# Be news Library building and Server setting

## Library building

The library shall be built for every applications deployment. The steps is automatically done configuring build.gradle with the following rules and tasks:

task buildSharedLib << {

println 'Building shared library'

exec {

executable "./build.sh"

}

}

task buildSharedLibRelease << {

buildSharedLib

}

task buildSharedLibDebug << {

buildSharedLib

}

project.afterEvaluate {

preDexDebug.dependsOn("buildSharedLib")

}

Encryption KEY is chosen by changing the build.sh’s KEY variable.

## Library building bolts and nuts

The most important part is inside by utils/encript.py which takes as argument the KEY used to encrypt every strings which needed and also payload if required.

usage :

encstring <inputDir> <outputDir> <key> <encrypter>

encpayload <inputDir> <suffix> <key> <encrypter>

The first parameter is the command which can be [encscript|encpayload]

encrypt.py is used inside the build script as follow:

KEY="ciao mondo"

python utils/encript.py encstring src/libbson/ src/libbson/preprocessed "$KEY" utils/tfc

That encrypts all the strings found in all the files contained in **src/libbson** wich are prefixed with **KKK**”string” and outputs the same files inside **/src/libbson/preprocessed**.

The **KEY** parameter is the encryption key.

Encstring command also generate a header file called rc4\_enc.h containing an array with the rot13 processed key.

An example of string encryption:

printf(KKK"you are welcome\n"); ==> printf(ENC("E951F6D72BEA61"))

## Server settings

- To encrypt payloads, the following command shall be used:

**encpayload <inputDir> <suffix> <key> <encrypter>**

The Servers takes as input a file named batch.txt (which is searched by default if no argument is supplied) which has the following format:

#Date|Title|headline|content|type[1=txt,2=audio,3=video,4=img,5=html]|filepath|imei|trial

1381212176|Good news|new bee discovered||4|payload/special.jpg||

1383312176|Bad news|maia doesn't exist ||4|img/maia.jpg||3

23-11-2014 12:25|new word|great||4|payload/xig.jpg||

1385212176|Good news|new bee discovered||4|payload/digest.jpg||

On Every line contains a news (or another type of information) that shall be send to a client, empty lines are skipped, lines which begin with hash sign (‘#’) are skipped and interpreted as comment.

Lines are interpreted as pipe sign separated list (‘|’) and interpreted positionally:

**Date**: it can be an epoch in seconds or a date, human readable dates shall be inserted using this format:

day-month-year hours:minutes i.e : 23-11-2014 12:25

**Title:** A string containing the news title.

**Headline:** The news summary.

**Content:** A string with the full news.

**Type:** A number indicating the attachment type.

**Filepath:** The server side path of the attachment to include inside the news.

**Imei:** If specified, indicates the news destination IMEI. Empty imei means every clients.

**Trial:** When specified, indicates the maximum number of resending trials in case of errors.

# BSON news feed library

The main aim of the library is to give a smooth and simple interface that can be used by an android apps to connect and retrieve bson news from a feed service.

## APIs

There are only two APIs available:

**getToken**

This function returns a valid bson token used to obtain a bson news from a server.

The java application shall ask the token and then it shall send it to the server, which in turn will serve the next news if available.

public static native byte[] getToken(String imei, String cks, String baseDir );

Parameters:

String **imei :** this is the caller imei number, used by the server to identify authorized subscriptions.

String **cks :** the checksum of the last received messages from the server, this field is obtained from the serialize() returned hashmap**1**.

String **baseDir :** this is the base dire where news must be stored.

For example, using as app name “org.bbc.last” a feasible base directory can be: /data/data/app/org.bbc.last/files/.

It’s mandatory for the applications to have writing and reading permissions on that directory.

Return:

A byte array is returned, null in case of error.

**Serialize**

A bson object from the server shall be parsed using the serialize interface. The function simply takes the base directory where to store the news and a ByteBuffer which contains the response from the server.

As result an hasmap is returned with all the informations of the decoded news.

public static native HashMap<String,String> **serialize**(String baseDir, ByteBuffer payload);

Parameters:

String **baseDir :** this is the base dire where news must be stored.

For example, using as app name “org.bbc.last” a feasible base directory can be: /data/app/org.bbc.last/files/.

Applications shall own writing and reading permissions.

ByteBuffer **payload:** server reply.

Return:

The returned HashMap is a simple name,value array used to gather informations about the decoded news. The following is an example of it:

|  |  |
| --- | --- |
| Key | Value examples |
| type | img |
| path | /data/app/org.bbc.last/files/img/1385212176.jpg |
| date | 1385212176 |
| title | News from Rosetta |
| headline | Life has been discovered on the comet |
| content | Elementary form of bactarias has been found on comet 67P/Churyumov–Gerasimenko! |
| checksum | 23dk3rfdksxcwreed3 |

## Library Integration

To be able to use the library and the server, an app shall include libbson.so library inside the specific application jniLibs.

It can be easily accomplished using a gradle rules like the following:

dependencies {

compile fileTree(**dir: 'libs'**, include: ['\*.jar'])

.....

}

And placing libbsons.so inside a directory called libs within the application root directory, libs dir shall reflect the architecture for which has been compiled the library, i.e:

libs/armeabi/libbson.so

To call the library functions a utility class has been prepared, it’s enough include it with the app srcs and use it as shown below:

/\* Get a bson object\*/

obj=BsonBridge.getTokenBson(imei,cks,getDumpFolder());

/\* save the object \*/

HashMap<String,String> ret=BsonBridge.serializeBson(getDumpFolder(), result);