We aspire to reach a state where developers are able to use the following modes for running Flutter code. Our tools should not expose any other combinations of features or modes. Each mode corresponds to a separate build of the engine that we provide.

1. **Debug** mode on device (including simulators, emulators): Turns on all the assertions in the world, includes all debugging information, enables all the debugger aids (e.g. observatory) and service extensions. Optimizes for fast develop/run cycles. Does not optimize for execution speed, binary size, or deployment. Used by `flutter run`. Built with `sky/tools/gn --android` or `sky/tools/gn --ios`. Also sometimes called "**checked** mode" or "**slow** mode".

2. **Release** mode on device (excluding simulators, emulators): Turns off all assertions, strips as much debugging information as possible, turns off all the debugger tools. Optimizes for fast startup, fast execution, small package sizes. Disables any debugging aids. Disables service extensions. Intended for deployment to end-users. Used by `flutter run --release`. Built with `sky/tools/gn --android --runtime-mode=release` or `sky/tools/gn --ios --runtime-mode=release`.

3. **Profile** mode on device (excluding simulators, emulators): Same as release mode except that profile-mode service extensions (like the one that turns on the performance overlay) is enabled, and tracing is enabled, as well as the minimum required to support using the tracing information (e.g. observatory can probably connect to the process). Used by `flutter run --profile`. Built with `sky/tools/gn --android --runtime-mode=profile` or `sky/tools/gn --ios --runtime-mode=profile`. Not available on simulators or emulators because profiling on simulators is not representative of real performance.

4. Headless **test** mode on desktop: Same as debug mode except headless and for desktop platforms. Used by `flutter test`. Built with `sky/tools/gn`.

In addition, for our purposes during development, each of the above should be able to be built in two modes: optimized, which is what end-developers use, and unoptimized, which is what we would use when debugging the engine. Optimized is the default, unoptimized engines are built by adding `--unoptimized` to the arguments.

Artifact differences

Debug mode produces a script snapshot, which is basically tokenized sources. Comments and whitespace are missing, literals are canonicalized. There is no machine code, tree-shaking or obfuscation.

Profile and release modes produce app-aot snapshots, either as dylibs (iOS and Fuchsia) or 4-tuples of blobs (Android). Both include machine code for all

the compiled functions, code metadata, method dictionaries, class and library structures, types, etc. The machine code is fully position-independent. The dylib additionally has DWARF debug info such as function names and source positions. There is tree-shaking. Obfuscation is opt-in.

Matrix

The following axes, as described above, exist:

- debug, release, profile
- opt, unopt
- iOS, Android, macOS, Linux, Windows

In addition, some versions can select alternative graphics backends:

- iOS can choose between: OpenGL, software
- Android can choose between: Vulkan, OpenGL, software
- macOS can choose between, OpenGL, software, headless (debug only)
- Linux can choose between: OpenGL, software, headless (debug only)
- Windows can choose between: OpenGL, software, headless (debug only)

Separate from all the above, Fuchsia has the following modes:

- AOT, JIT, interpreted DBC
- Observatory present, observatory absent
- opt, unopt

In total therefore there are $3\times2\times(2+3+2+2+2) + 1\times2\times3 + 3\times2\times2$ modes, which is 84 modes.