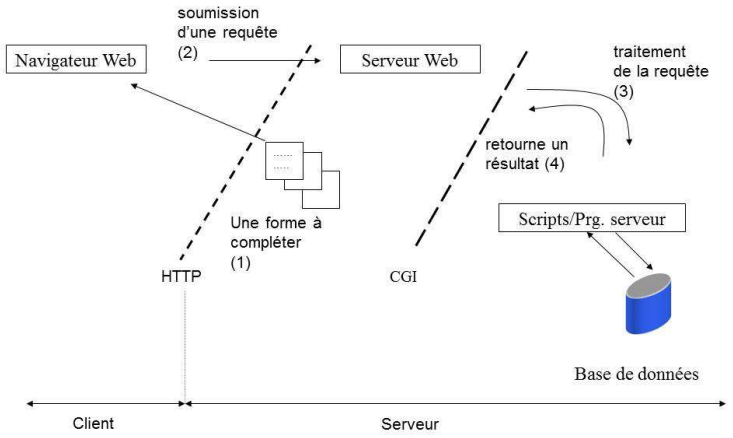
  - « Encodage » : est l'encodage des caractères dans la page web en question, nous allons utiliser « UTF-8 ». La liste complète est disponible ici :

    http://fr.wikipedia.org/wiki/Codage_de_caract%C3%A8res

Examiner l'exemple « **C06 : TestWebViewLoadData** »

Requête HTTP

- Une application web est constituée de 2 niveaux, le client émettant une requête et le serveur traitant cette requête.



(1) Un serveur web soumet une forme d'un document à compléter (une requête),

(2) Le client complète la forme et la retourne au serveur,

(3) Le serveur la réceptionne et la redirige vers le script qui l'a invoquée dans la page HTML. En exécutant cette requête, le programme peut communiquer avec une ou plusieurs bases de données si nécessaire. Ainsi donc, le programme analyse les données soumises, peut éventuellement les sauvegarder ou collecter d'autres données pour construire dynamiquement une page HTML.

(4) Ces programmes retournent les résultats au serveur qu'il se charge de les transmettre au client.

Le client peut utiliser :

- un formulaire HTML permettant de saisir les champs, ou l'utilisation de scripts (JavaScript ou autre) pour faire la validation ;

- des applets et communiquer avec le serveur via des sockets/RMI.

Les requêtes HTTP vers le serveur contiennent :

- l'URL de la ressource à accéder,
- la requête GET pour extraire des informations sur le serveur,
- la requête POST pour modifier les données sur le serveur.

Le serveur identifie avec la requête le type d'environnement d'exploitation à charger en fonction de :

- l'extension du fichier (.cgi, .jsp, etc.) ou
- le répertoire où il se trouve (cgi-bin/, servlet/, etc.).

Le serveur charge par la suite l'environnement d'exécution Perl (cgi-perl), JVM (servlets), etc.

Le script ou programme précise le type du contenu (HTML, images, etc.) et intègre la réponse dans un flot associé à la sortie.

Le navigateur définit le type MIME (l'encodage utilisé pour le transfert de documents multimédias à travers le réseau) text/HTML audio/basic image/gif etc. et affiche les données en fonction.

Clients HTTP

- Android inclut deux clients HTTP : « HttpURLConnection » natif du langage Java et « HttpClient » d'Apache. Il est conseillé par « Android » d'utiliser le client « HttpURLConnection » pour les applications qui ciblent la version Gingerbread (9+), car ce client est constamment maintenu par « Android » :

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

HttpURLConnection

- Origine : Java, du paquetage « java.net. HttpURLConnection ».

  Création d'une instance « url » en lui communiquant l'adresse « urlStr » que nous désirons communiquer avec :

  URL url = new URL(urlStr);

Établir une connexion à partir de l'instance "url".

```
URLConnection urlConn = url.openConnection();
```

On s'assurer qu'il s'agit du bon protocole, HTTP, et non pas FTP, file (fichier), etc.

```
if (!(urlConn instanceof HttpURLConnection)) {
      throw new IOException ("URL is not an Http URL");
}
```

Si c'est OK, on passe à l'étape de configuration de la connexion:

```
HttpURLConnection httpConn = (HttpURLConnection)urlConn;
```

Est-ce que l'application peut demander une information supplémentaire, « false » pour une réponse négative. À noter que cette option n'est pas utilisée sous Android.

```
httpConn.setAllowUserInteraction(false);
```

L'adresse « url » du site web du cours IFT1155, www.iro.umontreal.ca/~dift1155, renvoie vers la session en cours. Pour valider ce renvoi d'url, il faut utiliser :

```
httpConn.setInstanceFollowRedirects(true);
```

Nous utilisons « GET » pour récupérer de l'information, sinon il fallait utiliser « POST » pour poster l'information :

```
httpConn.setRequestMethod("GET");
```


On établit une connexion:

```
httpConn.connect();
```

On examine par la suite la réponse transmise par le serveur

```
resCode = httpConn.getResponseCode();
if (resCode == HttpURLConnection.HTTP_OK) {
   in = httpConn.getInputStream();
}
```

Plusieurs codes sont possibles :

http://developer.android.com/reference/java/net/HttpURLConnection.html
http://fr.wikipedia.org/wiki/Liste_des_codes_HTTP

HTTP_OK: "Requête traitée avec succès »

Ayant obtenu les accès nécessaires à l'url et attaché un flux à cette connexion, nous allons lire les données (l'opération « GET »).

Pour cela, il est nécessaire d'utiliser les threads pour éviter un « ANR » ou bien d'aller modifier le thread « UI », voir le chapitre « 05, les threads ».

Examiner l'exemple « **C06 : HttpURLConnectionA** »

L'exemple décrit comment télécharger un texte et une image.

Nous avons utilisé 2 boutons pour démarrer indépendamment le téléchargement du texte et de l'image. Pour chacun de ces téléchargements, nous avons ajouté une barre de progression afin d'informer l'utilisateur de l'état du téléchargement.

Dans les deux cas de figure, nous avons utilisé des threads pour réaliser ces opérations. Le handler s'est chargé par la suite de communiquer avec les threads afin de mettre à jour l'information sur le thread « UI ». Pour cela, il a utilisé un système de messagerie pour pouvoir communiquer sans problème avec les différents threads.

Image :

```
Message msg = Message.obtain();
msg.what = 1;
Bundle b = new Bundle();
b.putParcelable("bitmap", bitmap);
msg.setData(b);
messageHandler.sendMessage(msg);
```

Texte:

```
Message msg = Message.obtain();
msg.what=2;

Bundle b = new Bundle();
b.putString("text", text);
msg.setData(b);
messageHandler.sendMessage(msg);
```

```
switch (msg.what) {
        case 1:
           ImageView img = findViewById(R.id.imageview01);
           img.setImageBitmap(msg.getData().getParcelable("bitmap"));
           break;
        case 2:
           TextView text = findViewById(R.id.textview01);
           text.setText(msg.getData().getString("text"));
           break;
}
```


HttpClient

C'est un client natif à Apache.

Définir un client :

```
HttpClient client = new DefaultHttpClient();
```

Définir la requête avec l'adresse url à télécharger :

```
HttpGet request = new HttpGet(urlStr);
```

Exécuter la requête et récupérer la réponse :

```
HttpResponse response = client.execute(request);
```

On peut tester l'état de la réponse:

```
final int statusCode = response.getStatusLine().getStatusCode();
if (statusCode != HttpStatus.SC_OK) {
        Log.w("ImageDownloader", "Error " + statusCode +
                        " while retrieving data from " + urlStr);
        return null;
}

in = response.getEntity().getContent();
```

Examiner l'exemple « **C06 : HttpClientApache** »

Les codes de « HttpStatus » sont disponibles sur cette page :

http://developer.android.com/reference/org/apache/http/HttpStatus.html

Nous avons ajouté la même information dans le fichier « AndroidManifest.xml » :

```
<uses-library android:name="org.apache.http.legacy"
                        android:required="false"/>
```

Vu que cette librairie est dépréciée, nous avons désactivé dans l'interface les avertissements relatifs à ce sujet. Nous avons utilisé pour cela la déclaration « `@SuppressWarnings("deprecation")` » au début de chaque méthode utilisant cette librairie.

Si on retire cette déclaration, on obtient ce message :

HttpClientApache\app\src\main\java\ca\umontreal\iro\ift1155\httpclientapache\
HttpClientApache.java:

uses or overrides a deprecated API.

Recompile with -Xlint:deprecation for details.

Pour recompiler avec l'option suggérée, nous allons ajouter ces instructions dans le fichier « build.gradle » du projet, dans la section « allprojects » :

```
tasks.withType(JavaCompile) {

 options.compilerArgs << "-Xlint:unchecked" << "-Xlint:deprecation"

}
```

Après avoir recompilé le projet :

[deprecation] HttpStatus in org.apache.http has been deprecated

Le message affiché signale l'utilisation d'une librairie dépréciée.

Il faut ajouter ce qui suit dans le fichier « build.gradle » du module « app », section « android » :

```
useLibrary 'org.apache.http.legacy'


packagingOptions {
        resources.excludes.add("META-INF/DEPENDENCIES")
}
```

et dans la section « dependencies » :

```
implementation 'org.apache.httpcomponents:httpclient:4.5.13'
```

Socket

- Les sockets permettent de faire communiquer un processus avec un service qui gère un réseau.

- Un socket est constitué d'une adresse IP et d'un numéro de port.

- Le modèle de communication est défini par programmation (TCP, UDP, Etc.).

- Créer un socket :

```
Socket client = new Socket(nom_host,port_host);
```

- On peut envoyer des données via le socket:

```
PrintWriter  out = new PrintWriter(client.getOutputStream());
```


- On peut lire des données envoyées via le socket

```
BufferedReader in = new BufferedReader(
                    new InputStreamReader(incoming.getInputStream()));
```

- **Ne pas oublier de fermer le socket.**

Examiner l'exemple « **C06 : AndyChat** »

Le fichier « LisezMoi » inclus avec l'exemple contient les instructions comment utiliser l'application « AndyChat ».

JavaScript

- En activant JavaScript, vous permettez au webview d'afficher correctement les pages web réalisées en partie avec un tel langage.

- Il est possible de réaliser des interfaces qui échangeraient des données entre l'application Android et des pages web contenant du code JavaScript et hébergées localement.

HTML/JavaScript

Examiner l'exemple « **C06 : 10-1B-WebView-Local-Html** »

- L'exemple contient une page HTML locale et une application Android.

- L'application télécharge la page HTML et attend que l'utilisateur interagisse avec elle. Pour chaque clique sur le bouton, elle va afficher un « toast ».

- Le répertoire « assets » contient des données qui seront incluses avec l'application. On peut stocker dans ce répertoire des fichiers audio, vidéo, description de l'application, etc.

  Pour notre exemple, nous allons inclure dans ce répertoire le fichier HTML « my_local_webpage1.html ». Cette page contient ce qui suit :

```
<HTML>
    <input type="button"
           value="Say hello"
           onClick="showAndroidToast('Hello Android!')" />

    <script type="text/javascript">
        function showAndroidToast(toast) {
            AndroidInterface.showToast(toast);
        }
    </script>
</html>
```

  La page déclare un bouton et l'action associée, lors d'un clic sur ce bouton.

L'application Android définit le WebView qui doit héberger la page HTML :

```
<WebView
    android:id="@+id/webView1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_alignParentLeft="true" />
```

Un objet interface est attaché au WebView. Il va permettre la communication entre l'application Android et à la page HTML :

```
browser.addJavascriptInterface(
                new JavaScriptInterface(this),
                "AndroidInterface");
```

On charge la page dans la WebView :

```
browser.loadUrl("file:///android_asset/my_local_webpage1.html");
```


La méthode « showToast » sera par la suite invoquée par la méthode « onClick » définie dans le JavaScript associée à la page HTML. Le même évènement va permettre de récupérer l'argument transmis à la méthode « showToast ».

```
public void showToast(String toast) {
    Toast.makeText(mContext, toast,Toast.LENGTH_SHORT).show();
}
```

Fonctionnement : quand vous cliquez sur le bouton « Say hello », ce dernier va faire appel à la méthode « showAndroidToast » avec l'argument « Hello Android! ». Cette méthode va utiliser l'objet interface « AndroidInterface », qui sert à communiquer entre la page HTML et l'application Android. Cet objet est lié à une instance « JavaScriptInterface ». Il peut donc accéder aux méthodes de cette classe. Il va donc faire appel « showToast(toast) » en lui passant comme argument « Hello Android! ». Cette méthode va avoir la charge par la suite d'afficher la chaîne en question.

Cet exemple à montrer comment interconnecter une application Android et un programme écrit en JavaScript.

- Examiner les deux exemples suivants :

« C06 : 10-2-WebView-PassingObject-JS »

« C06 : 10-3-WebView-GoogleMapV3-Fixed-Location »

Le fichier « LisezMoi » inclus avec chacun des exemples contient plus de détails sur ces deux exemples.

JSON (JavaScript Object Notation)

- JSON est un format léger d'échange de données.
- JSON est indépendant de tout langage, facile à apprendre, car sa syntaxe est réduite et non extensible.
- JSON est plus performant que XML, vu que son format est plus allégé et compact. Du coup, il s'y prête mieux à des applications Android vu les limites techniques des appareils.

- Android contient un paquetage JSON responsable de traiter des éléments JSON.

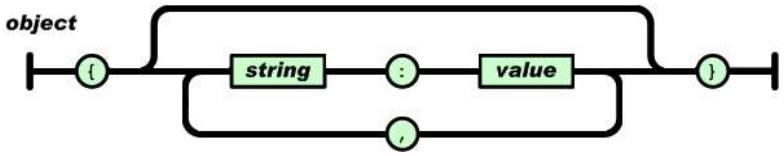
	XML	JSON
1	`<employes>`	`{`
2	`  <employe>`	`  "employes": [`
3	`  <nom> Eric Schmidt </nom>`	`  {`
4	`  <sexe>homme</sexe>`	`  "nom": " Eric Schmidt ",`
5	`  </employe>`	`  "sexe": "homme"`
6	`  <employee>`	`  },`
7	`  <nom> Marissa Mayer </nom>`	`  {`
8	`  <sexe>femme</sexe>`	`  "nom": " Marissa Mayer ",`
9	`  </employee>`	`  "sexe": "femme"`
10	`</employes>`	`  }`
11		`  ]`
12		`}`

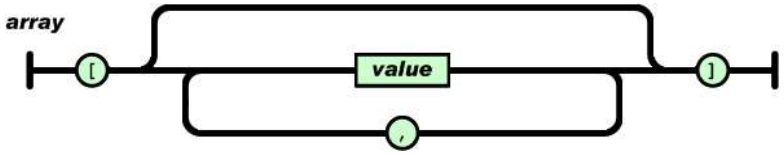
Examiner l'exemple « **C06 : JsonParsing**»

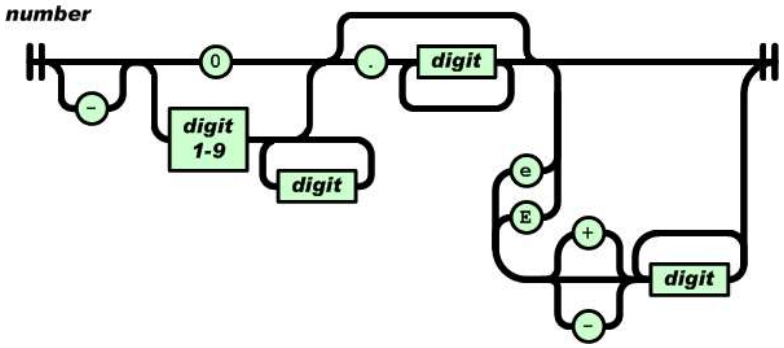
Structure d'un document JSON

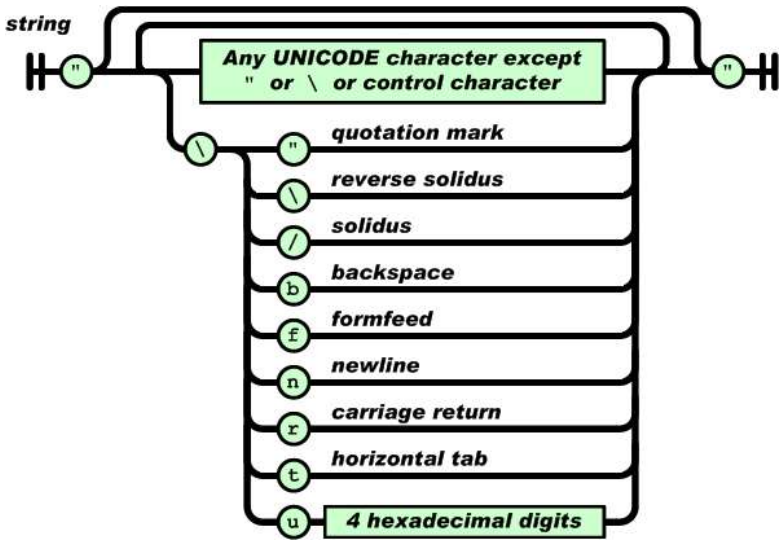
- 2 structures d'éléments : paires « nom, valeur » (non ordonnées) et des listes de valeurs (ordonnées).

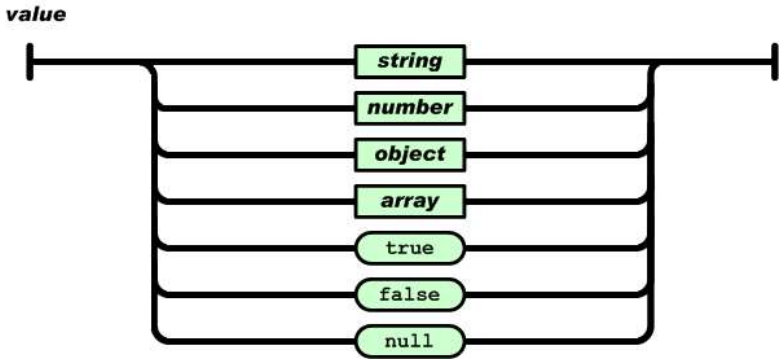
- Les éléments peuvent être des objets, des tableaux, des nombres, des chaînes de caractères, etc.











Les images ainsi que l'exemple :

http://upadhyayjiteshandroid.blogspot.ca/2013/02/android-json-parsing-example.html

Examiner les exemples « **C06 : SimpleJason** » puis « **C06 : AndroidJsonparser** ».

Bibliographies

Building Apps with Connectivity & the Cloud

http://developer.android.com/training/building-connectivity.html

Connectivity

https://developer.android.com/training/connectivity

Building Web Apps in WebView

http://developer.android.com/guide/webapps/webview.html

23.0 HTTP Connection

http://www.bogotobogo.com/Android/android23HTTP.php