WPILib Source

Last Updated: 01-08-2017

Table of Contents

Articles

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 ntcore 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
- nivision - 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.
- ntcore - 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.

- 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 sofwtare 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.

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 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

General Build Concepts

This article talks about general build concepts used across WPILib. This does not go into specifics for how to build individual projects, as that is covered the by README in the project root.

Gradle Based Projects

With the exception of the SmartDashboard 2.0 projects, all WPILibSuite projects use Gradle to build. This means that, for the most part, you do not need to install anything on your local machine other than the FRC Toolchains (for projects that build for ARM) and Maven (for the Eclipse Plugins project only!). The Gradle wrapper script (gradlew.bat for Windows, gradlew for Mac and Linux) takes care of downloading and running the correct version of Gradle for your system. These scripts are located in the root of the Gradle-based projects. Gradle is generally run from the console, with the following format:

```
./gradlew task1 task2 ...
```

task1 and task2 should be replaced with the names of the actual tasks that you want to run. Gradle accepts an arbitrary number of tasks, separated by spaces. On the command line, these tasks are not case sensitive. For all Gradle projects, there are 4 main tasks that you will be using:

- build - executes the build process. Gradle is generally smart enough to only rebuild the sections that have changed since the last build.
- clean - removes all created build targets. The next build will fully rebuild from scratch.
- publish - publishes all Maven targets to your local FRC maven repository. The repository's function is talked about in more detail in the next section. This task depends on parts of the build task, and will ensure that all relevant libraries have been built before publishing.
- tasks - a meta-task that lists all available tasks, along with their descriptions.

Local FRC Maven Repo

Because FRC projects are hosted in separate repositories, they push dependencies to each other in a Maven repository. This works in a tiered fashion:

1. dependences are resolved in the local FRC repository on your computer and if not available there then
2. dependencies are resolved against the WPI/FIRST Maven server.

Local Maven Repository

This repository is located in ~/releases/maven/<build channel> (where ~ refers to your home directory). Here, <build channel> is one of development, beta, stable, or release. These correspond to the 4 different stages of the WPILib release process. This default channel is development, which gets the latest changes that have been merged into the master branch of the project in question. To put artifacts in your local repository, run the publish task in the specific project. This also defaults to putting things in ~/releases/maven/development.

WPI/FIRST Maven Repository

For dependencies that can't be resolved on your machine (because you haven't published that particular project), the build system retrieves them from the FIRST Maven Repository. The specifics of what artifacts are available and the URL of the repository is covered in the Maven Artifacts article. The repository also has development, beta, stable, and release channels, and it uses the same channel as was used for local resolution.

What all of this means is that in order for your changes in a project to appear in a project that depends on it, you much publish the changes to your local maven repo. For example, the eclipse plugins depends on artifacts published by the allwpilib project. If you make some changes in allwpilib and would like to test them in the eclipse plugins, simply run the publish task in allwpilib. Then, in the eclipse plugins project, run the traditional build step, ./gradlew build, which will generate the plugins. This will prioritize the local maven repo over the official FRC server, pulling in your changed allwpilib builds and the latests development version of the rest of the utilities. If you want to go back to using the latest official development versions from the FRC server, you must delete the specific artifacts from your local repository.

Maven Artifacts

This article talks about the Maven repos, and the various artifacts we publish.

Public Maven Repositories

WPILib maintains 2 public Maven repositories: development and release. These repositories are available at http://first.wpi.edu/FRC/roborio/maven/<release channel>, where <release channel> is substituted by the repository you want to use. For developing WPILib, this defaults to development. However, you are free to use the release channel during the season if you want to use an alternative build system such as Maven or Gradle. However, note that we cannot provide official support for build issues if you take this route.

These repositories are browsable with your web browser: simply navigate to the URL of your chosen repository. Note that there are a few dependencies in the top level (first.wpi.edu/FRC/roborio/maven). These are there for legacy purposes and are not updated: you should not attempt to depend on them.

For each of our Maven artifacts, we use one of 2 versioning schemes, both based on semantic versioning (semver.org). The first scheme is unmodified semantic versioning (referred to as semver in the artifacts list below). The other versioning scheme is similar, but has the following structure: <season>.<mandatory update>.<optional update>. So, the first release of the 2017 season will have the version 2017.1.1. If an optional update was released, the new artifacts would have the version 2017.1.2. If a mandatory update was then released, it would have the version 2017.2.1. This versioning scheme will be referred to as wpilibver in the artifact listing below.

Artifacts

The following is a listing of the available Maven artifacts (build products). The format for each of these listings is as follows: package-id:artifact-id:versioning scheme - Description, including the project and repository it is published by. All artifacts are jar files, unless otherwise specified.

- edu.wpi.cscore.cpp:cscore:semver - This is the C/C++ version of the camera streaming and access library introduced in 2017. It is available here: https://github.com/wpilibsuite/cscore.
- edu.wpi.cscore.java:cscore:semver - This is the Java version of the camera streaming and access library introduced in 2017. It is available here: https://github.com/wpilibsuite/cscore.
- edu.wpi.first.wpilib:RobotBuilder:semver - This is the RobotBuilder executable installed with the eclipse plugins. It is available here: https://github.com/wpilibsuite/RobotBuilder.

- edu.wpi.first.wpilib:SmartDashboard:semver - This is the SmartDashboard version 1 executable installed with the eclipse plugins. It is available here: https://github.com/wpilibsuite/smartdashboard.
- edu.wpi.first.wpilib:athena-runtime:wpilibver - This is the base libraries for the wpilib HAL, including the HAL, NetworkTables, and WPIUtil. It is published as part of the allwpilib project, available here: https://github.com/wpilibsuite/allwpilib. This is a zip archive.
- edu.wpi.first.wpilib:hal:wpilibver - This is the base WPILib HAL library. It includes only the HAL and NI Libraries. It is published as part of the allwpilib project, available here: https://github.com/wpilibsuite/allwpilib. This is a zip archive.
- edu.wpi.first.wpilib:java-installer:semver - This is the Java Installer executable installed with the eclipse plugins. It is available here: https://github.com/wpilibsuite/java-installer.
- edu.wpi.first.wpilib.networktables:OutlineViewer:semver - This is the Outline Viewer executable installed with the eclipse plugins. It is available here: https://github.com/wpilibsuite/outlineviewer.
- edu.wpi.first.wpilib.networktables.cpp:NetworkTables:semver - These are the headers and libraries of the two published C++ NetworkTables libraries. There are several different classifiers available. The arm classifier contains the prebuilt libraries for running on the roboRIO. The desktop classifier contains the libraries for running on x86 and x64 Windows, Mac, and Linux. Finally, the sources classifier contains the sources for ntcore. All of these artifacts are zip files. It is available here: https://github.com/wpilibsuite/ntcore.
- edu.wpi.first.wpilib.networktables.java:NetworkTables:wpilibver - This is the Java implementation of NetworkTables. There are several classifiers available. The arm classifier contains the libraries for running on the roboRIO. The desktop classifier contains the libraries for running on x86 and x64 Windows, Mac, and Linux. The sources classifier contains the NetworkTables sources. Finally, the javadoc classifer contains the ntcore-specific javadoc. It is available here: https://github.com/wpilibsuite/ntcore.
- edu.wpi.first.wpilib:sfx:semver - This is the SmartDashboard 2.0 executable installed with the eclipse plugins. It is available here: https://github.com/wpilibsuite/sfx.
- edu.wpi.first.wpilib.simulation:simulation:wpilibver - This is installed by the simulation installer, and contains the non WPILibJ/C specific simulation components. This artifact is a zip file. It is published as part of the allwpilib project, available here: https://github.com/wpilibsuite/allwpilib.
- edu.wpi.first.wpilib:wpiutil:semver - This is a utility library of useful C++ classes used throughout WPILib C++ projects. It is published as part of the ntcore project currently, available here: https://github.com/wpilibsuite/ntcore. This artifact is a zip file.
- edu.wpi.first.wpilibc:athena:wpilibver - This is the WPILibC libraries. Dependencies are distributed via the edu.wpi.first.wpilib:athena-runtime package. It is published as part of the allwpilib project, available here: https://github.com/wpilibsuite/allwpilib. This is a zip archive.

- edu.wpi.first.wpilibj:athena-jni:wpilibver - This is the Java JNI bindings for the HAL used by WPILibJ. It depends on the athena-runtime package. It is published as part of the allwpilib project, available here: https://github.com/wpilibsuite/allwpilib.
- edu.wpi.first.wpilibj:athena:wpilibver - This is the main WPILibJ library. It depends on edu.wpi.first.wpilibj:athena-jni. It is published as part of the allwpilib project, available here: https://github.com/wpilibsuite/allwpilib.
- edu.wpi.first.wpilibj.simulation:SimDS:wpilibver - This is the driverstation used to control the Gazebo Simulator. It is published as part of the allwpilib project, available here: https://github.com/wpilibsuite/allwpilib.