Google Play Expansion Files

Google Play currently requires that your APK file be no more than 50MB. Gameloft games need more space for high-fidelity graphics, media files, or other large assets. Previously, this data was hosted on Gameloft's server and downloaded by the installer on first launch. For applications distributed on Google Play, we can now attach two large expansion files that supplement each APK.

Most of the work related to the files will be automated in future versions of the Gameloft Android Installer.

Before continuing this article, please read the following article: http://developer.android.com/guide/market/expansion-files.html

Each of the two 2GB files can be any format as long as the filename is respected (more below). For Gameloft games, the split should be done as follows:

- The *main expansion* file is the primary expansion file for additional resources required by your application.
- The *patch expansion* file is optional and intended for small updates to the main expansion file.

The Filename

Each expansion file you upload can be any format you choose (ZIP, PDF, MP4, etc.). Regardless of the file type, Google Play will rename the files using the following scheme:

[main|patch].<expansion-version>.<package-name>.obb

There are three components to this scheme:

main or patch: Specifies whether the file is the main or patch expansion file. There can be only one main file and one patch file for each APK.

<expansion-version>: This is an integer that matches the version code of the APK with which the expansion is first associated (it matches the application's android:versionCode value).

<package-name>: Your application's Java-style package name.

For example, suppose your APK version is 100 and your package name is com.gameloft.android.fungame. If you upload a main expansion file, the file is renamed to:

main.100.com.gameloft.android.fungame.obb

The installer will return the correct path and filename to the game as follows:

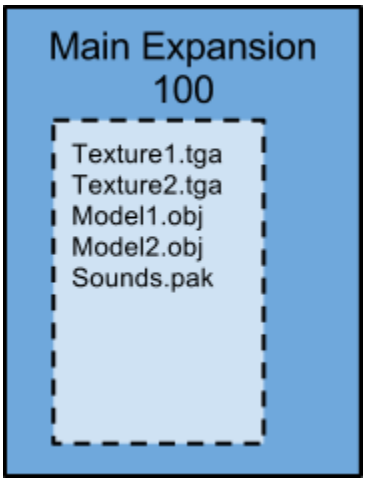
[INSTALLER FUNCTION CALLS HERE]

A Concrete Example

Let's look at the example of FunGame v1.0.0 with a data pack weighing 1GB.

When we first release FunGame, only the main expansion is needed. The Google Play app on the device will automatically download the apk & the main expansion to the user's device.

The main expansion would look like this:



One month later, we need to send an update. This update represents some minor bug fixes & some small new features. Most resources are still the same as the ones in the older version. If we re-upload a new main expansion for v1.0.1, users will need to re-download a 1GB+ file. This is not a very nice user experience.

To workaround this issue, we use the patch expansion. The patch expansion only holds the new files required to run v1.0.1 that were not in the original main expansion. We now have 2 scenarios:

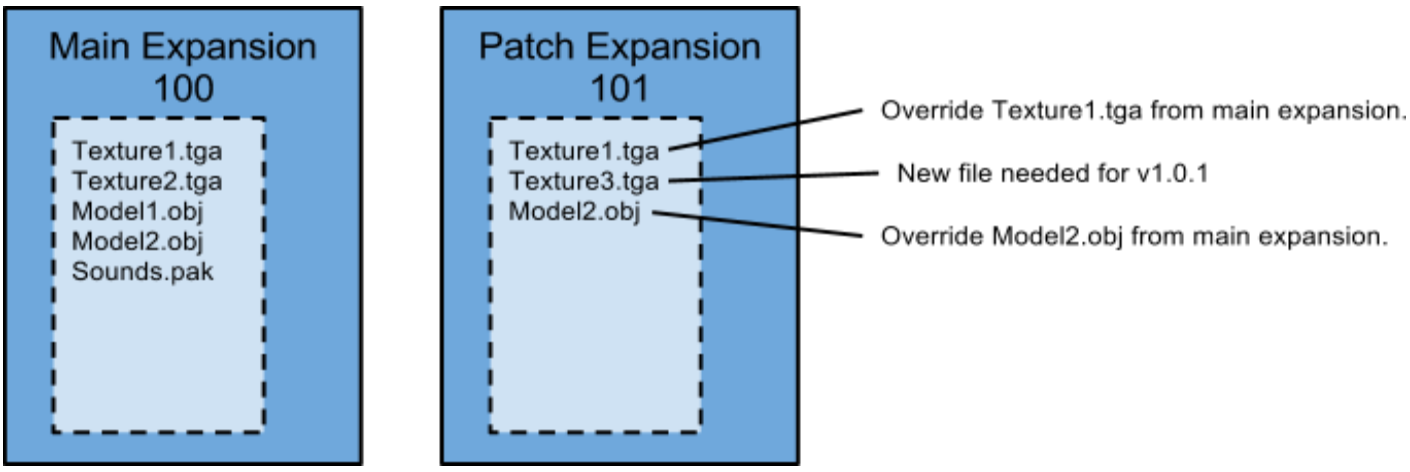
Users who already have FunGame v1.0.0 installed will only need to download:

- FunGame101.apk
- patch.101.com.gameloft.android.fungame.obb

New users who did not have FunGame installed will need to download:

- FunGame101.apk
- main.100.com.gameloft.android.fungame.obb
- patch.101.com.gameloft.android.fungame.obb

The contents of the files would be as follows:



When we update to FunGame v1.0.2, the same idea applies:

Users who already have FunGame v1.0.0 installed will only need to download:

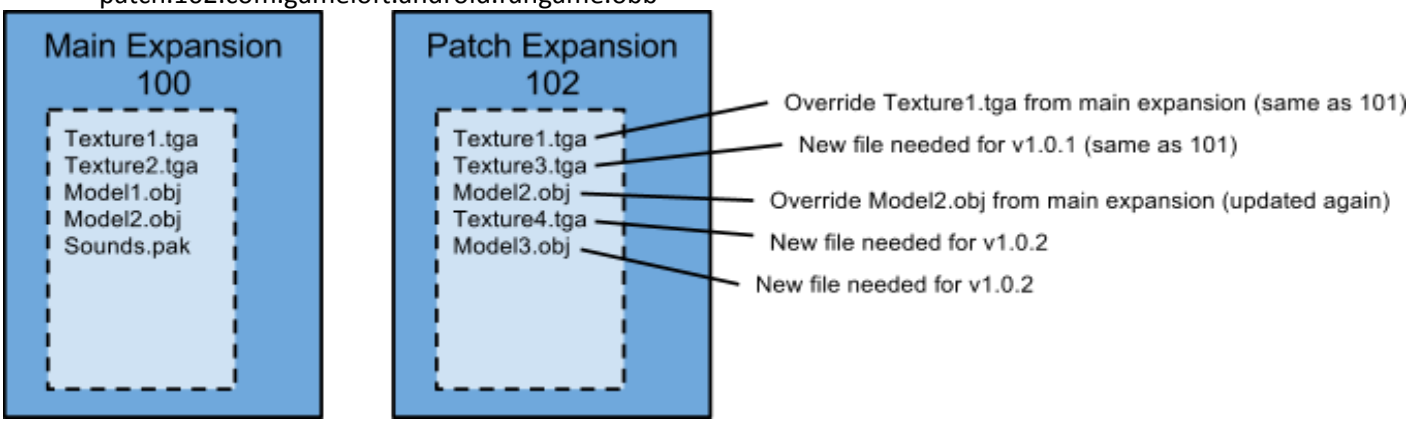
- FunGame102.apk
- patch.102.com.gameloft.android.fungame.obb

Users who already have FunGame v1.0.1 installed will only need to download:

- FunGame102.apk
- patch.102.com.gameloft.android.fungame.obb

New users who did not have FunGame installed will need to download:

- FunGame102.apk
- main.100.com.gameloft.android.fungame.obb
- patch.102.com.gameloft.android.fungame.obb



In each new version, the new patch file must include all previous patch files within it so that the user can update from any previous version. As we release more an more patches, we are increasing the number of duplicate files. If we continue in this manner, we will quickly reach a point where the patch file is larger than the main file which does not make sense. To work around this problem, we apply the following rule:

If the patch expansion filesize is larger than ½ the main expansion, we recreate the main expansion.

The Override System

Each game must implement its own system to select which version of the file to use - main expansion or patch expansion.

Each expansion file should be in zip format and contain the files required for the game. The zip format is chosen since it is a very simple way to simulate a file system. The zipped package can be compressed (can lead to longer loading times) or uncompressed (will represent a larger package). In the case where the game already has a packing system and all the resources can be easily packed into a single file, exceptions can be made but this must be verified with the HQ producer.

The algorithm for selecting correct file should be as follows:

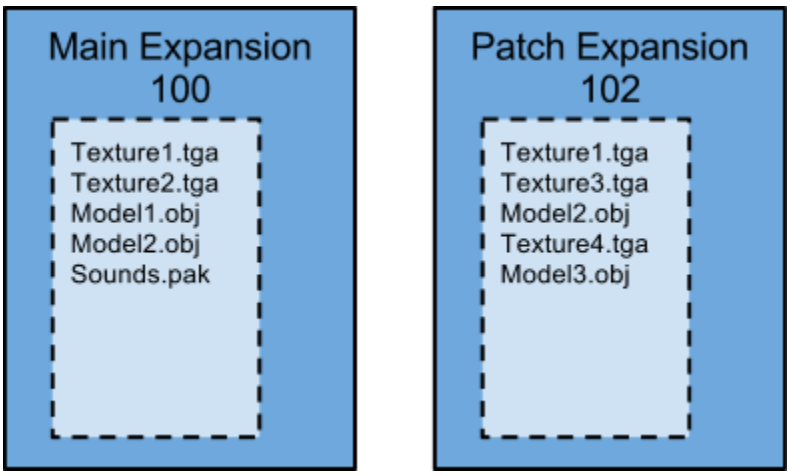
Search for the file in the patch expansion.

If the file is found in the patch expansion, open it.

Search for the file in the main expansion

If the file is found in the main expansion, open it.

However, doing this each time a file is loaded can be extremely slow. The game should build a list of files present in each pack so that it knows exactly where to open each file from. Looking at the example of the expansions below:



Our list of files would look like this:

File	Location
Texture1.tga	Patch Expansion
Texture2.tga	Main Expansion
Texture3.tga	Patch Expansion
Texture4.tga	Patch Expansion
Model1.obj	Main Expansion
Model2.obj	Patch Expansion
Model3.obj	Patch Expansion
Sounds.pak	Main Expansion

The Native Library (.so)

In most cases, the native code should be directly in the apk. There are some special cases where a particular device will need a small modification to the .so file. To handle this case, a secondary download has been included in the installer. The idea is identical to the patch expansion, except that the 2nd patch is hosted on Gameloft's servers.

The installer will handle loading the native code automatically using this algorithm:

Search for the native code in the patch expansion from Gameloft's servers.

    If the file is found in the patch expansion, load it.

Search for the native code in the patch expansion.

    If the file is found in the patch expansion, load it.

Search for the native code in the main expansion

    If the file is found in the main expansion, load it.

Search for the native code in the apk itself.

    If the file is found in apk itself, load it.

The game must use the handle data loading if a secondary, device-specific patch is required:

Search for the file in the patch expansion from Gameloft's servers.

If the file is found in the patch from Gameloft's servers, open it.

Search for the file in the patch expansion.

If the file is found in the patch expansion, open it.

Search for the file in the main expansion

If the file is found in the main expansion, open it.

Handling multiple resolutions

The game will likely need to have multiple data packs for each category of resolutions. This will be handled by the Manifest Filters described here: http://developer.android.com/guide/appendix/market-filters.html

It is important to note that it is not possible to split packs by phone name like it was on Gameloft's servers. This is a restriction on Google's side and we cannot work around it.

	Low density (120), *ldpi*	Medium density (160), *mdpi*	High density (240), *hdpi*	Extra high density (320), *xhdpi*
Small screen	QVGA (240x320)	Yellow	480x640	blue
Normal screen	WQVGA400 (240x400) WQVGA432 (240x432)	HVGA (320x480)	WVGA800 (480x800) WVGA854 (480x854) 600x1024	640x960
Large screen	WVGA800** (480x800) WVGA854** (480x854)	WVGA800* (480x800) WVGA854* (480x854) 600x1024	green	blue
Extra Large screen	1024x600	WXGA (1280x800)† 1024x768 1280x768	1536x1152 1920x1152 1920x1200	2048x1536 2560x1536 2560x1600

Each color in the table above represents a different APK, and so, a new set of data that can be associated to it.

To handle multiple APK you have to follow a few rules when making the APK. First you should that APK have 2 different version:

- **android:versionCode :** This version is **not** visible for the user. It must be different for all APKs, **Android marketplace doesn't let us upload 2 APK having the same version code**. See below for more info.
- **android:versionName :** This version is visible from the user if he goes in settings->application->Select your application. **It must be the same accross all APK for the same game update** / Game

version.3

The android:versionCode must be build using the following way:

- Game version (without point) usually it's a 3 digits number : 003, 100, 101, etc.
- The smallest number supported by this apk in the table below:

	Low density (120), *ldpi*	Medium density (160), *mdpi*	High density (240), *hdpi*	Extra high density (320), *xhdpi*
Small screen	11	12	13	14
Normal screen	21	22	23	24
Large screen	31	32	33	34
Extra Large screen	41	42	43	44

For example if this is the first version fo the game (0.0.1), and your build support 11, 12 and 13, then you version will be:

- versionCode : 00111
- versionName : 0.0.1

If your second apk support : 31, 22, 13, you version will be:

- versionCode: 00113
- versionName: 0.0.1.

Separating the data

Google does not allow us to have a separate APK file for each device. This means that we cannot afford having a split packs for powerful and weak devices having the same resolution. For example, the Adreno 200 (weak gpu) needs to share a pack with Adreno 220 (really strong GPU). Here are common problems and solutions:

Problem: Handling AMOLED screens and their oversaturated screens

Solution: This can be partially handled through shaders. If there are textures that really cannot be modified by a shader, you will need to duplicate them in the data pack. This needs to be done only for extreme cases and it is the responsibility of the local producer to explain the limitations to the HQ producer.

Problem: Need to optimize the resources for performance gains

Solution: This needs to be done in the code itself, not through the resources

Problem: Need to modify the game design to fit the smaller / larger screens

Solution: For this, the multiple APK support needs to be used to split the builds by screen size as described in the table from the previous section.

Problem: Performance is really slow after reading from the APK

Solution: Reading from the a zipped package can be slower than reading resources directly from individual files. All loading must be done with a progress bar on the screen. No lazy caching (load graphics as you need them) or streaming (pass graphics to GPU while in action phase) allowed. Additionally, one can try lowering the compression on the archive and should build a list of file locations as described in **The Override System**.