Building and Contributing to WPILib (2018-19 Edition)

Brad Miller
Sam Carlberg

Agenda

- WPILib Overview
- What's New in 2018
- Development Trends and 2019 Plans
- 2019 Demo
- Building WPILib Components
- Desktop Tools
- WPILib Components
- GitHub Workflow
- How to Contribute

WPILib Mission

- Enable *FIRST* teams to focus on writing game-specific software rather than on hardware details – “raise the floor, don’t lower the ceiling”
  - Enable teams with limited programming knowledge or mentor experience to do as much as possible
  - Enable teams with intermediate programming knowledge to use powerful tools such as PID, GRIP, Dashboards, RobotBuilder to improve their robot performance
  - Enable teams with advanced programming knowledge to use the full power of the system
- Support the Kit of Parts control system hardware (e.g. controllers, sensors)
- Provide parity across all officially supported languages (C++, Java, LabView)
  - Enable teams to pick the language of their choice without worrying about supported features
- To this end, the library and associated tools need to be robust, reliable, maintainable, and understandable!

Who Owns All the Pieces

WPI and Team
C++ and Java libraries
C++ and Java NetworkTables
C++ and Java CameraServer
SmartDashboard
Shuffleboard
Outline Viewer
RobotBuilder
GRIP
C++ and Java toolchains

National Instruments
roboRIO image / libraries
FRC NetComm on roboRIO
LabVIEW libraries
FRC Driver Station
LabVIEW Dashboard
FPGA code and interface library
roboRIO Imaging Tool

FIRST
Field Mgmt System (FMS)
DS-Robot-FMS interfaces
Field Network Configuration
Robot Radio
Robot and Game Rules

WPILib Suite of Projects

Project	Primary Language	Purpose
WPILib C++ Library	C++	User library for C++ robot programs
WPILib Java Library	Java	User library for Java robot programs
WPILib HAL	C++	Low-level hardware library for C++ and Java
SmartDashboard	Java	Graphical dashboard
Shuffleboard	Java	Graphical dashboard
Outline Viewer	Java	NetworkTable viewer
RobotBuilder	Java	Code generator
Eclipse Plugins	Java	IDE & example programs
ntcore	C++	NetworkTables library for C++ and Java
cscore	C++	CameraServer library for C++ and Java
FRCSim		Robot program simulation environment
GRIP	Java	Graphical builder for image processing

What was New in 2018

- New extensible JavaFX dashboard: Shuffleboard
- WPILib:
  - TimedRobot base class
  - RobotDrive rewrite into DifferentialDrive et al
  - Components automatically publish data to NetworkTables for logging
- Java installed automatically to RIO
  - Thanks to Azul Systems provided Zulu JRE
- Much faster deployment (<5 seconds)
- NetworkTables:
  - Improved synchronization behavior
  - Added instance/entry handle-based interfaces
- OutlineViewer rewrite
- TCP netconsole (riolog)

Recent Trends in FRC Programming

- Emergence of alternative build systems and IDEs
  - GradleRIO
  - Use of IntelliJ for Java programmers
- Seeing downsides of current vendor library approach
  - Dependency management
  - Separate installers
- GitHub being used more widely
- Increasing popularity of Java
- Continuing demand for simulation and unit testing features

2019 Plans

Drumroll please...

2019 Plans

- New standard build system and IDE

Visual Studio Code

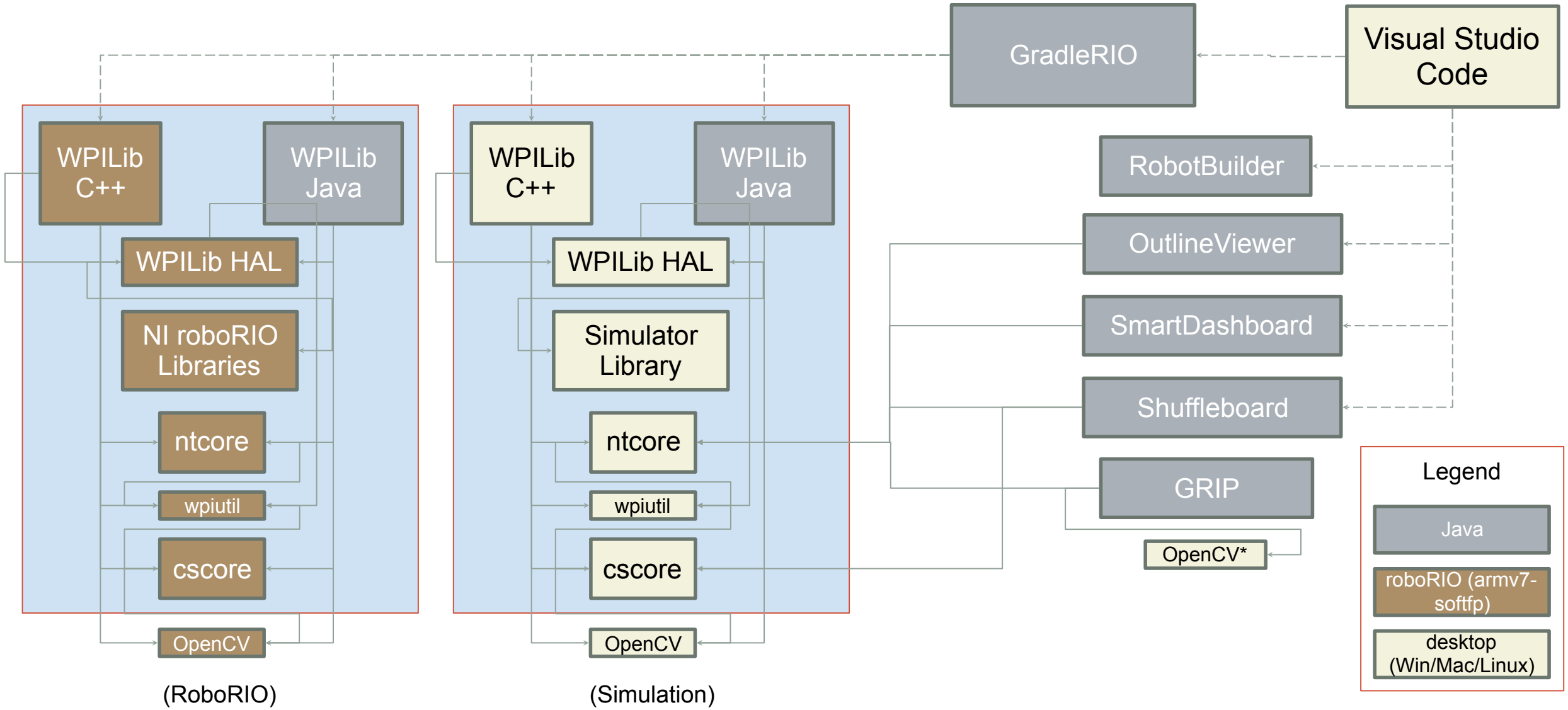
2019 Plans

- New standard build system and IDE
  - Visual Studio Code will be new standard IDE
  - GradleRIO build system (may be used independently of Visual Studio Code)
  - WPILib-maintained Maven repo for 3rd party libraries
- Java 11
  - OpenJDK 11 JRE for roboRIO
  - OpenJDK 11 JDK for desktop (installed with tools)
- Simplified install experience with single installer
  - Only separate download for offline install will be vscode itself (but we will automate it so it's easier than the old Java installer)
  - Correct version of OpenJDK installed automatically for FRC use
- Making other WPILib platform builds easier (desktop and coprocessor)
  - Gradle and cmake build tool options
  - WPILib, ntcore, cscore, wpiutil all in single source tree (remain separate libs/artifacts)
  - Simulator HAL used for non-RoboRIO builds so full WPILib can be built

More 2019 Plans

- Simulation usability improvements and documentation
- Integrate Jaci's Pathfinder library for path planning
- Software motion profile support
- Command framework improvements
  - State machines
  - Improved diagnostics
- RobotBuilder: more idiomatic output
- Shuffleboard
  - Performance improvements, more layout options, and more customizable
- Camera Server
  - Windows/Mac support for USB cameras, many other ideas in work
- Side-by-side multi-year installs (e.g. 2018 and 2019 can both be installed)
  - C++ toolchain will be "arm-frc2019-linux-gnueabi-" prefix for 2019
  - All tools installed to C:\Users\Public\FRC<year> on Windows
- C++ LLVM classes moving to "wpi" namespace

WPILib Suite Dependency Tree



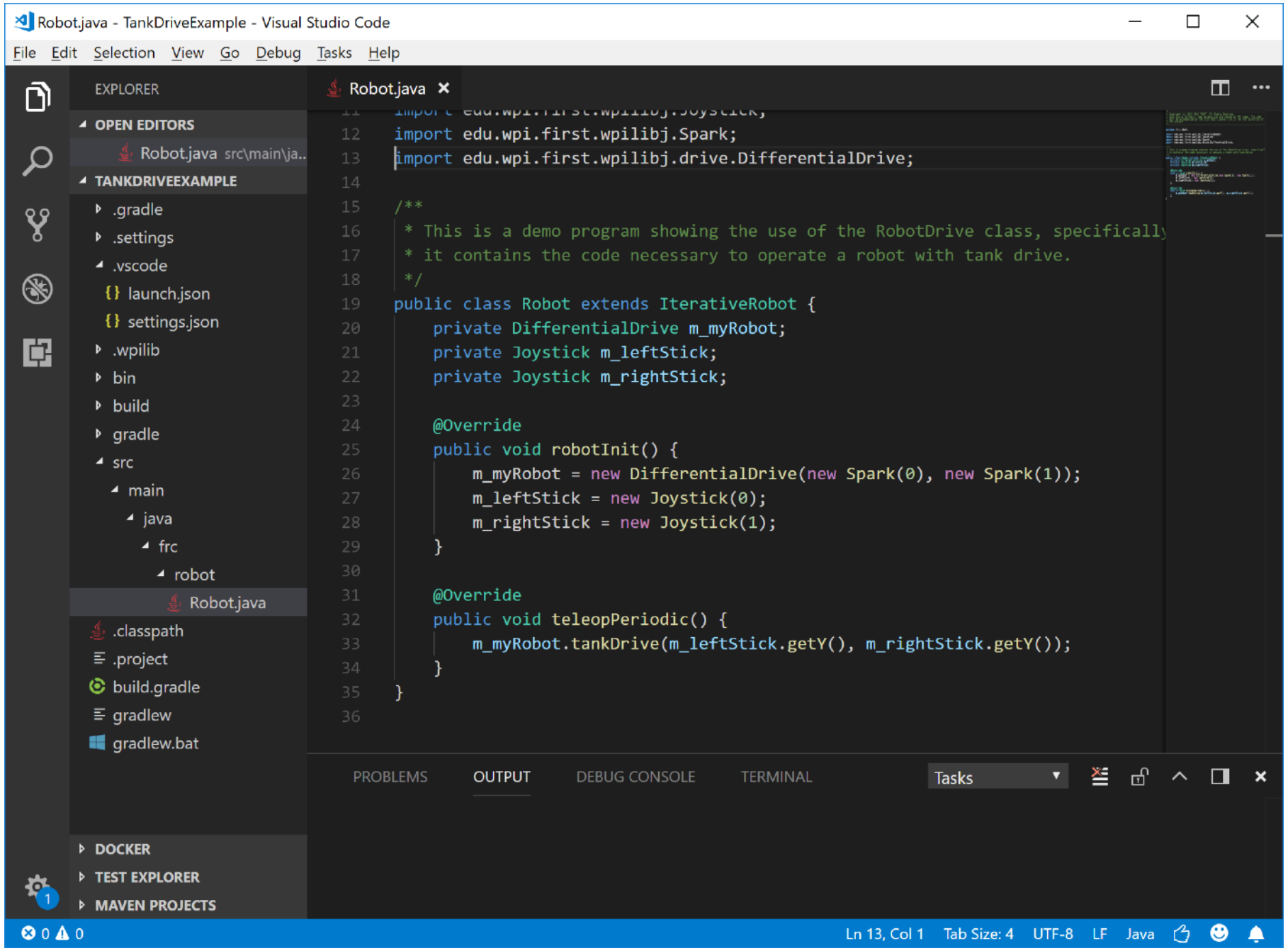
Visual Studio Code

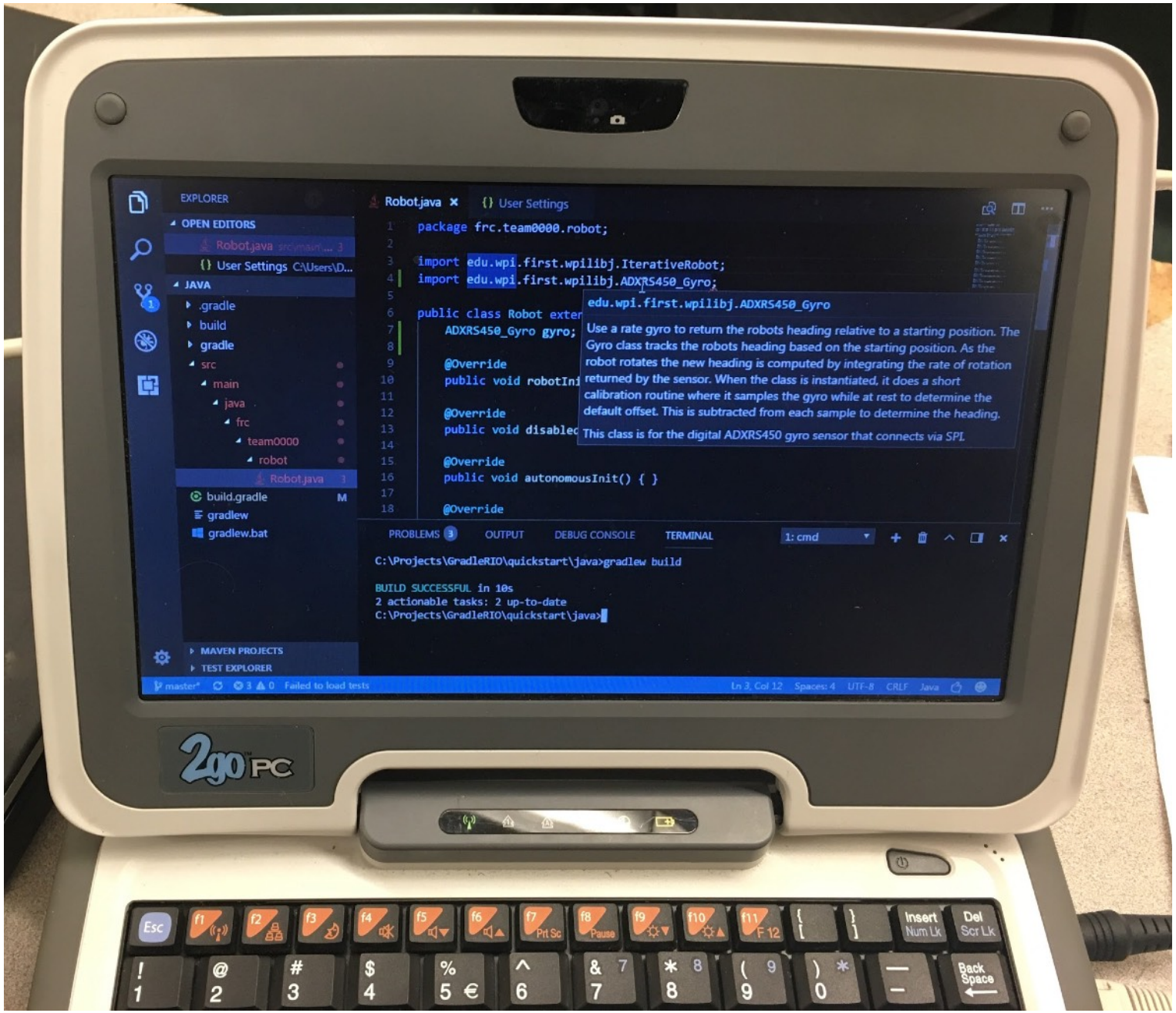
Visual Studio Code (new for 2019!)

- Free, Multi-Platform, Modern IDE
  - Not the same as Visual Studio!
- Enables teams to develop, deploy, and debug their code on the roboRIO
- Support for both C++ and Java (and other languages)
- Templates for generating classes
  - Subsystems
  - Commands
  - Starter robot code
  - Sample programs
- Allows interactive debugging
  - Early testing: more robust than Eclipse
- Many 3rd party plugins available to enhance development experience

Visual Studio Code: The “WPILIB” commands

- Type Ctrl-Shift-P (Or “View” Menu, “Command Palette”) to bring up command palette
- Start typing “wpilib” to bring up WPILib commands
- Some menu and keyboard shortcuts (e.g. Ctrl-Shift-B to deploy)


Robot.java - TankDriveExample - Visual Studio Code
File Edit Selection View Go Debug Tasks Help
EXPLORER
OPEN EDITORS
Robot.java src\main\ja..
TANKDRIVEEXAMPLE
.gradle
.settings
.vscode
launch.json
settings.json
.wpilib
bin
build
gradle
src
main
java
frc
robot
Robot.java
.classpath
.project
build.gradle
gradlew
gradlew.bat
DOCKER
TEST EXPLORER
MAVEN PROJECTS
Robot.java
import edu.wpi.first.wpilibj.Spark;
import edu.wpi.first.wpilibj.drive.DifferentialDrive;
/**
* This is a demo program showing the use of the RobotDrive class, specifically
* it contains the code necessary to operate a robot with tank drive.
*/
public class Robot extends IterativeRobot {
private DifferentialDrive m_myRobot;
private Joystick m_leftStick;
private Joystick m_rightStick;
@Override
public void robotInit() {
m_myRobot = new DifferentialDrive(new Spark(0), new Spark(1));
m_leftStick = new Joystick(0);
m_rightStick = new Joystick(1);
}
@Override
public void teleopPeriodic() {
m_myRobot.tankDrive(m_leftStick.getY(), m_rightStick.getY());
}
}
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Tasks
0 0
Ln 13, Col 1 Tab Size: 4 UTF-8 LF Java


EXPLORER
OPEN EDITORS
Robot.java
User Settings
JAVA
.gradle
build
gradle
src
main
java
frc
team0000
robot
Robot.java
build.gradle
gradlew
gradlew.bat
MAVEN PROJECTS
TEST EXPLORER
Robot.java
User Settings
package frc.team0000.robot;
import edu.wpi.first.wpilibj.IterativeRobot;
import edu.wpi.first.wpilibj.ADXRS450_Gyro;
public class Robot exter
ADXRS450_Gyro gyro;
@Override
public void robotIni
@Override
public void disablec
@Override
public void autonomousInit() { }
@Override
edu.wpi.first.wpilibj.ADXRS450_Gyro
Use a rate gyro to return the robots heading relative to a starting position. The Gyro class tracks the robots heading based on the starting position. As the robot rotates the new heading is computed by integrating the rate of rotation returned by the sensor. When the class is instantiated, it does a short calibration routine where it samples the gyro while at rest to determine the default offset. This is subtracted from each sample to determine the heading.
This class is for the digital ADXRS450 gyro sensor that connects via SPI.
PROBLEMS 3
OUTPUT
DEBUG CONSOLE
TERMINAL
1: cmd
C:\Projects\GradleRIO\quickstart\java>gradlew build
BUILD SUCCESSFUL in 10s
2 actionable tasks: 2 up-to-date
C:\Projects\GradleRIO\quickstart\java>
master*
Failed to load tests
Ln 3, Col 12
Spaces: 4
UTF-8
CRLF
Java
2go PC

Visual Studio Code Demo

- Demo!

Gradle Basics

- An open source build automation system
- Uses a Groovy-based format rather than a declarative project description (maven, make)
- Plugin based architecture
- Good multi-project support and scales to large projects
- Great support for Java and native languages (C++)
- Integrations for all the popular IDEs
- Most repos build with:
  - `./gradlew build`
- Desktop programs can usually be run with:
  - `./gradlew run`
- Robot projects will deploy to the robot with:
  - `./gradlew deploy`

GradleRIO

- Builds robot programs using Gradle
  - Replaces Ant (used in previous years in Eclipse)
- Easy to add additional libraries to your project
  - A standard set of maven dependencies will be made available
- Controlled by “build.gradle”

- Visual Studio Code knows how to read this file to find dependencies and do source lookup!

GradleRIO “build.gradle”

```
plugins {
  id "java"
  id "eclipse"
  id "idea"
  id "jaci.openrio.gradle.GradleRIO" version "2018.01.22"
}
```

Gradle plugin dependencies
“GradleRIO” part will change for official GradleRIO

```
def TEAM = 5333
def ROBOT_CLASS = "frc.robot.Robot"

if (project.hasProperty('teamNumber')) {
  TEAM = teamNumber.toInteger()
}
```

Configure team number; with Visual Studio Code, configured via Code preferences

```
def useDebug = false

if(project.hasProperty('debugMode')) {
  useDebug = true;
}
```

Debug support

GradleRIO “build.gradle”

```
deploy {
  targets {
    target("roborio", jaci.openrio.gradle.frc.RoboRIO) {
      team = TEAM
    }
  }
  artifacts {
    artifact('frcJava', jaci.openrio.gradle.frc.FRCJavaArtifact) {
      targets << "roborio"
      debug = useDebug
    }
  }
}
```

Defines targets (RoboRIO) and artifacts (deployable files)

```
dependencies {
  compile wpilib()
  compile ctre()
  compile navx()
  compile openrio.powerup.matchData()
}
```

Defines dependencies. In this case, WPILib (+ friends), CTRE Phoenix (Talon SRX) and NavX.

GradleRIO “build.gradle”

```
jar {
  from configurations.compile.collect { it.isDirectory() ? it : zipTree(it) }
  manifest jaci.openrio.gradle.GradleRIOPlugin.javaManifest(ROBOT_CLASS)
}
```

Setting up the Jar File. In this case, adding all libraries into the main jar ('fat jar') in order to make them all available at runtime. Also adding the manifest so WPILib knows where to look for our Robot Class.

```
task wrapper(type: Wrapper) {
  gradleVersion = '4.4'
}
```

Gradle wrapper version to use.
Don’t change this.

Building WPILib Components

Dependencies and Configuring Local Changes

- Build outputs are shared between repositories as Maven dependencies
- Global repositories for our builds, with 2 channels
  - Officially-released builds: http://first.wpi.edu/FRC/roborio/maven/release
  - Bleeding edge: http://first.wpi.edu/FRC/roborio/maven/development
- Local builds publish to ~/releases/maven/development
  - `./gradlew publish`
  - GradleRIO will immediately start using your locally built version (future)
- Dependencies are suffixed with target platform
  - There is a “-all” that provides combined native and/or JNI libraries
- Versioning is SemVer based
  - Year-style: year.required_update.optional_update (2018.3.2)
  - Standard-style: major.minor.revision (3.1.7)

Creating Vendor Libraries

- GradleRIO supports integration of vendor libraries
  - When the necessary files are installed in the $HOME/wpilib/<year>/user directory, GradleRIO will automatically make them available to user programs and take care of copying the libraries to the robot
  - This makes the vendor library transparently available to the user
- The CANTalon and NavX libraries use the vendor approach for 2017+
- There is a project template on GitHub (wpilibsuite/vendor-template) for creating new vendor libraries
  - Note if the library is a C/C++ library, it's still necessary to write appropriate JNI and Java wrapper code, although the wpiutil library has some classes that make writing wrappers easier

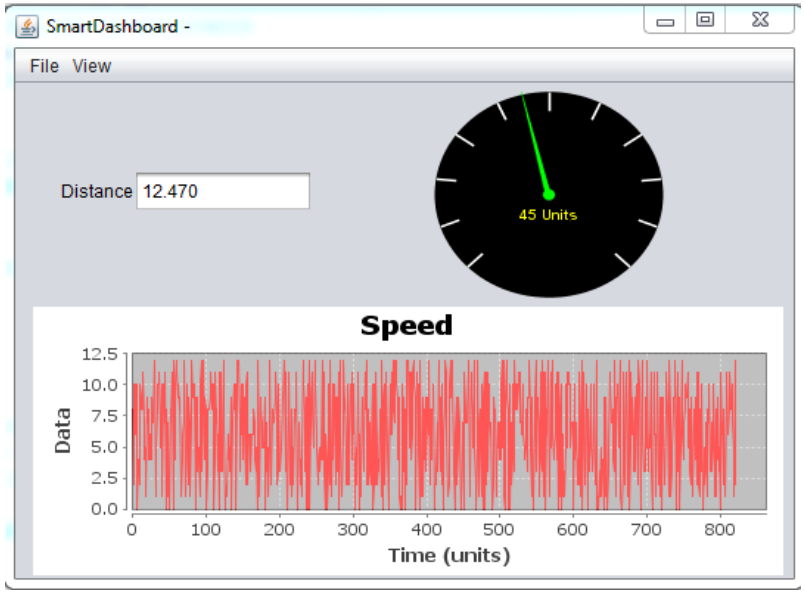
Desktop tools

- SmartDashboard
- Shuffleboard
- OutlineViewer
- RobotBuilder
- GRIP

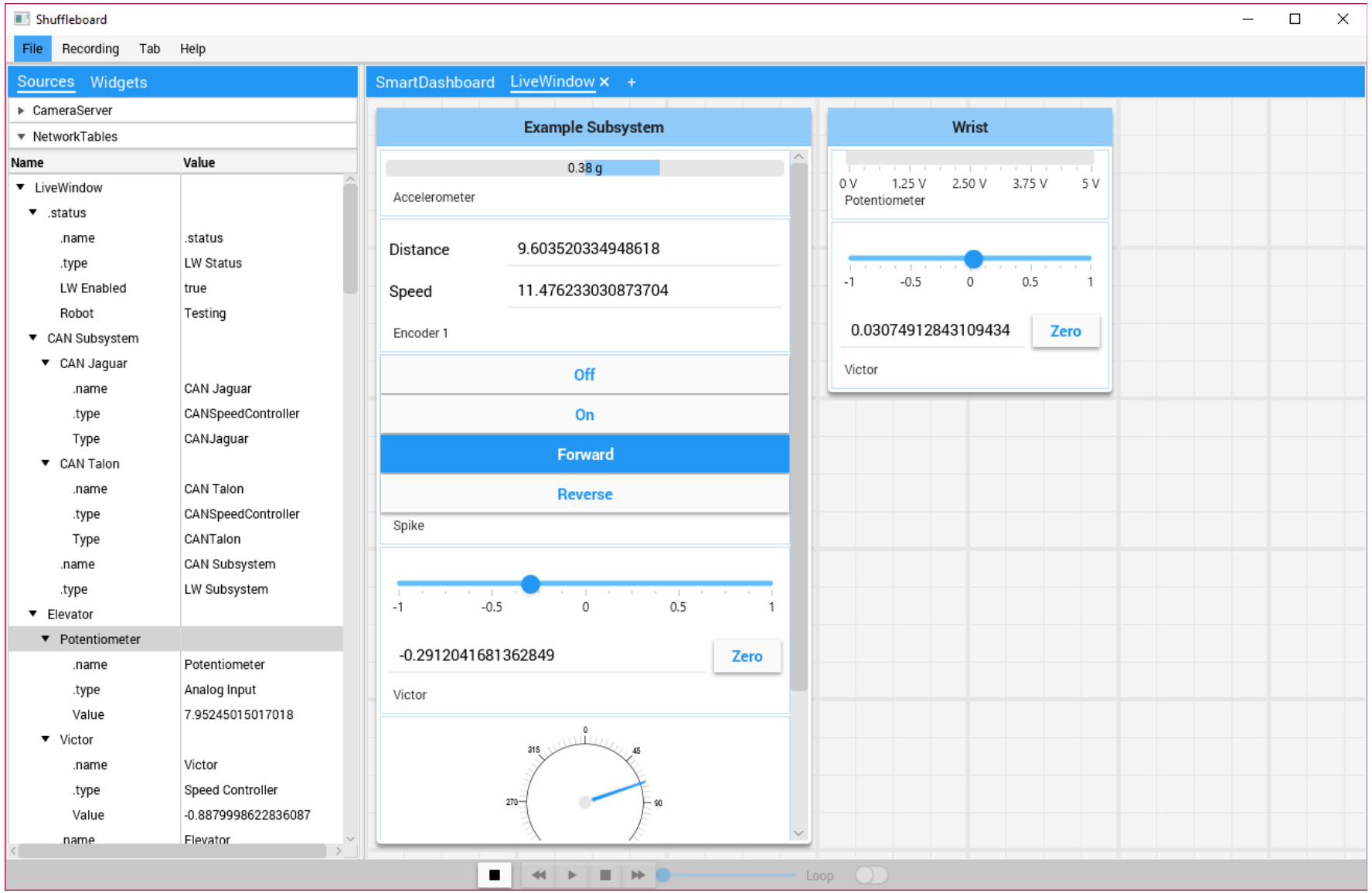
Synergy in the tools - the result is greater than the sum of the parts

SmartDashboard

- Java GUI to NetworkTables that displays robot data in real time
- Fields displayed as text fields or more elaborately in graphs, dials, etc.
- Displays robot program state
  (e.g., executing commands and subsystem status)
- Displays buttons for setting variables on your robot
- Allows choosing startup options on the dashboard for the robot program



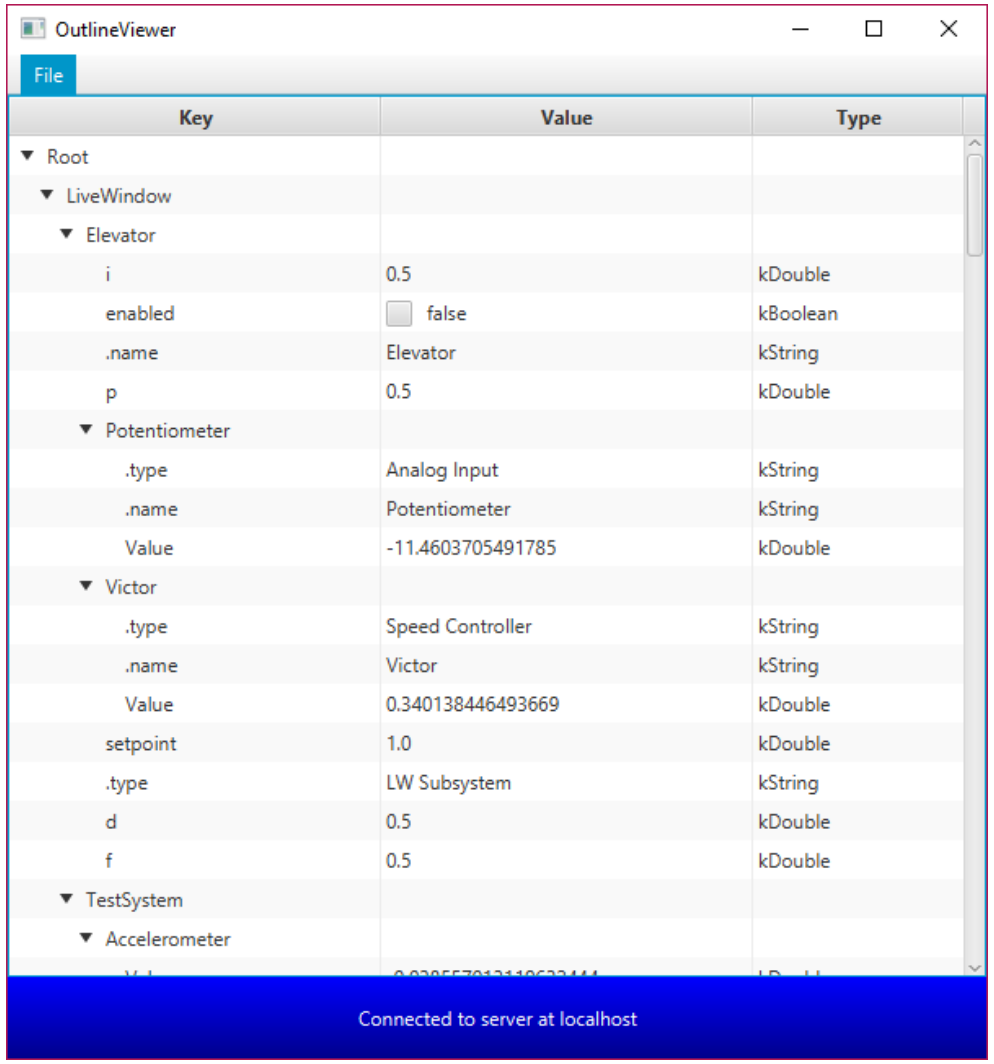
Shuffleboard



Shuffleboard

- JavaFX GUI with themes
- Supports data from arbitrary sources (not just NetworkTables)
- Can record and play back data
- Displays and controls robot state
- Replaces SFX, eventually SmartDashboard
- Extensible and customizable
- Tabbed user interface where tabs can have a filter to select data that should be displayed there.
  - Think debug display tab and driver display tab that load from /SmartDashboard/debug/ and /SmartDashboard/driver/

OutlineViewer

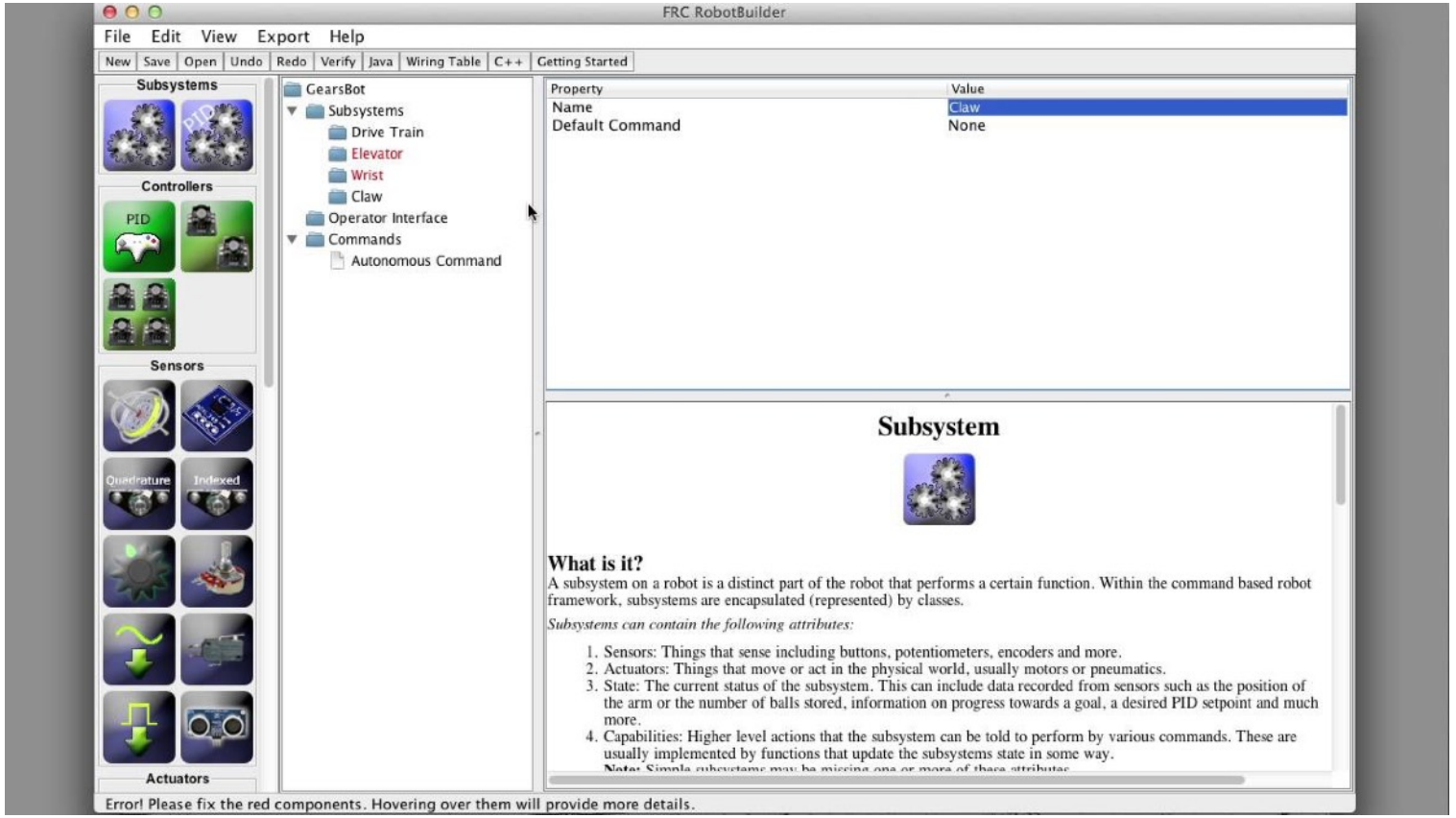


OutlineViewer

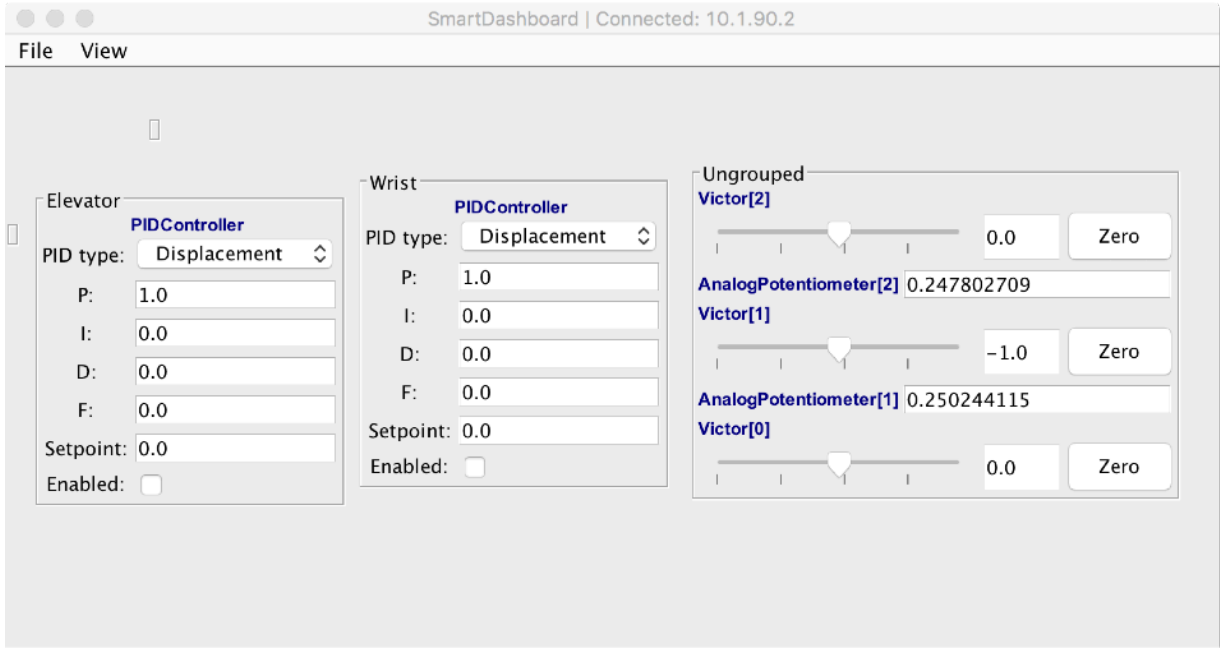
- NetworkTables client that displays every key and value
- Values can be edited and keys can be created and deleted
- At startup, prompts for address to connect to (use your team number)
- Displays connection status (color-coded)
- Can dump/load the current state of NetworkTables to a file and reload it later

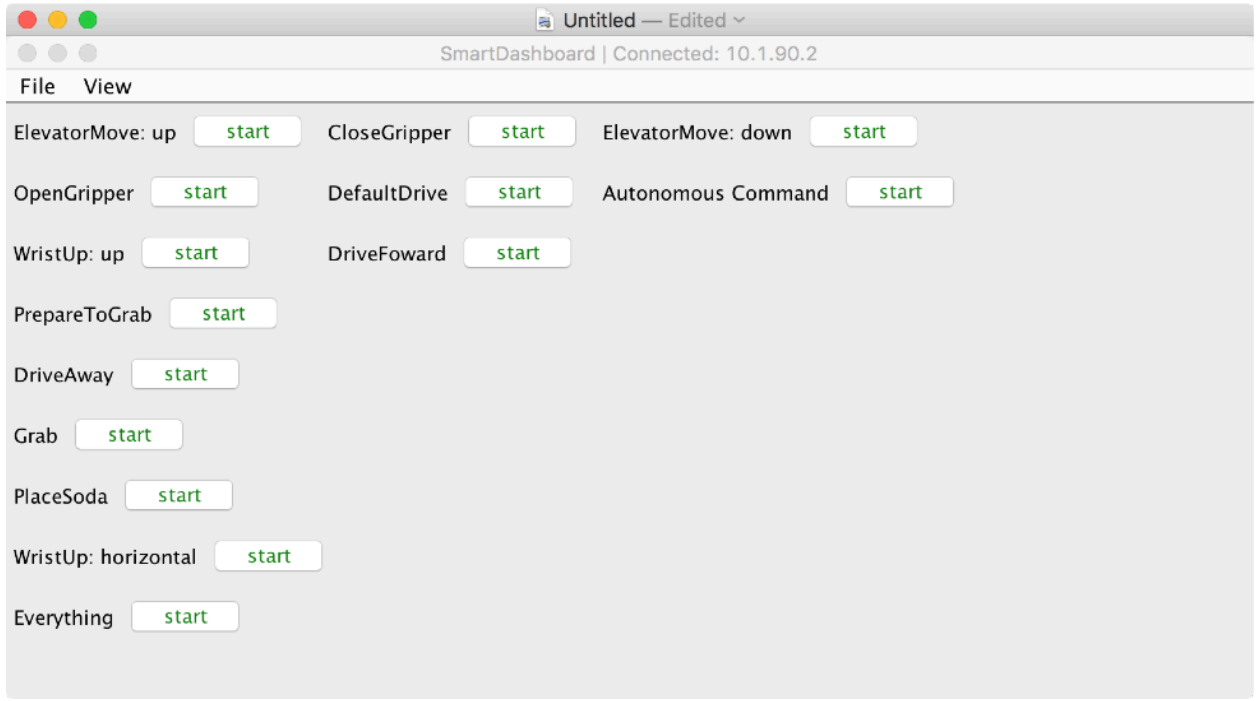
RobotBuilder

- Create a command based framework for your robot program



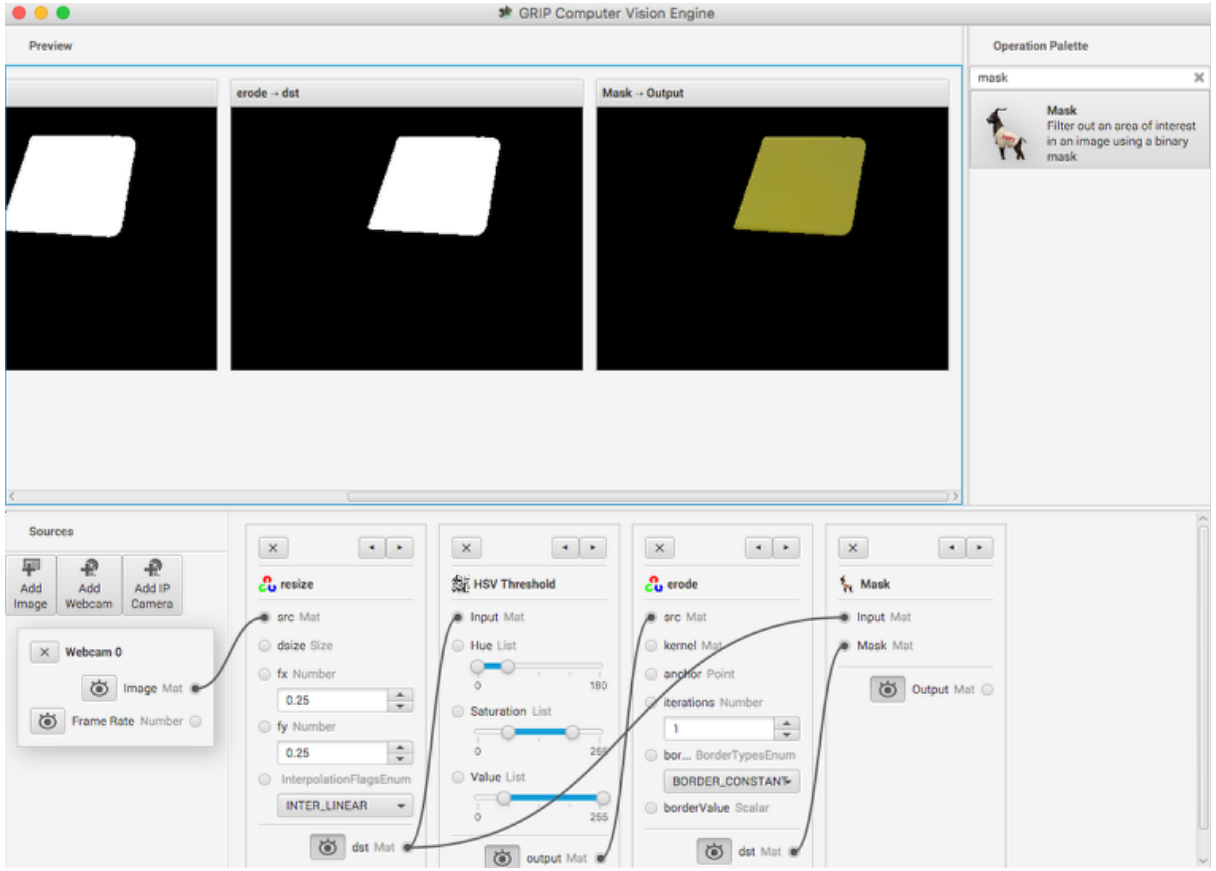
RobotBuilder automated test code





GRIP

- Graphically Represented Image Processing engine
- Allows rapidly prototyping and deploying computer vision algorithms for robotics applications



Future Plans for Desktop Tools

- Shuffleboard
  - Convert recording files to CSV
  - Performance improvements (graphs etc.)
  - More layout options
  - Tighter WPILib integration
- GRIP
  - Dark theme
  - Custom python operations
  - Automatic file backup
  - GPU acceleration
  - Limelight camera support

Cross Platform “Core” Libraries

wpiutil
ntcore
cscore

Cross Platform “Core” Library Dependencies

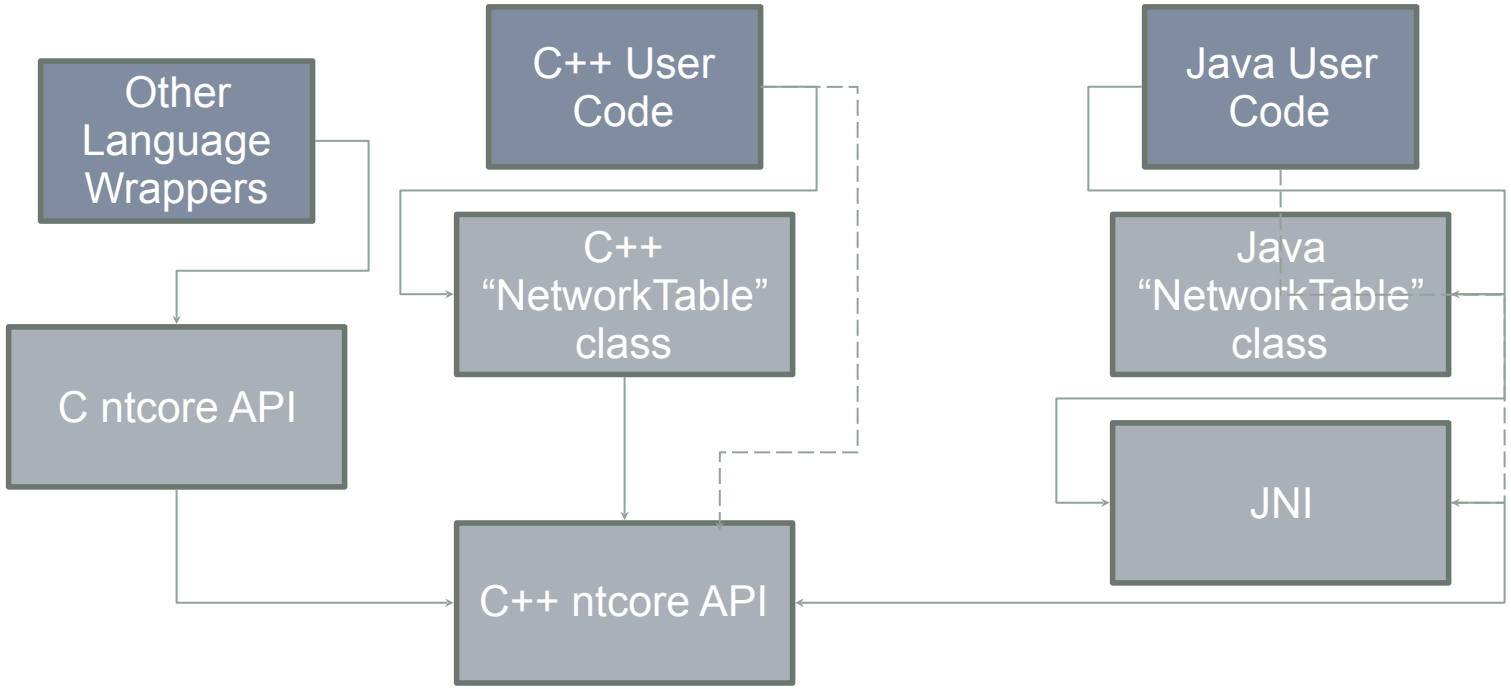
- Compiler toolchains
  - Platform native C++ compiler/linker
    - Windows: MSVC, Mac: clang, Linux: GCC
    - C++11 support is required as ntcore extensively uses C++11 features including std::thread
  - Platform native JDK (if building Java library)
  - FRC ARM toolchain (if building roboRIO library)
- Both ntcore and cscore depend on wpiutil
- The cscore library also depends on the WPI build of OpenCV

WPI Utility Library (wpiutil)

- Cross-platform C++ library used by C++ WPILib, ntcore, and cscore to provide useful high performance, low overhead C++ classes
- Classes include StringRef, StringMap, SmallVector
- Many of these are copied from LLVM (some with minor tweaks)
  - Will be moved from “llvm” namespace to “wpi” namespace for 2019 to avoid conflicts with system libraries

NetworkTables Core Library (ntcore)

- Implements NetworkTables 3.0 client and server
- High performance, cross-platform C++ library
- JNI wrappers for Java use
- Provides C++ and Java “NetworkTable” and “ITable” classes



CameraServer Core Library (cscore)

- High-performance access to USB and HTTP cameras
- HTTP streaming camera server
- OpenCV access to camera streams
  - OpenCV outputs can also be streamed via HTTP
- Designed for robust (e.g. physical camera disconnections) and low-overhead operation
  - HTTP streaming of MJPG-capable USB camera typically requires <5% CPU
- Cross-platform C++ implementation with C interfaces and JNI wrappers
- Note: cscore does not provide the “CameraServer” class (instead, this is part of WPILib) to avoid a NetworkTables dependency

Contributing & GitHub

GitHub Workflow

- https://github.com/wpilibsuite/allwpilib
- Standard GitHub workflow - branch, commit, pull request
- Use Jenkins, Travis CI, AppVeyor to run tests
  - Travis and AppVeyor run automatically
  - Jenkins waits for permission
  - Runs unit and integration tests
- After merge
  - Jenkins checks out and builds changes
  - Downstream dependencies are rebuilt
  - Build artifacts published to the development channel
- Build promotion
  - Development: each change merged to master
  - Release: alpha, beta, rc, release

Our Policy on WPILib Suite Changes

- Everything in the library has to work for the 3000+ teams that will use it
- We need to be able to support submitted changes even if the author loses interest
- Tool suite changes must be generally useful for a broad group of teams
- Changes in one language usually need corresponding changes in the other language
- Substantial changes often need to have corresponding LV changes
- Library changes should have tests
- Code should be well documented, often with ScreenSteps

What Should You Contribute

- Bug reports and fixes!
- Improvements (large and small) to existing classes
  - 2016 example: configurable SPI address for a sensor, making it more general purpose
  - 2017 example: HAL overhaul, PWMSpeedController subclass
  - 2018 example: TimedRobot, RobotDrive overhaul, SpeedControllerGroup
- Generic reusable library components
  - 2016 example: Filter and LinearDigitalFilter classes
  - 2017 example: XBoxController class
  - 2018 example: low-fi sim support

- Good rule of thumb: we'll almost always gratefully accept bug fixes. It's best to ask about new features before making a large time investment
- Particularly like features that make it easier to help teams with less experience be more successful

What Should You Not Contribute

- Game-specific code – the game changes each year!
- Team-specific code – WPILib needs to be generic
- New sensors
  - Logistics of this is tricky—at the very least, we need to be able to test it in hardware!
  - If you work for a company and want to get a sensor into the KOP, please contact FIRST
  - We're looking at establishing a "white pages" of unofficial (not officially supported) user-contributed modules
  - If you just want to add support for a new sensor, consider making it available as a vendor library instead!
- Major restructuring of library classes or major rewrites
  - Ideas accepted, but please talk to us before putting in a lot of work
  - Backwards compatibility (and bitrot) are real concerns for teams

WPILib Suite Development Schedule

Time	What Happens
Championship	Pare down list of improvements for next season
Mid May	Working full speed on updates
September	Beta team selection (~30 teams per language)
Late September – October	Beta testing begins New projects wind down
December	Beta testing ends, code freeze, and only bug fixes
Late December	Final builds and kits are created
At Kickoff	Game-specific code and field images are posted or merged

Questions?

Backup

Cross Platform Library Code Organization

Location	Purpose
doc/	Standalone documentation (e.g. protocols)
src/main/java/	Java wrapper source code
src/test/java/	Java automated unit tests
src/main/native/include/	External-facing (API) C and C++ headers
src/main/native/cpp/	C++ source code
src/main/native/cpp/jni/	JNI bridge C++ source code
examples/	C++ standalone tests (example programs, not automated)
src/test/native/cpp/	C++ automated unit tests (uses gtest/gmock)
gradle/	Build support files
src/dev/java	Java development executable (Use `gradlew run` to run)
src/dev/native/cpp	C++ development executable (Use `gradlew runCpp` to run)