# Contributing

xcbuild actively welcomes contributions from the community. If you run into problems, please open an issue. Pull requests are great too!

## Getting Started

To learn about xcbuild, start with the techincal overview below. Each of the components of xcbuild is documented in the `Headers` directory, with both module and API documentation available.

If you're new to xcbuild, we have a [list of issues](https://github.com/facebook/xcbuild/labels/starter) designed for new contributors to get started in the codebase. We're always happy to help people working on these issues and learning about xcbuild.

## Overview

To make xcbuild useful as more than just a build tool, it's built as a collection of libraries. Each of these libraries has its headers under `Headers` and `PrivateHeaders`, source code under `Sources`, and unit tests under `Tests`. Under `Tools` are the executables that use the xcbuild libraries, including `xcbuild` itself.

Shared components:

- `ext`: Standard library extensions from the future.

- `plist`: Reader, writer, and model for ASCII, binary, and XML property lists.

- `libutil`: Common utilities and platform abstractions.

- `process`: Process launching and parameter abstraction.

- `graphics`: Image loading, color conversion, and serialization.

Build components:

- `ninja`: Generator for Ninja build files.

- `dependency`: Parse dependency info from various tools.

- `pbxsetting`: Build setting parsing, levels, and resolution.

- `xcsdk`: Loads the available platforms, SDKs, and toolchains in a developer root.

- `pbxspec`: Parses available build settings, tools, and file types into strongly-typed structures.

- `pbxproj`: Parses project files into strongly-typed structures.

- `xcworkspace`: Parses workspaces into strongly-typed structures.

- `xcscheme`: Parses schemes into strongly-typed structures.

- `builtin`: Built-in tools used for building, like file copying and validation.

- `pbxbuild`: Performs the build. Made up of a number of subcomponents:

- `Build`: Top-level context for the scheme, targets, and action for a build.

- `Target`: Creates the build environment and settings for each target.

- `Phase`: Implementations for the various build phases (compile, link, copy, etc).

- `Tool`: Tool implementations for generating command line invocations.

- `xcformatter`: Build log formatting styles.

- `xcexecution`: Build execution engines.

- `xcdriver`: Parses command line options and drives the build.

Asset components:

- `xcassets`: Parses asset catalogs into strongly-typed structures.

- `libbom`: Reads and writes standard BOM archives.

- `libcar`: Reads and writes asset archives archives within a BOM.

- `acdriver`: Parses command line options and compiles asset catalogs.

## Style

C++ can be complex, so xcbuild tries to stick with a reasonable subset modeled after Swift, Rust, and other modern langages. Prefer immutable structures and classes. Prefer passing by const reference, and avoid non-const references. Templates are avoided when possible. Exceptions and RTTI are not used.

## Contributor License Agreement ("CLA")

In order to accept your pull request, we need you to [submit a CLA](https://code.facebook.com/cla). You only need to do this once to work on any of Facebook's open source projects.

## Code of Conduct

This project adheres to the [Open Code of Conduct](http://todogroup.org/opencodeofconduct/#xcbuild/opensource@fb.com). By participating, you are expected to honor this code.

## License

By contributing to xcbuild you agree that your contributions will be licensed under its BSD license.