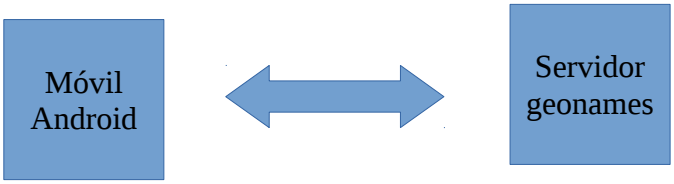
# Desarrollo para dispositivos móviles con Android: Conexión a internet.

## Contenido

# 1 Introducción

Si bien se pueden crear aplicaciones aisladas que realicen tareas útiles, una poderosa capacidad de los dispositivos Android es la de poder conectarse a internet. Para aprovecharla, es necesario solicitar los permisos de acceso a internet y de acceso al estado de la conexión, dado que antes de conectarse a internet lo correcto es comprobar que alguna de las conexiones (Wifi o datos del teléfono) están activas. Una vez hecho esto, es posible manejar fácilmente conexiones HTTP, lo cual permite acceder facilmente a servicios web REST o de todo tipo. De hecho, durante este tema se hace referencia a una aplicación que puede clonarse desde GitHub, y que obtiene periódicamente (cada 10s), la hora desde *geonames.org*.[1]




# 2 Comprobando la conectividad

Es necesario solicitar los permisos INTERNET y ACCESS_NETWORK_STATE, de forma que serán incluidos el el archivo *AndroidManifest.xml*.

```
<manifest
        <!-- más cosas… -->
        <uses-permission android:name="android.permission.INTERNET" />
        <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
        <!-- aún muchas más cosas… -->
</manifest>
```

De otra forma, se produciría una excepción en el momento de la conexión (metodo **HttpURLConnection**.*connect()*), de forma que la aplicación detendría su funcionamiento, o al menos, no sería útil.

Comprobar la conectividad significa que es necesario saber si existe una conexión que ofrezca acceso a internet, no importa si es Wifi o es una conexión de datos de otro tipo. Al menos para aplicaciones básicas no tiene importancia, siempre es posible desgranar el código siguiente para comprobar el tipo de conexión, de ser necesario.

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

NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();

boolean connected = ( networkInfo != null && networkInfo.isConnected() );
```

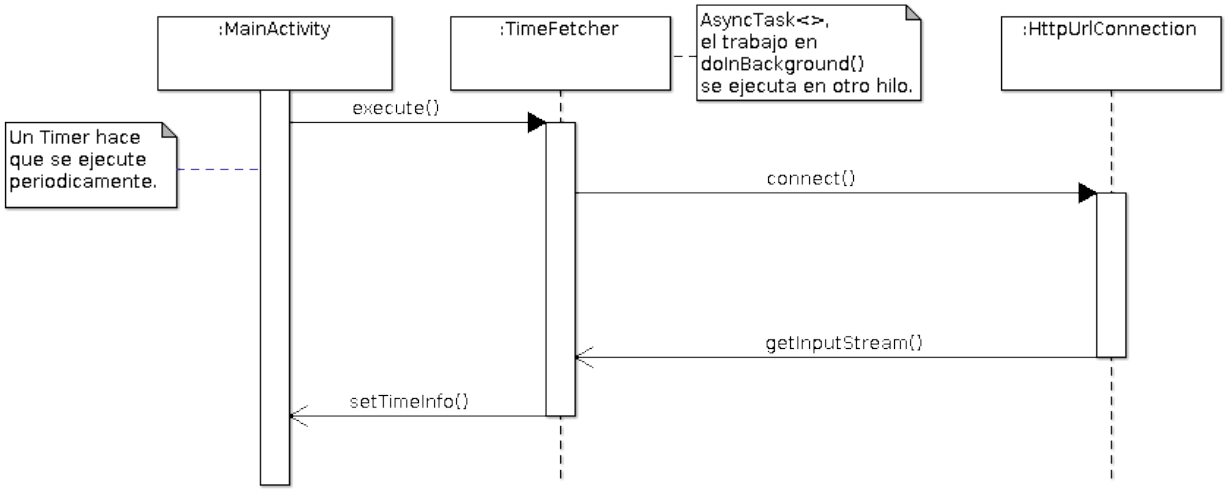
En el código más arriba, se obtiene el servicio de conexiones, y de este se obtiene la conexión de información sobre red. Lo único que deseamos saber es si hay conexión, lo que obtenemos con **NetworkInfo**.*isConnected()*, método que comprueba si alguno de los modos de conexión están activos, y si se tiene acceso a internet con ellos.

---

1El servicio que ofrece *geonames.org* es de tipo gratuito mediante el empleo del usuario “demo”, el cual tiene conexiones diarias y horarias limitadas. Sin embargo, este usuario suele estar colapsado, así que se ha creado el usuario “dispositivos_moviles”.

# 3 Accediendo a internet

El acceso a internet se realiza mediante la clase **HttpUrlConnection**, que se puede obtener fácilmente a partir de un objeto **URL**. El esquema de funcionamiento básico de la aplicación es como sigue.




Un **Timer** hace que la ejecución se repita periódicamente, informando por tanto del tiempo con una exactitud suficiente para un uso casual.

Como se verá en una sección más adelante, no es posible conectarse a internet desde el hilo principal. Android de hecho no lo permite, al lanzar una excepción en cuanto se detecta una conexión a internet desde dicho hilo. Por ello, es necesario crear un objeto **AsyncTask**<> que se encargue del trabajo de la conexión.

## 3.1 El trabajo de conexión detallado

Una vez obtenido el objeto **HttpUrlConnection**, se lanza *connect()*, y se obtiene la respuesta como un **InputStream**.

```
URL url = new URL(
               "http://api.geonames.org/timezoneJSON?"
                      +"lat=42.34&lng=-7.86&username=dispositivos_moviles" );
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.setReadTimeout( 1000 /* milisegundos */ );
conn.setConnectTimeout( 1000 /* milisegundos */ );
conn.setRequestMethod( "GET" );
conn.setDoInput( true );

// Obtener la respuesta del servidor
conn.connect();
int codigoRespuesta = conn.getResponseCode();
InputStream is = conn.getInputStream();
```

En el código más arriba, se establece una conexión a la **URL** deseada, indicando que el método deseado es *GET*. Se obtiene la respuesta con **HttpURLConnection**.*getResponseCode()*, que muchas veces es 200 (*ok*) si se ha obtenido la información con éxito, aunque existen multitud de otros códigos (ver referencias). En general, los códigos 2xx significan éxito, 4xx errores del cliente (problemas con la petición o su sintaxis), y 5xx errores del servidor.

En cualquier caso, la respuesta del servidor se obtendrá a través del método **HttpURLConnection**.*getInputStream()*, y si se desea convertir este código en una cadena simple, entonces será necesario utilizar un código como el de más abajo. No siempre es necesario convertir a cadena: si la respuesta se va a entender como una respuesta JSON, y se desea utiliza la clase **JSONObject**, entonces no quedará otro remedio. En cambio, los parser XML son capaces de tomar un **InputStream** en lugar de una cadena o el nombre de un archivo.

## 3.2 Convirtiendo la respuesta en una cadena

Es bastante probable que sea necesario, en varias situaciones, convertir la respuesta de una solicitud web en una cadena de texto (**String**). El algoritmo solo utiliza un objeto **StringBuilder** para ir añadiendo cada línea leída desde el objeto **InputStream**., mediante el método *readLine()*, que devuelve *null* cuando no se puede leer más.

```
private String getStringFromStream(InputStream is)
    {
        StringBuilder toret = new StringBuilder();
        String line = "";

        try ( BufferedReader reader = new BufferedReader( new InputStreamReader( is ) ) ) {
            while( ( line = reader.readLine() ) != null ) {
                toret.append( line );
            }
        } catch (IOException e) {
            Log.e( LOG_TAG, " in getStringFromString(): error converting net input to string"  );
        }

        return toret.toString();
    }
```

## 3.3 Respuesta XML

El formato XML es un formato de intercambio de datos de sobra conocido: se basa en texto, siendo capaz de proporcionar una estructura jerárquica. Se basa en etiquetas o *tags*, y siempre hay una etiqueta raiz de la que cuelga el resto. En cuanto al caso concreto de *geonames.org*, la respuesta a la URL "*http://api.geonames.org/timezone?lat=42.34&lng=-7.86&username=dispositivos_moviles*" es similar a la siguiente:

```
<geonames>
  <timezone tzversion="tzdata2015a">
    <countryCode>ES</countryCode>
    <countryName>Spain</countryName>
    <lat>42.34</lat>
    <lng>-7.86</lng>
    <timezoneId>Europe/Madrid</timezoneId>
    <dstOffset>2.0</dstOffset>
    <gmtOffset>1.0</gmtOffset>
    <rawOffset>1.0</rawOffset>
    <time>2015-12-06 15:40</time>
    <sunrise>2015-12-06 08:45</sunrise>
    <sunset>2015-12-06 17:58</sunset>
  </timezone>
</geonames>
```

Nótese que, si bien en este ejemplo los datos XML se formatean para que tengan mejor legibilidad, el formato XML no precisa de retornos de carro, pudiendo estar toda la información contenida en una sola línea.

Para Android se recomienda el uso de **XMLPullParser**, aunque para una respuesta tan simple como esta supone un procesamiento relativamente complejo.

## 3.4 Respuesta JSON

### *3.4.1 El formato JSON*

Si bien originalmente la mayoría de las respuestas de servicios web utilizaban XML, JSON se ha establecido muy pronto como una alternativa muy popular, ya que es mucho más simple. Se trata de la representación más simple de un diccionario, de forma que si es necesario que una entrada tenga una estructura más compleja, es posible repetirla recursivamente, utilizando las llaves '{' y '}' como delimitadores. Los retornos de línea y tabuladores son opcionales, y solo se usan para clarificar la estructura.

```
{
	“asignatura”: “Dispostivos Móviles”,
	“profesor”: “Baltasar García”,
	“cuatrimestre”: 1
}
```

Así, cada elemento se separa del siguiente mediante una coma (','), y la etiqueta de su valor mediante dos puntos (':'). Todos los elementos se encierran entre llaves ('{' y '}'). En caso de que un elemento precise a su vez una estructura compleja, esta se encierra entre llaves, indicando los elementos que sean necesarios.

```
{
	“asignatura”: “Dispostivos Móviles”,
	“profesor”: “Baltasar García”,
	“cuatrimestre”: 1,
	“fecha_comienzo”:	{
				dia: 13,
				mes: 9,
				anyo:2016
			}
}
```

## 3.5 La respuesta de geonames.org

En cuanto al caso concreto de *geonames.org*, la respuesta a la URL “*http://api.geonames.org/timezoneJSON?lat=42.34&lng=-7.86&username=dispositivos_moviles*” es la siguiente:

```
{
    "sunrise":"2015-12-06 08:45",
    "lng":-7.86,
    "countryCode":"ES",
    "gmtOffset":1,
    "rawOffset":1,
    "sunset":"2015-12-06 17:58",
    "timezoneId":"Europe/Madrid",
    "dstOffset":2,
    "countryName":"Spain",
    "time":"2015-12-06 15:39",
    "lat":42.34
}
```

Como en el caso de XML, se ha formateado de manera que resulte más legible, pero puede ser perfectamente servido en una sola línea de texto. El siguiente código demuestra cómo obtener los datos desde el formato *JSON* mediante **JSONObject**.

```
JSONObject json = new JSONObject( getStringFromStream( is ) );
this.time = json.getString( “time” );
this.timeInfo = json.getString( “timezoneId” )
                     + " (" + json.getString( “countryName” ) + ")";
int gmtOffset = json.getInt( "gmtOffset" );
```

# 4 El problema de realizar el trabajo en el hilo principal

Las aplicaciones de Android tienen un hilo (*thread*) principal, el hilo en el que se muestra y se maneja la interfaz de usuario. Así, se trata del hilo que se encarga de realizar todo el trabajo relacionado con manejar menús, visualizar botones… si este hilo se sobrecarga, entonces el usuario no podrá interactuar con la aplicación, apareciendo esta como bloqueada o que no responde.

De hecho, Android no permite que se realice el trabajo de acceso a internet en el hilo principal. En el caso de acceso a base de datos aún lo tolera, pero no en cuanto al manejo de servicios web.

La forma correcta es forzar a que dicho trabajo se produzca en un hilo al margen. Y para ello, se utiliza la clase genérica **AsyncTask**<>. La pregunta subyacente es: ¿cuándo es necesario utilizar **AsyncTask**<>? La respuesta es que siempre que se realicen tareas de manejo de red o de bases de datos. Y en el primer caso, ni siquiera hay otro remedio.

Un segundo problema derivado del uso de **AsyncTask<>,** consiste en que los controles gráficos (*widgets*) solo pueden ser manejados por el hilo principal. Así, será necesario diseñar cuidadosamente la tarea en un hilo aparte para que realice única y exclusivamente dicha tarea.

Lo más cómodo es derivar nuestra clase de la propia clase **AsyncTask<>**, de forma que deben sobrecargarse obligatoriamente el método *doInBackground()*. Este método es, específicamente, el método que se ejecuta en un hilo al margen. También es posible sobrecargar los métodos *onPreExecute()* y *onPostExecute()*, pero esos se ejecutan siempre en el hilo principal. También es posible que sea necesario sobrecargar el método *onProgressUpdate()*, que permite actualizar una barra de progreso o una etiqueta con un porcentaje con cada cambio en la tarea.

Dado que es una clase genérica, es necesario definir en concreto tres tipos dentro de la misma. Se trata, respectivamente, de: *params*, *progress* y *result*. Es decir, el primer tipo indica qué tipo de parámetros requiere la tarea, el segundo el tipo de progreso que se va a publicar (si se publica), y el resultado. Por ejemplo, en el siguiente código se establece que se va a llamar a servicios web, que el progreso se publicará mediante un número real, y que el resultado será un entero con el número de servicios que han respondido.

```
class Downloader extends AsyncTask<URL, Double, Integer> {
	// …
	@Override
	public Integer doInBackground(URL… url) {
		// …
	}

	@Override
	public void onPostExecute(Integer result) {
		// ….
	}
}
```

Nótese que es necesario utilizar las clases asociadas a los tipos básicos en el caso de que estos se empleen (como en el ejemplo).

El caso más simple sería indicar que no se va a visualizar ningún progreso (en este caso se utiliza la clase **Void**), y que el resultado de la tarea será un *true* si se ha llevado a cabo y *false* en otro caso.

```
class Downloader extends AsyncTask<URL, Void, Boolean> {
	// …
	@Override
	public Boolean doInBackground(URL… url) {
		// …
	}

	@Override
	public void onPostExecute(Boolean result) {
		// ….
	}
}
```

Así, URL será el tipo de los parámetros de *doInBackground()*, mientras que **Boolean** será el tipo del resultado que se pasará a *onPostExecute()*. La sintaxis URL… denota que es posible pasar varios objetos del tipo URL, aceptando cualquier número de ellos. En este caso, *url* se comporta como un vector, pudiendo acceder a un elemento en concreto mediate el operador [], u obtener el número de url's accediendo a *length*.

Así, todo el trabajo que se realice en doInBackground() se hará en un hilo al margen del principal. Por diferentes razones (como que la aplicación termine), es posible que el hilo se cancele. Por eso es importante comprobar el estado devuelto por el método *isCancel()*, por ejemplo si se está accediendo a varias webs, para saber si la tarea ha sido cancelada, y terminarla en ese caso.

Finalmente, no sabremos cuándo termina el hilo la tarea encomendada. Una de las soluciones a este problema consiste en pasarle a esta clase la actividad en el hilo principal, de forma que pueda ser accedida desde *onPostExecute()*. Nótese que esta se ejecuta ya en el hilo principal, y no en el nuevo hilo, como en *doInBackground()*.

El mismo ejemplo relacionado con el acceso a *geonames.org* es el siguiente:

```
public class TimeFetcher extends AsyncTask<URL, Void, Boolean> {
	private Main activity;
	private String time;
	private String timeInfo;
	private String gmtInfo;

	public TimeFetcher(Main activity) {
		this.activity = activity;
	}

	@Override
	protected void onPreExecute() {
		final TextView lblStatus = (TextView) this.activity.findViewById( R.id.lblStatus );
		this.activity.setStatus( R.string.status_connecting );
		this.setDefaultValues();
	}
	@Override
	protected Boolean doInBackground(URL... urls) {
		InputStream is = null;
		boolean toret = false;

		try {
			// Check connectivity
			ConnectivityManager connMgr = (ConnectivityManager)
			    this.activity.getSystemService( Context.CONNECTIVITY_SERVICE);
			NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
			boolean connected = ( networkInfo != null && networkInfo.isConnected() );

			if ( !connected ) {
				this.activity.setStatus( R.string.status_not_connected );
			} else {
				// Connection
				HttpURLConnection conn =
						(HttpURLConnection) urls[ 0 ].openConnection();
				conn.setReadTimeout( 1000 /* milliseconds */ );
				conn.setConnectTimeout( 1000 /* milliseconds */ );
				conn.setRequestMethod( "GET" );
				conn.setDoInput( true );

				// Obtain the answer
				conn.connect();
				is = conn.getInputStream();

				// Starts the query
				JSONObject json = new JSONObject( getStringFromStream( is ) );
				this.time = json.getString( “time” );
				this.timeInfo = json.getString( “timezoneId” ) + " ("
					+ json.getString( “countryName” ) + ")";
				toret = true;
			}
		}
		catch( //...
		}

		return toret;
	}

	@Override
	public void onPostExecute(Boolean result)
	{
		final TextView lblStatus = (TextView) this.activity.findViewById( R.id.lblStatus );
		int idFinalStatus = R.string.status_ok;

		if ( !result ) {
			idFinalStatus = R.string.status_error;
		}

		lblStatus.setText( idFinalStatus );
		this.activity.setTimeInfo( time, timeInfo, gmtInfo );
	}
}
```

# 5 Referencias


- Documentación y recursos de Android para desarrolladores (accedido en sept. 2015) http://developer.android.com/
- Comprobación de conectividad http://developer.android.com/intl/es/training/monitoring-device-state/connectivity-monitoring.html
- Conexión a internet http://developer.android.com/intl/es/training/basics/network-ops/connecting.html
- Códigos de retorno de conexiones HTTP: https://es.wikipedia.org/wiki/Anexo:Códigos_de_estado_HTTP
- Procesado de XML: http://developer.android.com/intl/es/training/basics/network-ops/xml.html
- Procesado de JSON: http://developer.android.com/intl/es/reference/org/json/JSONObject.html
- Utilización de Volley especificamente para JSON: http://developer.android.com/intl/es/training/volley/request.html#request-json
- AsyncTask: http://developer.android.com/intl/es/reference/android/os/AsyncTask.html
- Ejemplo completo: https://github.com/Baltasarq/AndroidCurrentTime