

Getting the Source

This article talks about getting the WPILib sources, the general tools necessary for getting and building WPILib, and how to contribute.

Obtain the Code

WPILib is hosted in a series of git repos, available as part of the WPILibSuite organization on Github. You can see all available repositories at <https://github.com/wpilibsuite>. To view a repository in your web browser or to check out the repository, simply click the link for that repository to load the repository page. To check out the repository, click the Clone or download button on the repository page

There are a number of projects available for checkout. A brief description of each project is below.

- allwpilib - This project contains the majority of the actual WPILib robot code. This includes the HAL, WPILibJ, and WPILibC.
- build-tools - These are tools used by our Jenkins server for a few small tasks. Mainly, it is used to combine the 3 different desktop ncore platforms into a single zip and single jar for use on Windows, Mac, and Linux platforms. It also contains our teststand and driverstation, which allows us to automatically run our integration tests. Generally, this repository isn't used by an individual developer.
- cscore - This is a new camera access and streaming library for 2017. This provides much of the functionality for the replacement of the old WPILib CameraServer class.
- CANJaguar - These are the CANJaguar libraries that were removed from WPILib as part of the change to separate 3rd party provided libraries.
- design-docs - These are documents for proposed large scale changes\new features for WPILib.
- eclipse_plugins - This repository contains the C++ and Java FRC Eclipse Plugins projects. These use Maven targets to obtain the actual libraries, explained in the General Build Concepts section.
- gazebo_exporter - This is the SolidWorks plugin that allows teams to export a CAD model of their robot for use in the Gazebo simulator.
- netconsole-host - This is a roboRIO utility that forwards console messages out over the network
- nvision - This is the NIVision wrapper that was previously part of WPILib. We have moved to using OpenCV + cscore as the primary bundled vision tools for C++\Java WPILib, however teams may still use the NIVision libraries using the installers provided here.
- ncore - This is the implementation of NetworkTables 3.0. In this version, both Java and C++ use the same library, Java by way of a JNI interface also contained in this project.

Getting the Source

- opencv - This is a fork of the main OpenCV repo that contains changes necessary to automatically build and published our desired OpenCV versions for use with the roboRIO\cscore
- opencv-build-tools - These are tools for building the openCV artifacts that we publish on our Jenkins server
- OutlineViewer - This is the Outline Viewer tool that is shipped with the eclipse plugins.
- RobotBuilder - This the RobotBuilder project, also shipped with the eclipse plugins.
- sfx, sfxcv, sfxlib, sfxmeta, sfx-livewindow - These are all parts of the SmartDashboard 2.0 project.
- SmartDashboard - This is the original SmartDashboard project.
- SmartDashboard-Extensions - This repo contains SmartDashboard extensions
- styleguide - This contains the styleguide and style checking utilities for WPILibSuite projects
- toolchain-builder - This project contains the software for building the FRC C++ toolchain
- vendor-template - This is a template that vendors or teams can use to develop and distribute software to plug in to the WPILib Eclipse Plugins 3rd party library mechanisms
- wpilibtools - These are various tools that help teams publish to their robots. The most prominent tool in this repository is the Java Installer.
- wpilib-version-plugin - This is a Gradle (build system used by WPILib projects) plugin for determining where to pull dependencies from and push completed builds to, and what version numbers to tag things with
- wpiutil - This is a C++ library that contains common utility functions used by multiple other WPILibSuite libraries

Build Tools

Generally, WPILib uses Gradle for building and publishing. There is some variation by project, and the major projects have README files in their root directories explaining how to build them. Build details will be gone over in more detail in the General Build Concepts section.

As mentioned above, WPILib is hosted in git repositories, so some git tool is needed to clone them to your local machine.

WPILib builds are done, for most projects, by use of a build tool called Gradle. More information can be found about Gradle at their website, <https://gradle.org>. Gradle requires Java to be installed, and the JAVA_HOME environment variable to be correctly set. How to run Gradle is covered in General Build Concepts.

Many projects require the FRC ARM toolchains. Installing these is covered in ScreenSteps, in the [Installing Eclipse \(C++/Java\)](#) section.

Getting the Source

The eclipse plugins are built using a Gradle wrapper that calls Maven. Information about Maven is available at <https://maven.apache.org/>. More information about how the eclipse plugins retrieve different portions of the library is available in the General Build Concepts section.

How to Contribute

We (the WPILib developers) welcome changes from the community: a number of mentors and students from FRC teams have submitted a large number of changes to WPILib over the past few years. Generally, we want to be able to steer the direction of WPILib and review ideas for changes before we review code for the ideas. If you have an idea for a change, or a bug you'd like to fix, please create an issue in the appropriate WPILib project on GitHub. To submit an issue, you simply need an account on GitHub; you do not need to be a member of the WPILib project to submit an issue. At this point, we'll talk with you directly about further instruction on how to go about implementing and submitting this change. Before submitting a change, please read the following paragraphs about what types of changes we accept, and please ensure that your proposed changes fit within that description.

While we are very happy to accept contributions to the library, there are some constraints that we operate under that are a little different than doing code for a particular team or a few teams. If the code can fit those constraints, and it is something that we believe would be used by a large number of teams then we are likely to consider incorporating it. We do our best to make our code fit these guidelines, sometimes more successfully than others, but these are guiding principles that we operate with.

First it that the library must work for all the teams that use it. This is the first and most important requirement. It's easy to write code that works in a particular situation, with a particular robot configuration, for a particular team. But the software that we release has to work for everyone. That means thinking about the corner cases that 3000 teams will find and for the most part is reliable. We try hard to do this, usually pretty successfully.

We (the development team) have to guarantee that as the library changes, all the code continues to work. That means that any code added has to be pretty easily maintainable and understandable. We think we've done pretty well there, with some exceptions, and in those cases we try to go back and fix them. You can imagine that if someone from the community graciously contributes some complex piece of code, then a year or two later, for whatever reason stops participating, then the maintenance of that code falls upon the remainder of the development team to ensure that it continues to work.

We operate under a mandate from FIRST to have parity across all the languages. That means that if a significant feature is added to one language, we try to get it into all the languages. To do this we work very closely with NI, CTRE, and FIRST on all significant changes. What this means is that a

Getting the Source

developer contributing a change for one language is possibly obligated to implement it for all the languages. This doesn't necessarily apply to every new feature because everyone understands that each language is different and some library features just don't make sense to be implemented everywhere. For example, we have been upgrading the C++ library to take advantage of the, now available, C++11/14 language features to make the code more reliable and understandable. This clearly has no analog in the other languages. And it often, if it makes sense, needs to work in simulation.

Hopefully this explanation helps you understand what types of changes we are willing to accept to the library. Note that we will not be merging any new features changes during the build season; we will happily accept bugs, but new features will need to wait until after the 2016 competition season.

Additional detail is available in the Contributing.MD file of the AllWPILib project:
<https://github.com/wpilibsuite/allwpilib/blob/master/CONTRIBUTING.md>