*If you've already built the engine and have the configuration set up but merely need a refresher on actually compiling the code, see [[Compiling the engine]].*

Make sure you have the following dependencies available:

- Linux, macOS, or Windows.
  - Linux supports cross-compiling artifacts for Android and Fuchsia, but not iOS.
  - macOS supports cross-compiling artifacts for Android and iOS.
  - Windows doesn't support cross-compiling artifacts for any of Android, Fuchsia, or iOS.
- git (used for source version control).
- An IDE. See also the section at the bottom of this page for advice on setting up syntax highlighting while editing the engine.
- An ssh client (used to authenticate with GitHub).
- **Chromium's depot_tools (make sure it's in your path). We use the `gclient` tool from depot_tools.**
  - Make sure to install gclient by running ./gclient in the command line.
- Python (used by many of our tools, including `gclient`).
- On macOS and Linux: curl and unzip (used by `gclient sync`).
- On Windows:
  - Visual Studio 2017 or later (required for non-Googlers only).
  - Windows 10 SDK (required for non-Googlers only).
    - Be sure to install the "Debugging Tools for Windows" feature.
- On macOS: the latest Xcode.
- Recommended for Googlers: Goma for distributed builds. The compiling page has more information on how to set this up.

You do not need Dart installed, as a Dart tool chain is automatically downloaded as part of the "getting the code" step. Similarly for the Android SDK, it's downloaded by the `gclient sync` step below.

Run the following steps to set up your environment:

1. Fork `https://github.com/flutter/engine` into your own GitHub account. If you already have a fork, and are now installing a development environment on a new machine, make sure you've updated your fork so that you don't use stale configuration options from long ago. Do not clone this repo locally, scripts will take care of that for you.

2. If you haven't configured your machine with an SSH key that's known to github then follow the directions here: https://help.github.com/articles/generating-ssh-keys/.

3. Create an empty directory called `engine` for your copy of the repository and `cd` into it. (It is possible to use a different name, but some tools assume this name unless configured otherwise, so calling it `engine` will make thing easier.)

4. Create a `.gclient` file in the `engine` directory with the following contents, replacing `<your_name_here>` with your GitHub account name:

```
solutions = [
  {
    "managed": False,
    "name": "src/flutter",
    "url": "git@github.com:<your_name_here>/engine.git",
    "custom_deps": {},
    "deps_file": "DEPS",
    "safesync_url": "",
  },
]
```

   - Note: You can use `gclient config` command, or your favorite text editor to create the `.gclient` file.

5. `gclient sync` in that directory. This will fetch all the source code that Flutter depends on. Avoid interrupting this script, as doing so can leave your repository in an inconsistent state that is tedious to clean up. (This step automatically runs `git clone`, among other things.)

6. Add a remote for the upstream repository:

   - `cd src/flutter` (This was created in your `engine` directory by `gclient sync`.)
   - `git remote add upstream git@github.com:flutter/engine.git` (So that you fetch from the master `flutter/engine` repository, not your clone, when running `git fetch` et al.)
   - `cd ..` (Return to the `src` directory that `gclient sync` created in your `engine` directory.)

7. If you're on Linux, the following may or may not help you install required dependencies. **Note:** These scripts are distro- and version-specific, so are not guaranteed to work on any configuration. If they fail, you may need to find comparable packages to the ones that weren't found. You are free to update them if you wish for your distribution, but it is often easier to just install the packages you need as you go, particularly for the `install-build-deps.sh` script.

   - `sudo ./build/install-build-deps-android.sh`
   - `sudo ./build/install-build-deps.sh`
   - `sudo ./flutter/build/install-build-deps-linux-desktop.sh`

8. If you're on Mac:

   - install Oracle's Java JDK, version 1.8 or later

9. If you're planning on working on the buildroot repository as well, and have a local checkout of that repository, run the following commands in the `src` directory to update your git remotes accordingly: `bash`

`git remote rename origin upstream` `git remote add origin git@github.com:<your_name_here>/buildroot.git`

Next steps:

- [[Compiling the engine]] explains how to actually get builds, now that you have the code.
- [[The flutter tool]] has a section explaining how to use custom engine builds.
- [[Signing commits]], to configure your environment to securely sign your commits.

## Editor autocomplete support

### Xcode [Objective-C++]

On Mac, you can simply use Xcode (e.g., `open out/host_debug_unopt/products.xcodeproj`).

### VSCode with C/C++ Intellisense [C/C++]

VSCode can provide some IDE features using the C/C++ extension. It will provide basic support on install without needing any additional configuration. There will probably be some issues, like header not found errors and incorrect jump to definitions.

Intellisense can also use our `compile_commands.json` for more robust functionality. Either symlink `src/out/compile_commands.json` to the project root at `src` or provide an absolute path to it in the `c_cpp_properties.json` config file. See "compile commands" in the c_cpp_properties.json reference. This will likely resolve the basic issues mentioned above.

For adding IDE support to the Java code in the engine with VSCode, see "Using VSCode as an IDE for the Android Embedding".

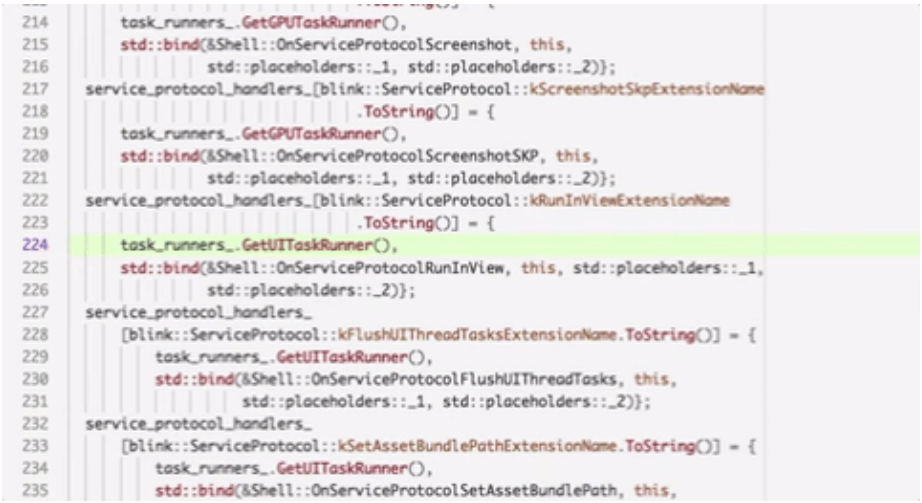
### cquery/ccls (multiple editors) [C/C++/Objective-C++]

Alternatively, cquery and a derivative ccls are highly scalable C/C++/Objective-C language server that supports IDE features like go-to-definition, call hierarchy, autocomplete, find reference etc that works reasonably well with our engine repo.

They(https://github.com/cquery-project/cquery/wiki/Editor-configuration) supports editors like VSCode, emacs, vim etc.

To set up: 1. Install cquery 1. `brew install cquery` or `brew install ccls` on osx; or 1. Build from source 1. Generate compile_commands.json which our GN tool already does such as via `src/flutter/tools/gn --ios --unoptimized` 1. Install an editor extension such as VSCode-cquery or vscode-ccls 1. VSCode-query and vscode-ccls requires the compile_commands.json to be at the project root. Copy or symlink `src/out/compile_commands.json` to

`src/` or `src/flutter` depending on which folder you want to open. 1. Follow Setting up the extension to configure VSCode-query.



```
    task_runners_.GetGPUTaskRunner(),
    std::bind(&Shell::OnServiceProtocolScreenshot, this,
              std::placeholders::_1, std::placeholders::_2)};
service_protocol_handlers_[blink::ServiceProtocol::kScreenshotSkpExtensionName
                           .ToString()] = {
    task_runners_.GetGPUTaskRunner(),
    std::bind(&Shell::OnServiceProtocolScreenshotSKP, this,
              std::placeholders::_1, std::placeholders::_2)};
service_protocol_handlers_[blink::ServiceProtocol::kRunInViewExtensionName
                           .ToString()] = {
    task_runners_.GetUITaskRunner(),
    std::bind(&Shell::OnServiceProtocolRunInView, this, std::placeholders::_1,
              std::placeholders::_2)};
service_protocol_handlers_
    [blink::ServiceProtocol::kFlushUIThreadTasksExtensionName.ToString()] = {
        task_runners_.GetUITaskRunner(),
        std::bind(&Shell::OnServiceProtocolFlushUIThreadTasks, this,
                  std::placeholders::_1, std::placeholders::_2)};
service_protocol_handlers_
    [blink::ServiceProtocol::kSetAssetBundlePathExtensionName.ToString()] = {
        task_runners_.GetUITaskRunner(),
        std::bind(&Shell::OnServiceProtocolSetAssetBundlePath, this,
```

**Using VSCode as an IDE for the Android Embedding [Java]**

1. Install the extensions vscjava.vscode-java-pack and vscjava.vscode-java-dependency.
2. Right click on the `shell/platform/android` folder in the engine source and click on `Add Folder to Java Source Path`. This creates an anonymous workspace and turns those files from "syntax mode" to "compile mode". At this point, you should see a lot of errors since none of the external imports are found.
3. Find the "Java Dependencies" pane in your Explorer view. Use the "Explorer: Focus on Java Dependencies View" command if hidden.
4. Refresh the view and find the "flutter_*" project. There should be a "_/shell/platform/android" source folder there.
5. In the "Referenced Libraries" sibling node, click the + button, navigate to `engine/src/third_party/android_embedding_dependencies` and add the entire folder. This is the equivalent of adding `"java.project.referencedLibraries": [ "{path to engine}/src/third_party/android_e ]` to your VSCode's settings.json for your user or for your workspace.
6. If you previously had a `shell/platform/android/.classpath`, delete it.

## VSCode Additional Useful Configuration

1. Create snippets for header files with this configuration. This will let you use `hdr` keyboard macro to create the boiler plate header code. Also

consider some of these settings and more tips.

2. To format GN files on save, consider using this extension.