Background

Bitcode requires that *all libraries being built* have bitcode enabled, and that they're all built with compatible toolchains. If you're building the engine locally, you must specify the `--bitcode` flag to the `gn` command, and this will result in building with the local Xcode installation's clang for compatibility reasons. You must also ensure that all plugins are built with bitcode - the best way to do so is to simply try to build with bitcode (instructions below) and see if the build succeeds.

The binaries we ship with the framework now have bitcode, as of version 1.9.6 (some commits earlier than this may have bitcode available as well).

Bitcode takes longer to build, and results in much larger intermediate binaries - since they contain the bitcode representation of your code embedded to be recompiled later. Local testing has shown that binaries are smaller when thinned and rebuilt from bitcode - for example, building the Flutter Gallery with bitcode for an iPhone 8+ results in a 39mb IPA.

Enabling bitcode on a Flutter app

During the experimental phase of this rollout, bitcode will not be enabled by default in Flutter application templates. It is very unlikely that we will auto-migrate templates to enable bitcode either, as this could potentially break projects that are consuming other libraries that are not bitcode enabled.

In your Xcode project, ensure that `ENABLE_BITCODE` is set to `YES` for all targets. Open the xcworkspace in Xcode:

```
my_flutter_app$ open ios/Runner.xcworkspace
```

Click on Runner, and then build settings. Ensure that *all* build settings are visible. Search for `bitcode`, and change it to `Yes`. Ensure this is done for all targets, including any targets created by the Cocoapods for plugins.

Before:

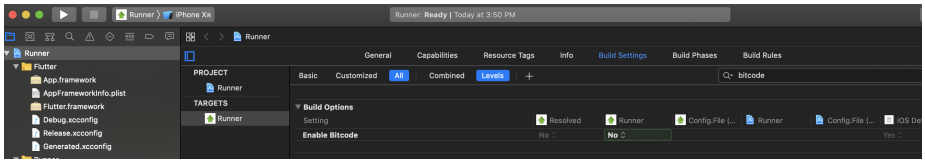

Figure 1: Xcode settings

After:

Or with plugins (you can select multiple targets at once by holding shift or cmd while clicking):

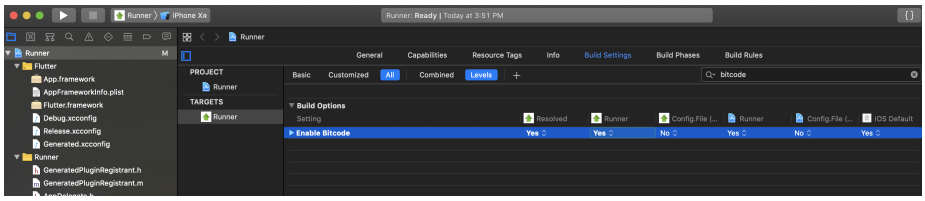

Figure 2: Xcode settings

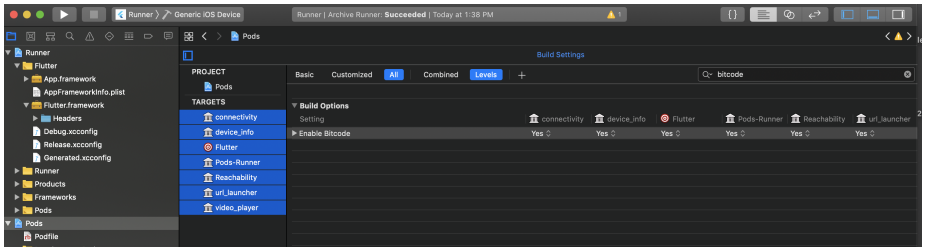

Figure 3: Xcode settings with plugins

You should also remove `config.build_settings['ENABLE_BITCODE'] = 'NO'` from the Podfile if you have plugins. For example, you can see how this was done for flutter_gallery.

Now, builds of your project will use bitcode. You can validate this by creating an archive for the product in Xcode and creating distribution artifacts for local development that use bitcode recompilation:

to check on the app size. And of course, you can do this to actually distribute your app (in which case you would not want to pre-thin the app).

Building a local engine with bitcode

The following setps will give you a profile engine with bitcode. GOMA cannot be used because bitcode requires using the Xcode toolchain, which is not GOMA aware.

```
src$ ./flutter/tools/gn --ios --runtime-mode=profile --no-goma --bitcode
src$ ./flutter/tools/gn --runtime-mode=profile # The host does not have to be built with bi
src$ autoninja -C out/ios_profile
src$ autoninja -C out/host_profile
```

If you build your local engine without bitcode, make sure to disable bitcode in your consuming app, as it will otherwise fail to build.

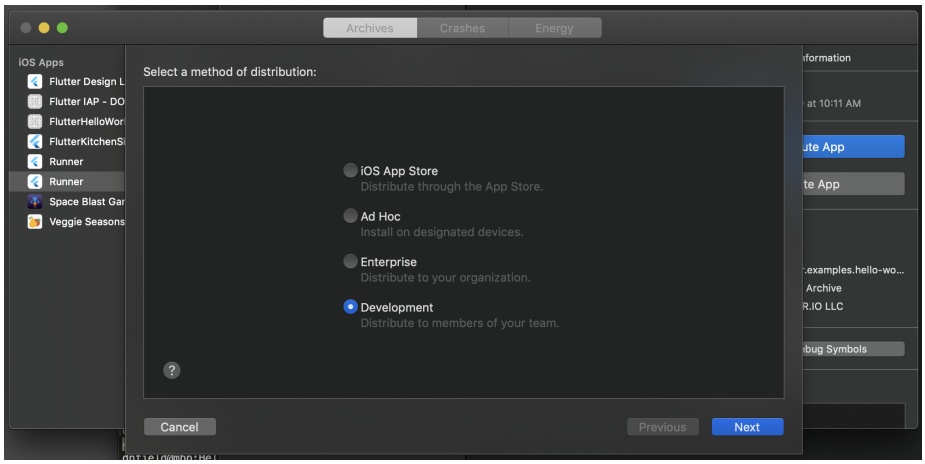

Figure 4: for development

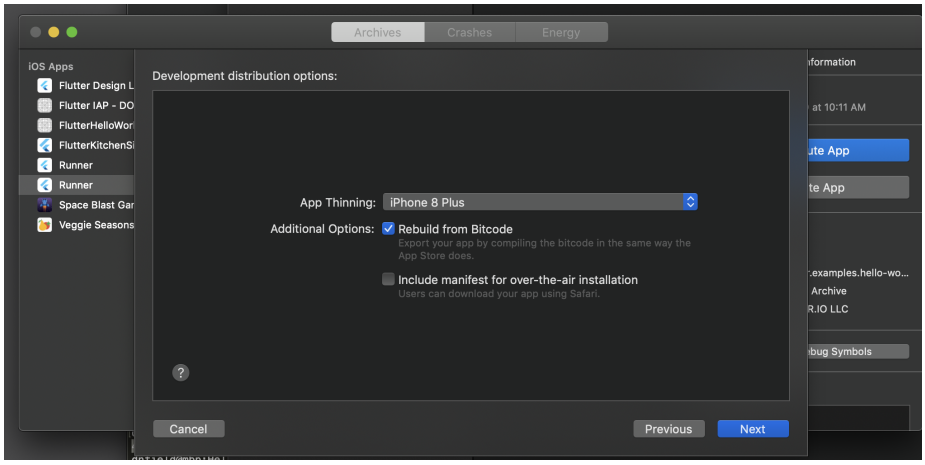

Figure 5: archive with bitdcode

Additional technical details

Bitcode can be either `marker` or "regular". Marker means that there is no actual bitcode in the binary - we use this for debug and profile builds. This makes the build go faster, and since these builds should not be shipped to the store anyway they do not need to have full bitcode. Release builds get actual bitcode, which embeds blobs of structured data into your binary that can later be recompiled.

Bitcode is meant to be platform independent. Flutter and Dart use assembly code on iOS which is platform specific. These sections of code get annotated with a `__LLVM,__asm` section to tell the compiler that they should be left as-is.