

# ¿Qué ofrece Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.  
 Ese apoyo que siempre quiso tener...

## 1. Desarrollo de componentes y proyectos a medida



## 2. Auditoría de código y recomendaciones de mejora

## 3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



## 4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces,  
 HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)  
 Gestor de contenidos (Alfresco)  
 Aplicaciones híbridas

Tareas programadas (Quartz)  
 Gestor documental (Alfresco)  
 Inversión de control (Spring)

Control de autenticación y  
 acceso (Spring Security)  
 UDDI  
 Web Services  
 Rest Services  
 Social SSO  
 SSO (Cas)

JPA-Hibernate, MyBatis  
 Motor de búsqueda empresarial (Solr)  
 ETL (Talend)

Dirección de Proyectos Informáticos.  
 Metodologías ágiles  
 Patrones de diseño  
 TDD

BPM (jBPM o Bonita)  
 Generación de informes (JasperReport)  
 ESB (Open ESB)

AdictosAlTrabajo

Temporada Completa de Terrakas terrakas.com



autentia Soporte a desarrollo informático Hosting patrocinado por enREDados

Entra en Adictos a través de

E-mail

Contraseña

[Deseo registrarme](#)  
[Olvidé mi contraseña](#)

[Inicio](#) [Quiénes somos](#) [Formación](#) [Comparador de salarios](#) [Nuestros libros](#) [Más](#)

» Estás en: [Inicio](#) [Tutoriales](#) [Cómo añadir Volley \(librería de Android\) en Android Studio](#)



Francisco J. Arroyo

Consultor tecnológico de desarrollo de proyectos informáticos.

Puedes encontrarme en Autentia: Ofrecemos servicios de soporte a desarrollo, factoría y formación

Somos expertos en Java/JEE

[Ver todos los tutoriales del autor](#)



Fecha de publicación del tutorial: 2014-03-11

Tutorial visitado 1 veces [Descargar en PDF](#)

## Cómo añadir Volley(librería de Android) en Android Studio

### 0. Índice de contenidos.

- 1. Introducción.
- 2. Entorno.
- 3. Manos a la obra.
  - 3.1 Descarga de Volley.
  - 3.2 Con Apache Ant.
    - 3.2.1 Importando el proyecto.
    - 3.2.2 Generación de jar con código fuente.
  - 3.3 Con Gradle.
    - 3.3.1 Generación de paquete con código fuente.
- 4. Conclusiones

### 1. Introducción

Volley es una librería para realizar peticiones http que fue presentada en el pasado Google I/O 2013.

Si hemos necesitado realizar conexiones http en alguna de las aplicaciones Android que hayamos desarrollado, seguramente hayamos usado **URLConnection** o **httpClient** o incluso los más aventurados quizás hayan usado **Ion** de **Koush** o la implementación de Android de **RestTemplate** de la gente de Spring

El problema de usar alguna de las dos primeras es que tienes que tener en cuenta la versión de Android en la que se está ejecutando tu aplicación porque las implementaciones de ambas tienen bastantes bugs. Por debajo de la 2.2 deberías usar **httpClient** y por encima **URLConnection**.

El "problema" de usar las dos últimas es que no son oficiales. Habrá quien tenga en cuenta esto y habrá quien no. También habrá quien diga que este tutorial va de importar una librería que no se distribuye aún con el **SDK de Android**

Sabiendo un poco en el contexto que nos movemos os describo brevemente las principales características de Volley.

- Realiza las peticiones de forma asíncrona. Ya no más **NetworkOnMainThreadException** ;P
- Gestiona una cola de descargas con prioridad. También podemos cancelar las peticiones
- Caché de peticiones. También nos permite usar nuestra propia caché implementando una interfaz.
- Toolbox de las peticiones más comunes: Json, etc... También para establecer imágenes de internet en nuestros **ImageView**...
- Tiene en cuenta la versión de Android para usar **URLConnection** y **httpClient** de forma transparente para nosotros.
- Previsiblemente acabará incluyéndose en el **SDK** o en alguna de las librerías de soporte de Google.

Como véis es una librería muy completa, aunque el ámbito del tutorial sólo va a llegar a importar el módulo :P.

Por si queréis echar un vistazo a la presentación de la librería en el Google IO, podéis verla [aquí](#) o en el canal de Youtube de Google Developers

### 2. Entorno

- Hardware
  - MacBook Pro
    - Intel Core i7 2Ghz

### Catálogo de servicios Autentia



### Síguenos a través de:



### Últimas Noticias

» [Buscamos personal para Autentia y nuestros clientes \(10-Marzo-2014\)](#)

» [Charla de Auto Layout en nuestra oficina](#)

» [PhoneGap y Apache Cordova: resolviendo el enredo.](#)

» [Mi semana de desk-surfing en Otagami](#)

» [Enamórate de un geek](#)

[Histórico de noticias](#)

### Últimos Tutoriales

» [Kettle no es una tetera, es la herramienta de ETL de Pentaho!](#)

» [Primeros pasos de MapReduce con Hadoop](#)

» [Primeros pasos con Hadoop: instalación y configuración en Linux](#)

» [Como configurar CloudFlare en nuestra web](#)

Page Pushers Community [Help?](#)

0 people brought clicks to this page

no clicks + + + + + + + +

powered by [karmacracy](#)

- 8GB RAM
- 250GB SSD
- Sistema Operativo: Mac OS X (10.9)
- Software
  - Android Studio 0.4.2
  - Git
  - Volley
  - Apache Ant 1.9.2
  - Gradle 1.10

» Crea todo un entorno de máquinas virtuales con un solo comando, gracias a Vagrant

### Últimos Tutoriales del Autor

- » Integrar Barcode Scanner en nuestra aplicación Android
- » Jugando con JSON en Java y la librería GSON. Parte 2
- » Android Beam
- » Como hacer nuestros test más legibles con Hamcrest
- » Ejemplo de Viewpager para android

### 3. Manos a la obra.

Volley viene preparado para compilar el proyecto con **Ant** y en las últimas versiones también con **Gradle**.

#### 3.1 Descarga del proyecto.

Como hemos dicho en la introducción, Google aún no distribuye la librería a través del sdk de Android, por lo que lo tendremos que descargar por nuestra cuenta. Para ello, vamos a usar git para clonar el repositorio de **Volley** y tener una copia del código fuente en nuestro equipo.

Desde un terminal con git instalado hacemos: `git clone https://android.gcodesource.com/platform/frameworks/volley`

```

fjarroyo:volley-tuto fjarroyo$ git clone https://android.gcodesource.com/platform/frameworks/volley
Cloning into 'volley'...
remote: Counting objects: 21, done
remote: Finding sources: 100% (21/21)
remote: Total 2081 (delta 115), reused 2081 (delta 115)
Receiving objects: 100% (2081/2081), 860.37 KiB | 0 bytes/s, done.
Resolving deltas: 100% (115/115), done.
Checking connectivity... done
fjarroyo:volley-tuto fjarroyo$
    
```

Cuando termine tendremos en la carpeta que hayamos hecho el clone una nueva carpeta con el nombre volley y con el código fuente necesario para que podamos contruir la librería.

#### 3.2 Con Apache Ant.

Lo que vamos a hacer con Ant es generar un jar que luego podremos añadir como dependencia en el proyecto Android en el que necesitemos usarlo.

Vamos a ver el contenido de proyecto:

```

fjarroyo:volley fjarroyo$ ls
Android.mk          custom_rules.xml   src
AndroidManifest.xml proguard-project.txt tests
build.gradle        proguard.cfg
build.xml           project.properties
fjarroyo:volley fjarroyo$
    
```

Ejecutamos `android update project -p` . (el punto al final también hay que añadirlo) para modificar el proyecto con los datos locales: por ejemplo donde se encuentra nuestra instalación del sdk.

```

volley — cdl@app22:/datos2/apps/scripts/abaco — bash — 76x18
fjarroyo:volley fjarroyo$ /Applications/Development/android-sdk-macosx/tools
/android update project -p .
Updated local.properties
Updated file ./proguard-project.txt
It seems that there are sub-projects. If you want to update them
please use the --subprojects parameter.
fjarroyo:volley fjarroyo$ cat local.properties
# This file is automatically generated by Android Tools.
# Do not modify this file -- YOUR CHANGES WILL BE ERASED!
#
# This file must *NOT* be checked into Version Control Systems,
# as it contains information specific to your local configuration.
#
# location of the SDK. This is only used by Ant
# For customization when using a Version Control System, please read the
# header note.
sdk.dir=/Applications/Development/android-sdk-macosx
fjarroyo:volley fjarroyo$

```

Y una vez que tenemos el proyecto listo, lo compilamos con Ant para generar un jar. Ejecutamos **ant jar**.

```

volley — cdl@app22:/datos2/apps/scripts/abaco — bash — 76x18
fjarroyo:volley fjarroyo$ /Applications/Development/apache-ant-1.9.2/bin/ant
jar
Buildfile: /Users/fjarroyo/Documents/workspaces/volley-android/volley-tuto/v
olley/build.xml

-pre-build:

-check-env:
[checkenv] Android SDK Tools Revision 22.3.0
[checkenv] Installed at /Applications/Development/android-sdk-macosx

-setup:
[echo] Project Name: volley

[aapt] Generating resource IDs...
[echo] -----
[echo] Handling BuildConfig class...
[buildconfig] Generating BuildConfig class.

-pre-compile:

-compile:
[javac] Compiling 42 source files to /Users/fjarroyo/Documents/workspace
s/volley-android/volley-tuto/volley/bin/classes

jar:
[jar] Building jar: /Users/fjarroyo/Documents/workspaces/volley-androi
d/volley-tuto/volley/bin/volley.jar

BUILD SUCCESSFUL
Total time: 2 seconds
fjarroyo:volley fjarroyo$

```

Si todo ha ido bien y tenemos un bonito **BUILD SUCCESSFUL** entonces tendremos un jar llamado volley.jar en el directorio bin que podremos usar para importar en Android Studio.

```

bin — cdl@app22:/datos2/apps/scripts/abaco — bash — 76x18
fjarroyo:volley fjarroyo$ cd bin/
fjarroyo:bin fjarroyo$ ls
AndroidManifest.xml  dexedLibs          res
AndroidManifest.xml.d  jarlist.cache     volley.jar
classes              proguard.txt
fjarroyo:bin fjarroyo$

```

### 3.2.1 Importando el proyecto.

Para usar la librería creamos una carpeta lib y copiamos el jar que acabamos de generar en esa carpeta.

Las últimas versiones de Android Studio vienen ya configuradas para que use esa carpeta como dependencias (como ocurría con Eclipse). Si no es nuestro caso, añadimos la dependencia nosotros a mano o a través del asistente de Android Studio. Nos tiene que quedar el fichero **build.gradle** así:

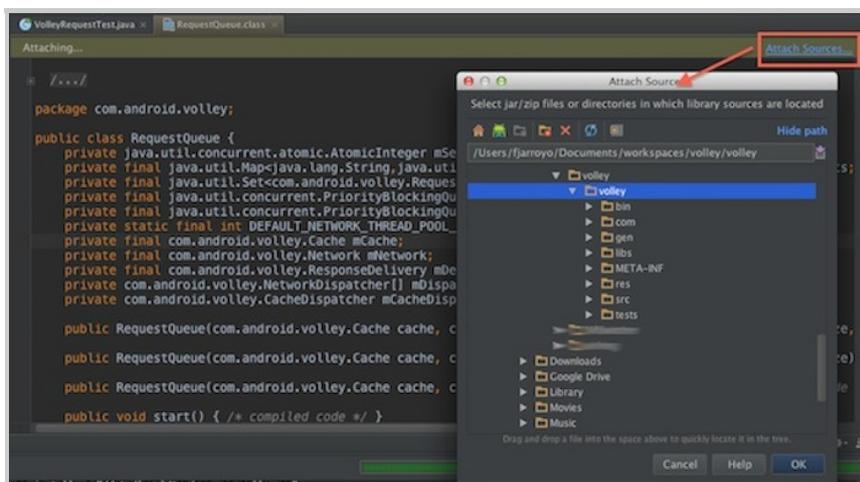
```
view plain print ?
01. dependencies {
02.     compile 'com.android.support:appcompat-v7:+'
03.
04.     compile files('libs/volley.jar')
05. }
```

Como ya podemos usar **Volley** vamos hacer un test que lo corrobore :P

```
view plain print ?
01. @LargeTest
02. public class VolleyRequestTest extends AndroidTestCase{
03.
04.     public void testVolleyWithSources() throws InterruptedException {
05.         //Como la llamada es asincrona, vamos a usar CountdownLatch para esperar a que term
06.         final CountdownLatch latch = new CountdownLatch(1);
07.         //Workaround para asignar el valor de la respuesta dentro de un objeto
08.         //anónimo y poder comprobar el resultado una vez que el callback a terminado.
09.         final String[] responseBody = new String[1];
10.
11.         final RequestQueue requestQueue = Volley.newRequestQueue(getContext());
12.         requestQueue.start();
13.         StringRequest request = new StringRequest(
14.             Request.Method.GET,
15.             "http://www.autentia.com",
16.             new Response.Listener<String>() {
17.                 @Override
18.                 public void onResponse(String response) {
19.                     responseBody[0] = response;
20.                     latch.countDown();
21.                 }
22.             },
23.             new Response.ErrorListener() {
24.                 @Override
25.                 public void onErrorResponse(VolleyError error) {
26.                     latch.countDown();
27.                 }
28.             }
29.         );
30.         requestQueue.add(request);
31.
32.         //Bloqueamos el hilo hasta que el callback llame a latch.countDown
33.         latch.await();
34.
35.         //Comprobamos la petición.
36.         assertNotNull(responseBody[0]);
37.         assertTrue(responseBody[0].contains("autentia"));
38.     }
39. }
```

### 3.2.2 Generacion de jar con código fuente.

En muchas ocasiones nos interesa poder ver el código de las clases que estamos usando incluso cuando estamos esa librería no es nuestra. Hay ocasiones en las que el entorno es capaz de acceder al código fuente sin que nosotros tengamos que hacer nada (descargándolo de internet...). Hay otras ocasiones en las que el entorno (en este caso Android Studio) nos permite seleccionar dónde se encuentra el código fuente para que luego podamos verlo desde el entorno.



Si somos nosotros los que hemos generado el jar de Volley, tendremos el código fuente y podremos adjuntarlo desde el directorio donde hayamos hecho el clone, pero si vamos a compartir el código (por ejemplo en Bitbucket o Github) quizás nos interese generar un jar que también lleve los fuentes.

Lo que vamos a hacer es añadir una tarea de ant en medio del proceso de generación del jar de **Android** cuando este se construye por medio de Ant. Con la liberación de Android Studio, se está liberando un plugin de Android para Gradle de tal forma que se va a dejar de usar Ant para construir los proyectos a favor de Gradle, pero de momento, podemos usar Ant con Volley.

Cada proyecto tiene un fichero build.xml con información sobre como construir la aplicación. Este fichero tiene una referencia a un fichero build.xml que se encuentra en el directorio **tools/ant** de vuestro sdk de Android y es en este último fichero donde se encuentran las tareas de Ant comunes a la construcción de una aplicación o librería de Android.

Si miramos ese fichero nos encontraremos una línea como la siguiente

```
view plain print ?
01. <!-- Os pongo lo que nos interesa a nosotros -->
02.
03. <!-- empty default pre-compile target. Create a similar target in
04.      your build.xml and it'll be called instead of this one. -->
05. <target name="-pre-compile"/>
06.
07. <target name="-compile" depends="-pre-build, -build-setup, -code-gen, -pre-compile">
08.   <!-- Se omite el resto -->
09. </target>
10.
11. <!-- Se omite el resto -->
```

Como podéis ver, cuando se llama a la fase de compilación, esta depende de una fase llamada **-pre-compile** que por defecto está vacía, pero que han añadido para que nosotros podamos personalizarla y añadir nuestras necesidades al proceso de construcción de la aplicación.

Lo que vamos a hacer es crear una tarea en la que copiamos los fuentes de la aplicación en el jar resultante.

- Creamos un fichero build-with-sources.xml en el directorio donde hayamos clonado el repositorio de Volley
- Añadimos nuestro build-with-sources.xml con la tarea para copiar los fuentes
- Volvemos a llamar a ant jar para que nos vuelva a generar el jar que ya incluirá los fuentes.

```
Franciscos-MacBook-Pro-4:volley fjarroyo$ ls
Android.mk          build.xml          proguard-project.txt
AndroidManifest.xml  com               proguard.cfg
META-INF           custom_rules.xml  project.properties
bin                gen               res
build-withsources.xml  libs              src
build.gradle        local.properties  tests
Franciscos-MacBook-Pro-4:volley fjarroyo$
```

El contenido de build-with-sources.xml es el siguiente:

```
view plain print ?
01. <?xml version="1.0" encoding="UTF-8"?>
02. <project name="buil-with-sources" default="help">
03.
04. <!-- Importamos el build.xml del proyecto para tener disponible todas las tareas -->
05. <import file="build.xml" />
06.
07. <!-- Definimos nuestra propia tarea que lo que va a hacer es copiar los .java al jar -->
08. <target name="-pre-compile">
09.   <echo message="Copying sources" />
10.   <copy todir="${out.classes.absolute.dir}">
11.     <fileset dir="${source.absolute.dir}" />
12.   </copy>
13. </target>
14.
15. </project>
```

Una vez que tengamos el fichero en el directorio de Volley, volvemos a llamar a **ant jar** especificando el nuevo fichero de build.

```
view plain print ?
01. ant jar -f build-withsources.xml
```

Cuanto termina de compilar, veréis que hay un mensaje en la fase de **-pre-compile**

```

[echo] -----
[echo] Handling BuildConfig class...
[buildconfig] Generating BuildConfig class.

-pre-compile:
[echo] Copying sources

-compile:
[javac] Compiling 1 source file to /Users/fjarroyo/Documents/workspaces/volley/volley/bin/classes

jar:
[jar] Building jar: /Users/fjarroyo/Documents/workspaces/volley/volley/bin/volley.jar

BUILD SUCCESSFUL
Total time: 1 second
Franciscos-MacBook-Pro-4:volley fjarroyo$

```

Para comprobar que se ha generado bien, hacemos `jar -tf bin/volley.jar`. Veréis que hay archivos .java en el jar.

```

com/android/volley/toolbox/JsonObjectRequest.java
com/android/volley/toolbox/JsonRequest.class
com/android/volley/toolbox/JsonRequest.java
com/android/volley/toolbox/NetworkImageView$1.class
com/android/volley/toolbox/NetworkImageView$1.class
com/android/volley/toolbox/NetworkImageView.class
com/android/volley/toolbox/NetworkImageView.java
com/android/volley/toolbox/NoCache.class
com/android/volley/toolbox/NoCache.java
com/android/volley/toolbox/PoolingByteArrayOutputStream.class
com/android/volley/toolbox/PoolingByteArrayOutputStream.java
com/android/volley/toolbox/RequestFuture.class
com/android/volley/toolbox/RequestFuture.java
com/android/volley/toolbox/StringRequest.class
com/android/volley/toolbox/StringRequest.java
com/android/volley/toolbox/Volley.class
com/android/volley/toolbox/Volley.java
Franciscos-MacBook-Pro-4:volley fjarroyo$

```

Ahora solo tenemos que añadir volley.jar a nuestro proyecto Android como hemos hecho antes o reemplazar el jar si ya lo habíamos añadido.

```

/**
 * A request dispatch queue with a thread pool of dispatchers.
 *
 * Calling {@link #add(Request)} will enqueue the given Request for dispatch,
 * resolving from either cache or network on a worker thread, and then delivering
 * a parsed response on the main thread.
 */
@RawTypes
public class RequestQueue {

    /** Used for generating monotonically-increasing sequence numbers for requests. */
    private AtomicInteger mSequenceGenerator = new AtomicInteger();

    /**
     * Staging area for requests that already have a duplicate request in flight.
     *
     * <ul>
     * <li>containsKey(cacheKey) indicates that there is a request in flight for the given cache
     * key.</li>
     * <li>get(cacheKey) returns waiting requests for the given cache key. The in flight request
     * is <em>not</em> contained in that list. Is null if no requests are staged.</li>
     * </ul>
     */
    private final Map<String, Queue<Request>> mWaitingRequests =
        new HashMap<>();
}

```

### 3.3 Con Gradle.

Ahora vamos a cambiar un poco el enfoque. Vamos a usar Gradle, para generar la librería e instalarla en un repositorio local que luego usaremos para importar la librería en nuestro proyecto. Al final del tutorial, en las conclusiones comentaré qué ventajas e inconvenientes veo a cada método.

Como vamos a usar gradle, tenemos que fijarnos en el fichero build.gradle. Primero nos tenemos que asegurar que existan estas líneas en el fichero:

```

view plain print ?
01. buildscript {
02.     repositories {
03.         mavenCentral()
04.     }
05.     dependencies {
06.         classpath 'com.android.tools.build:gradle:0.8.+'
07.     }
08. }

```

Comento esto porque si estamos usando una versión antigua del repositorio de Volley, no existirán y tendréis que añadir las a mano para poder construir la librería. Lo que significan es que están indicando a Gradle qué versión del plugin de Android para Gradle usar y dónde puede encontrarlo. Así que cuando más adelante en el fichero se encuentre esta línea: **apply plugin: 'android-library'**, gradle sepa que hacer.

Vamos a ver que tareas nos permite ejecutar el plugin de Android. Ejecutamos gradle tasks en un terminal y obtenemos algo como lo siguiente:

```
volley-gradle -- bash -- 74x63

Build tasks
-----
assemble - Assembles all variants of all applications and secondary packages.
assembleDebug - Assembles all Debug builds
assembleDebugTest - Assembles the Test build for the Debug build
assembleRelease - Assembles all Release builds
build - Assembles and tests this project.
buildDependents - Assembles and tests this project and all projects that depend on it.
buildNeeded - Assembles and tests this project and all projects it depends on.
clean - Deletes the build directory.

Build Setup tasks
-----
init - Initializes a new Gradle build. [incubating]
wrapper - Generates Gradle wrapper files. [incubating]

Help tasks
-----
dependencies - Displays all dependencies declared in root project 'volley-gradle'.
dependencyInsight - Displays the insight into a specific dependency in root project 'volley-gradle'.
help - Displays a help message
projects - Displays the sub-projects of root project 'volley-gradle'.
properties - Displays the properties of root project 'volley-gradle'.
tasks - Displays the tasks runnable from root project 'volley-gradle'.

Install tasks
-----
installDebugTest - Installs the Test build for the Debug build
uninstallAll - Uninstall all applications.
uninstallDebugTest - Uninstalls the Test build for the Debug build

Verification tasks
-----
check - Runs all checks.
connectedCheck - Runs all device checks on currently connected devices.
connectedInstrumentTest - Installs and runs the tests for Build 'debug' on connected devices.
deviceCheck - Runs all device checks using Device Providers and Test Servers.
lint - Runs lint on all variants.
lintDebug - Runs lint on the Debug build
lintRelease - Runs lint on the Release build

Rules
-----
Pattern: build<ConfigurationName>: Assembles the artifacts of a configuration.
Pattern: upload<ConfigurationName>: Assembles and uploads the artifacts belonging to a configuration.
Pattern: clean<TaskName>: Cleans the output files of a task.

To see all tasks and more detail, run with --all.

BUILD SUCCESSFUL

Total time: 4.603 secs
Franciscos-MacBook-Pro-4:volley-gradle fjarroyo$
```

Ejecutamos gradle clean build para generar una versión de Release y cuando termine deberíamos tener un mensaje de "BUILD SUCCESSFUL". Una vez terminado podemos ver que Gradle nos ha generado una carpeta **Build** en el directorio del proyecto con los recursos y las clases que ha necesitado para construir la librería. Si miramos dentro de esa carpeta veremos otra carpeta con nombre "libs" en donde se encuentra la librería generada.

```
Franciscos-MacBook-Pro-4: libs fjarroyo$ pwd
/Users/fjarroyo/Documents/workspaces/volley/volley-gradle/build/libs
Franciscos-MacBook-Pro-4: libs fjarroyo$ ls
volley-gradle-debug.aar volley-gradle.aar
Franciscos-MacBook-Pro-4: libs fjarroyo$
```

Ahora bien, podríamos añadir la librería a nuestro proyecto igual que hemos hecho antes con la versión jar, pero vamos a cambiar el enfoque. Lo que vamos a hacer es añadir el plugin de **Maven** para gradle en el fichero de configuración del proyecto de tal forma que podamos instalar la librería en un repositorio de artefactos. En mi caso vamos a desplegar la librería en un Apache Archiva montado en otro equipo para hacer la prueba aunque también os voy a poner cómo instalarla en el repositorio local de maven.

Para evitar tocar lo máximo posible el fichero build.gradle de Volley y evitar posibles conflictos cuando actualicemos el proyecto, crearemos otro archivo que referenciaremos desde build.gradle.

Vamos a crear un archivo con nombre **install.gradle** con la configuración necesaria para desplegar la librería.

```
view plain print ?
01. apply plugin: 'maven'
02.
03. uploadArchives {
04.     repositories {
05.         mavenDeployer {
06.             //Donde queremos desplegar el artefacto
07.             repository(url: "http://192.168.0.109:8080/repository/snapshots") {
08.                 authentication(userName: "www.autentia.com", password: "www.adictosaltrabajo.com")
09.             }
10.         }
11.         //Evitamos que el plugin añada un timestamp en la generación del fichero
12.         uniqueVersion = false;
13.
14.         //Información del artefacto. Son por estas propiedades por las que luego añadire
15.         pom.groupId = "com.adictosaltrabajo"
16.         pom.artifactId = "volley"
17.         pom.version = "0.1-SNAPSHOT"
18.     }
19. }
20. }
```

Si en vez de un repositorio remoto, quisiésemos instalarla en un repositorio local solo tenemos que cambiar la dirección de la propiedad **repository**. En este caso no necesitamos credenciales.

```
view plain print ?
01. //En este caso estaríamos usando /Users/fjarroyo
    .m2/repository (directorio por defecto de maven)
02. repository(url: "file://${System.env.HOME}/.m2/repository")
```

Antes de ejecutar la tarea, añadimos la referencia a nuestro script en el fichero build.gradle. Añadimos en el fichero la línea **apply from: 'install.gradle'**

```
view plain print ?
01. //comienzo del script del fichero build.gradle del proyecto Volley
02. apply plugin: 'android-library'
03. //fichero que acabamos de crear
04. apply from: 'install.gradle'
05. //el script continua
```

El siguiente paso es ejecutar la tarea: Ejecutamos **gradle uploadArchives** y veremos como al final de la ejecución, gradle nos muestra una traza de la tarea.

```

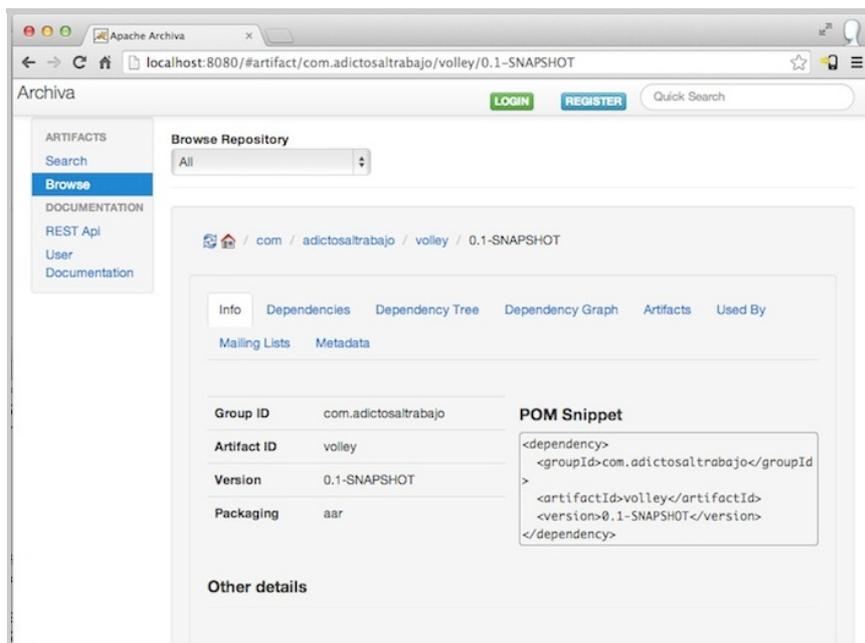
volley-gradle — bash — 74x18
:compileReleaseJava
:processReleaseJavaRes UP-TO-DATE
:packageReleaseJar
:compileReleaseNdk
:packageReleaseJniLibs UP-TO-DATE
:packageReleaseLocalJar UP-TO-DATE
:packageReleaseRenderscript UP-TO-DATE
:bundleRelease
:uploadArchives
Uploading: com/adictosaltrabajo/volley/0.1-SNAPSHOT/volley-0.1-SNAPSHOT.aar to repository remote at http://192.168.0.109:8080/repository/snapshots
Transferring 78K from remote
Uploaded 78K

BUILD SUCCESSFUL

Total time: 8.181 secs
Franciscos-MacBook-Pro-4:volley-gradle fjarroyo$

```

Y si navegamos en nuestro Archiva también seremos capaces de verlo:



Bueno, pues ya que por fin tenemos lista la librería, vamos a ver cómo tendríamos que configurar nuestro proyecto de Android para usar la librería.

Lo primero que tenemos que hacer es configurar el repositorio donde se encuentra la librería

```

view plain print ?
01. repositories {
02.     //Si hemos desplegado la librería en un repositorio remoto, tenemos que decirle a gradle
03.     //En mi caso, el acceso al repositorio es público dentro de mi red local, por lo que no
04.     maven {
05.         url "http://192.168.0.109:8080/repository/snapshots"
06.     }
07.
08.     //Si la hemos desplegado en local, gradle nos proporciona un método para facilitarnos
09.     mavenLocal()
10. }

```

Y para usar la librería, añadimos la dependencia.

```

view plain print ?
01. dependencies {
02.     //otras dependencias..
03.
04.     compile 'com.adictosaltrabajo:volley:0.1-SNAPSHOT@aar'
05.     //compile groupId:artifactId:version@tipo
06. }

```

NOTA, cada vez que modifiquemos el fichero build.gradle desde Android Studio, nos tenemos que acordar de darle al botón para sincronizar el proyecto con los ficheros de gradle.



Para probar que hemos añadido bien la dependencia, podemos probar el mismo test que hay en la parte de integración con Apache Ant.

### 3.3.1 Generación de paquete con código fuente.

Y ahora, igual que con ant, vamos a generar un jar con los fuentes, pero a diferencia de ant, vamos a generar un paquete con la librería y otro con los fuentes.

- Crearemos una nueva tarea **sourcesJar**.
- Modificamos la tarea **uploadArchives** para indicarle que coga el jar resultante de **sourcesJar**.
- Indicamos que **uploadArchives** depende de **sourcesJar**, de tal manera que siempre despleguemos la librería con su paquete con el código fuente.

```

view plain print ?
01. apply plugin: 'maven'
02.
03. //Tarea que va a generar el jar con los fuentes
04. task sourcesJar(type: Jar) {
05.     classifier = 'sources'
06.     //especificamos donde se encuentran los fuentes del proyecto.
07.     from android.sourceSets.main.java.srcDirs
08. }
09.
10. uploadArchives {
11.     artifacts {
12.         //Añadimos a los artefactos de despliegue, el artefacto de la tarea encargada de ge
13.         archives sourcesJar
14.     }
15. }
16.
17. repositories {
18.     mavenDeployer {
19.         repository(url: "http://192.168.0.109:8080/repository/snapshots") {
20.             authentication(userName: "www.autentia.com", password: "www.adictosaltrabajo.com")
21.         }
22.
23.         uniqueVersion = false;
24.
25.         pom.groupId = "com.adictosaltrabajo"
26.         pom.artifactId = "volley"
27.         pom.version = "0.1-SNAPSHOT"
28.     }
29. }
30.
31. //Hacemos que la tarea de despliegue, dependa de la de generación de fuentes
32. uploadArchives.dependsOn sourcesJar
33.
34.

```

NOTA: El mecanismo para descarga de fuentes de forma automática por Android Studio ha sido liberado con la versión 0.4.6 ([mas info aqui](#)) pero parece ser que aún no va del todo fino.

## 4. Conclusiones

Estas son solo dos formas en las que podemos incorporar Volley a nuestro proyecto, pero no son únicas. Se podrían combinar entre ellas, podríamos usar maven (no el plugin de maven para Gradle) para hacer hacer el despliegue...

Las ventajas que veo a cada método son:

- Generación de jar y añadido de la librería copiandola a mano en libs
  - Permitimos que desarrolladores que no tengan acceso a un repositorio privado puedan compilar el proyecto
- Generación con gradle y despliegue con plugin de maven para gradle.
  - Evitamos subir librerías al contol de versiones.
  - Separamos código fuente de binarios.
  - Si tenemos varios proyectos que use Volley, no tenemos que copiar la librería en todos, solo añadir la dependencia
  - Es el mecanismo estandar de Gradle

Para cualquier comentario, duda o sugerencia, tenéis el formulario que aparece a continuación.

Un saludo.

**A continuación puedes evaluarlo:**

[Regístrate para evaluarlo](#)

**Por favor, vota +1 o compártelo si te pareció interesante**

Share |



Ánimate y coméntanos lo que pienses sobre este **TUTORIAL**:

» **Regístrate** y accede a esta y otras ventajas «



Esta obra está licenciada bajo [licencia Creative Commons de Reconocimiento-No comercial-Sin obras derivadas 2.5](#)

Copyright 2003-2014 © All Rights Reserved | [Texto legal y condiciones de uso](#) | [Banners](#) | [Powered by Autentia](#) | [Contacto](#)

