Symbolicating stack traces for engine crashes

The easiest way to symbolicate stack traces for Android and iOS is using @flutter-symbolizer-bot (see the bot's profile page for documentation). If that is not an option, the steps below explain how to do it manually.

Android

Get the symbols

1. Get the Flutter Framework or Flutter Engine revision from the report. If you have the Engine revision, skip to step 3.
2. Get the Engine revision from the Framework (this could be automated). https://github.com/flutter/flutter/blob/master/bin/internal/engine.version is the file which contains the information. Substitute the framework hash for `master` in that url.
3. With the full engine revision (e.g. cea5ed2b9be42a981eac762af3664e4a17d0a53f), you can now get the proper symbol files:

   To view the available artifacts for a build, visit this URL in a browser (replacing the engine hash with your hash):
   `https://console.cloud.google.com/storage/browser/flutter_infra_release/flutter/cea5ed2b9be42a981eac762af3664e4a17d0a53f`

   To download the symbols for android-arm, download this URL *using your browser* (replacing the hash again, and noting that this URL is on a different host, "storage", compared to the one above, which uses "console"):
   `https://storage.cloud.google.com/flutter_infra_release/flutter/cea5ed2b9be42a981eac762af3664e4a17d0a53f/android-arm/symbols.zip`.

   Please be aware that the type of the symbol must match your Apps release type. In above example, this refers to a android-arm **debug** build. If you work with a **release** or **profile** build, the URLs would look like this:

   - https://storage.cloud.google.com/flutter_infra_release/flutter/cea5ed2b9be42a981eac762af3664e4a17d0a53f/android-arm-release/symbols.zip
   - https://storage.cloud.google.com/flutter_infra_release/flutter/cea5ed2b9be42a981eac762af3664e4a17d0a53f/android-arm-profile/symbols.zip

   You have to use your browser because it does authentication.

Symbolicate with ndk-stack

Once you have the symbols unzipped, you can use ndk-stack from your Android NDK. Suppose `stack.txt` contains the stack (including the leading `*** *** ***` line from the crash):

```
.../ndk/prebuilt/linux-x86_64/bin/ndk-stack -sym .../path/to/downloaded/symbols < stack.txt
```

Or on macOS:

```
.../ndk/prebuilt/darwin-x86_64/bin/ndk-stack -sym .../path/to/downloaded/symbols < stack.txt
```

Some debugging tools, like *pidcat* may not show the full tombstone logs. In that case, please use `adb logcat` directly and copy the full output.

Symbolicate with addr2line

A alternative way to symbolicate is by using the addr2line tool. It is bundled with the NDK. To use it, simply call it with a path to the .so file downloaded above and feed the stack addresses to it manually. For example, on macOS:

```
% $ANDROID_HOME/ndk/20.0.5594570/toolchains/llvm/prebuilt/darwin-x86_64/bin/aarch64-linux-android-addr2line -e
~/Downloads/libflutter.so
```

The tool is now awaiting your input, so let's feed it a memory address:

```
0x00000000006a26ec
/b/s/w/ir/cache/builder/src/out/android_release_arm64/../../third_party/dart/runtime/vm/dart_api_impl.cc:1366
```

This revealed address `0x00000000006a26ec` to correspond with `dart_api_impl.cc:1366`.

Making sure you got the right libflutter.so

The build system sets a build id for each `libflutter.so` file. In the tombstones, you would see the ID like so:

```
#00 pc 000000000062d6e0  /data/app/com.app-tARy3eLH2Y-QN8J0d0WFog==/lib/arm64/libflutter.so!libflutter.so (offset
0x270000) (BuildId: 34ad5bdf0830d77a)
```

This equals to a build id of **34ad5bdf0830d77a**. The `libflutter.so` debug files downloaded as shown above could be verified using the `file` command:

```
% file ~/Downloads/libflutter.so
/Users/user/Downloads/libflutter.so: ELF 64-bit LSB shared object, ARM aarch64, version 1 (SYSV), dynamically linked,
BuildID[xxHash]=34ad5bdf0830d77a, with debug_info, not stripped
```

Ensure the build IDs match, else you will not be able to symbolicate.

Expanding Git Revisions

Go to a commit page with the short commit as the last fragment of the URL: (e.g. https://github.com/flutter/flutter/commit/9cb914df1 or https://github.com/flutter/engine/commit/2a13567) and then find the full revision on the page.

Symbolicating local builds

If you have made your own builds, you can use ndk-stack directly:

```
# dev/engine is where your engine's .gclient file is
# android_debug_unopt is whatever build of your engine you are using
adb logcat | ~/dev/engine/src/third_party/android_tools/ndk/prebuilt/linux-x86_64/bin/ndk-stack -sym
~/dev/engine/src/out/android_debug_unopt
```

iOS

The dSYM file for `Flutter.framework` (which is the Flutter Engine) for ios-release builds can be downloaded from Google Cloud Storage. Follow the steps from the Android section in this guide, but in the last step use a download url following this schema: `https://storage.cloud.google.com/flutter_infra_release/flutter/38a646e14cc25f5a56a989c6a5787bf74e0ea386/ios-release/Flutter.dSYM.zip` (replace the engine hash with your hash).

Symbolicating local builds

If you built your local engine in debug or profile Dart modes, the framework's dylib's symbols aren't stripped and are available by default.

Crashes in Dart AOT code

If the crash is in AOT Dart code (in `--release` or `--profile` builds) on iOS, and you can build your own engine, these steps will be helpful for the VM team to fix the bug:

- Prepare a reduced test case.
- Compile the engine in profile mode and disable optimizations for symbolicated traces.
  - `sky/tools/gn --ios --unopt --runtime-mode profile; ninja -C out/ios_profile_unopt -j800` .
- Launch the application via the Xcode project and make it crash in the debugger.
- File a bug on dart-lang/sdk.
- Dump the register state and paste it into the bug.
  - In `lldb` , `register read` .
- Copy the backtrace and paste it into the bug.
  - In `lldb` , `thread backtrace` . Assumes you are on the thread that crashed. If not, `thread select n` .
- Disassemble the last frame and paste it into the bug.
  - In `lldb` , `frame select 0` then `disassemble --frame` .
- Disassemble using the `gen_snapshot` and paste the function into the bug for more detailed information.
  - In the backtrace, look for the name of the precompiled function that caused the crash.
  - Open `SnapshotterInvoke` from Xcode and to the `RunCommand ... Snapshotter` call, add the `--disassemble` flags.
  - Modify the `RunCommand` function to dump to a file.
  - Build again. The results should end up in the file.
  - Look for the function name (by substring match) in this file and copy out that information to the bug.
- Ping someone on dart-lang/sdk.