Reducing OTA Size

This page describes build changes added to AOSP to reduce unnecessary file changes between builds. Device implementers who maintain their own build system can use this information as a guide for reducing over-the-air (OTA) update size.

Android OTAs occasionally contain changed files that do not correspond to code changes but are instead artifacts of the build system. This can occurs when the same code built at different times, from different directories, or on different machines, produces a large number of changed files. These excess files not only increase the size of an OTA, but make it difficult to determine which code is changed in the OTA.

To make the contents of an OTA more transparent, AOSP includes build system changes designed to reduce OTA size by eliminating unnecessary file changes between builds. The aim is to reduce the size of OTAs to include only the files that relate to the patches contained in the OTA. AOSP also includes a build diff tool (#the_build_diff_tool) that filters out common build-related file changes and provides a cleaner build file diff.

The build system can create unnecessary file diffs in several ways. The following sections discuss some of these issues and solutions, providing examples of fixes in AOSP when possible).

File order

Problem: Filesystems don't guarantee a file order when asked for a list of files in a directory, though it's commonly the same for the same checkout. Tools such as `ls` sort the results by default, but the wildcard function used by commands such as `find` and `make` do not. Before using these tools, you must sort the outputs.

Solution: Users of tools such as `find` and `make` with wildcard must sort the output of these commands before using them. Use of `$(wildcard )` or `$(shell find )` in Android.mk files should also be sorted. Some tools, such as Java, do sort inputs so verify sorting is necessary.

Examples: Many instances were fixed in the core build system using the builtin `all-*-files-under` macro, which includes `all-cpp-files-under` (as several definitions were spread out in other makefiles). For details, refer to the following CLs:

- https://android.googlesource.com/platform/build/+/4d66adfd0e6d599d8502007e4ea9aaf82e95569f (https://android.googlesource.com/platform/build/+/4d66adfd0e6d599d8502007e4ea9aaf82e95569f)
- https://android.googlesource.com/platform/build/+/379f9f9cec4fe1c66b6d60a6c19fecb81b9eb410 (https://android.googlesource.com/platform/build/+/379f9f9cec4fe1c66b6d60a6c19fecb81b9eb410)
- https://android.googlesource.com/platform/build/+/7c3e3f8314eec2c053012dd97d2ae649ebeb5653 (https://android.googlesource.com/platform/build/+/7c3e3f8314eec2c053012dd97d2ae649ebeb5653)
- https://android.googlesource.com/platform/build/+/5c64b4e81c1331cab56d8a8c201f26bb263b630c (https://android.googlesource.com/platform/build/+/5c64b4e81c1331cab56d8a8c201f26bb263b630c)

Build directory

Problem: Changing the directory in which things are built can cause the binaries to be different. Most paths in the Android build are relative paths so `__FILE__` in C/C++ isn't a problem. However, the debug symbols encode the full pathname by default, and the `.note.gnu.build-id` is generated from hashing the pre-stripped binary, so it will change if the debug symbols change.

Solution: AOSP now makes debug paths relative. For details, refer to CL: https://android.googlesource.com/platform/build/+/6a66a887baadc9eb3d0d60e26f748b8453e27a02 (https://android.googlesource.com/platform/build/+/6a66a887baadc9eb3d0d60e26f748b8453e27a02).

Timestamps

Problem: Timestamps in the build output result in unnecessary file changes. This is likely to happen in the following locations:

- `__DATE__/__TIME__/__TIMESTAMP__` macros in C or C++ code.
- Timestamps embedded in zip-based archives.

Solutions/Examples: To remove timestamps from the build output, use the instructions in the sections below.

__DATE__/__TIME__/__TIMESTAMP__ in C/C++

These macros always produce different outputs for different builds, so they shouldn't be used. Here are a few options on how to eliminate these macros:

- Just remove them, they often aren't necessary. For an example, refer to https://android.googlesource.com/platform/system/core/+/30622bbb209db187f6851e4cf0cdaa147c2fca9f (https://android.googlesource.com/platform/system/core/+/30622bbb209db187f6851e4cf0cdaa147c2fca9f)
- To uniquely identify the running binary, read the build-id from the ELF header.
- To know when the OS was built, read the `ro.build.date` (should work for everything except incremental builds, which may not update this date). For an example, refer to https://android.googlesource.com/platform/external/libchrome/+/8b7977eccc94f6b3a3896cd13b4aeacbfa1e0f84 (https://android.googlesource.com/platform/external/libchrome/+/8b7977eccc94f6b3a3896cd13b4aeacbfa1e0f84)

Note: We turned on `-Werror=date-time` so using timestamps is a build error.

Embedded timestamps in archives (zip, jar)

We fixed the problem of embedded timestamps in zip archives by adding `-X` to all uses of the `zip` command, so the UID/GID of the builder and the extended Unix timestamp are not embedded in the zip file.

A new tool, `ziptime` (located in `/platform/build/+/master/tools/ziptime/` (https://android.googlesource.com/platform/build/+/master/tools/ziptime/)) resets the normal timestamps in the zip headers. For details, refer to the README file (https://android.googlesource.com/platform/build/+/master/tools/ziptime/README.txt).

The `signapk` tool sets timestamps for the APK files that may vary depending on the server timezone. For details, refer to the CL https://android.googlesource.com/platform/build/+/6c41036bcf35fe39162b50d27533f0f3bfab3028 (https://android.googlesource.com/platform/build/+/6c41036bcf35fe39162b50d27533f0f3bfab3028).

Version strings

Problem: APK version strings often had the BUILD_NUMBER appended to the hardcoded version. Even if nothing else changed in the APK, the APK would still be different.

Solution: Remove the build number from the APK version string.

Examples:

- https://android.googlesource.com/platform/packages/apps/Camera2/+/5e0f4cf699a4c7c95e2c38ae3babe6f20c258d27 (https://android.googlesource.com/platform/packages/apps/Camera2/+/5e0f4cf699a4c7c95e2c38ae3babe6f20c258d27)
- https://android.googlesource.com/platform/build/+/d75d893da8f97a5c7781142aaa7a16cf1dbb669c (https://android.googlesource.com/platform/build/+/d75d893da8f97a5c7781142aaa7a16cf1dbb669c)

Consistent build tools

Problem: Tools that generate installed files must be consistent (the same input should always produce the same output).

Solutions/Examples: Changes were required in the following build tools:

- **NOTICE file creator**. The NOTICE file creator needed the changes. Refer to CL: https://android.googlesource.com/platform/build/+/8ae4984c2c8009e7a08e2a76b1762c2837ad4f64 (https://android.googlesource.com/platform/build/+/8ae4984c2c8009e7a08e2a76b1762c2837ad4f64)
- **Java Android Compiler Kit (Jack)**. The Jack toolchain required an update to handle an occasional change in generated constructor ordering. Refer to CL: https://android.googlesource.com/toolchain/jack/+/056a5425b3ef57935206c19ecb198a89221ca64b (https://android.googlesource.com/toolchain/jack/+/056a5425b3ef57935206c19ecb198a89221ca64b)
- **ART AOT compiler (dex2oat)**. The ART compiler binary required an update to create a deterministic image. Refer to CL:

https://android.googlesource.com/platform/art/+/ace0dc1dd5480ad458e622085e51583653853fb9 (https://android.googlesource.com/platform/art/+/ace0dc1dd5480ad458e622085e51583653853fb9)

- **The libpac.so file (V8)**Every build creates a different `/system/lib/libpac.so` file because the V8 snapshot changes for each build. The solution is to remove the snapshot. Refer to CL: https://android.googlesource.com/platform/external/v8/+/e537f38c36600fd0f3026adba6b3f4cbcee1fb29 (https://android.googlesource.com/platform/external/v8/+/e537f38c36600fd0f3026adba6b3f4cbcee1fb29)
- **Application pre-dexopt'd (.odex) files**. The pre-dexopt'd (.odex) files contained uninitialized padding on 64-bit systems. Refer to CL: https://android.googlesource.com/platform/art/+/34ed3afc41820c72a3c0ab9770be66b6668aa029 (https://android.googlesource.com/platform/art/+/34ed3afc41820c72a3c0ab9770be66b6668aa029)

Using the build diff tool

For cases where it is not possible to eliminate build-related file changes, we supply a build diff tool, `target_files_diff.py` (https://android.googlesource.com/platform/build/+/master/tools/releasetools/target_files_diff.py) for use in comparing two file packages. This tool performs a recursive diff between two builds, excluding common build-related file changes, such as:

- Expected changes in the build output (for example, due to a build number change).
- Changes due to known issues in the current build system.

To use the build diff tool, run the following command:

```
$ target_files_diff.py dir1 dir2
```

`dir1` and `dir2` are base directories that contain the extracted target files for each build.

Last updated June 17, 2017.