SystemCore Software

Presenters

- **Kevin O’Connor**
  - Senior Robotics Engineer, *FIRST* Robotics Competition
- **Danny Diaz**
  - Senior Engineering Manager, *FIRST* Tech Challenge
- **Peter Johnson**
  - WPILib Core Developer
- **Amanda Bessette**
  - Associate Director Robotics Resource Center, WPI

Goals

Unify the programs

- Software experience should be largely the same across FTC and FRC, deviating only where absolutely necessary

Lower the barrier to entry

- Software must be easier to install, configure devices and get started

Leverage new capabilities

- Utilize the performance of new controller to reduce common issues such as loop overrun, memory constraints, and unit errors

Maintain the ceiling

- Avoid adding any significant barriers to high performing teams

Philosophy

Software for both programs, not make WPILib work for FTC

- Existing WPILib architecture and infrastructure will still serve as the base of the new libraries

Make the right decisions for the future

- In some cases this may mean a little more transition pain

Visible development and testing processes

- Folks can follow along and see what's coming, even if they may not have hardware yet

What are we sharing today?

Focused on software

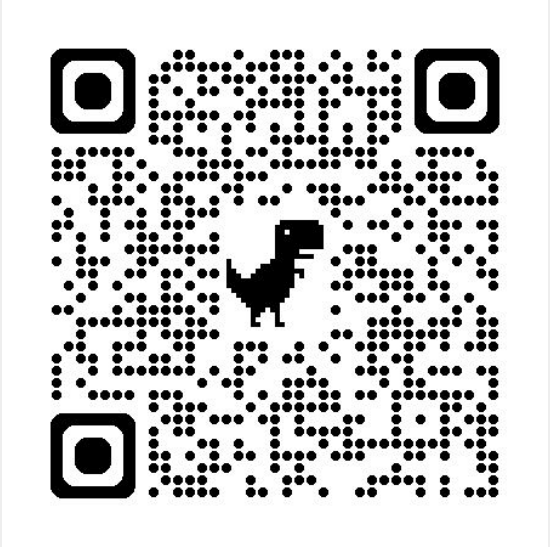
- Latest information on System Core hardware can be found in March 19 Blog, more info will be released as we get closer to team testing

Subject to change

- This information is where we think we're headed today. Feedback, roadblocks, and other factors can and will change some of this as we proceed through development

High Level

- We're going to talk more about concepts and less about syntax. Don't expect to walk out ready to program your 2027 robot.



WPILib Mission - Fundamentally Unchanged for 2027+

- Enable *FIRST* teams to focus on writing game-specific software rather than on hardware details
- Make new robotics technologies more approachable to teams: “raise the floor, don’t lower the ceiling”
  - Enable teams with limited programming knowledge or mentor experience to be more successful
  - Enable teams with intermediate programming knowledge to use powerful tools 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
  - 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!

WPI Lib

Contributing to WPILib - It Takes a Village

- WPILib is much more than just the robot library; the team experience includes everything from documentation to the installer and various tools
- We're a volunteer open source project; YOU can help make a difference and have a broad impact across thousands of FRC and FTC teams
- You don't need to be a C++ or Java expert to help; we're also looking for help in these areas:
  - VS Code extension, e.g. for Python integration, RobotBuilder 2.0 (TypeScript)
  - Installer improvements (C# GUI)
  - Controller-hosted tools (modern TS/JS web dev)
  - Documentation!
- We mainly collab via GitHub, but you can also find us in community spaces
  - GitHub project boards (we have a 2027 project) and allwpilib issues are good places to start
  - Reach out (e.g. comment on an issue) to coordinate before diving in

SystemCore Key Software Benefits

Powerful 64-bit processor (Raspberry Pi CM5 based) with 4 GB of RAM enables:

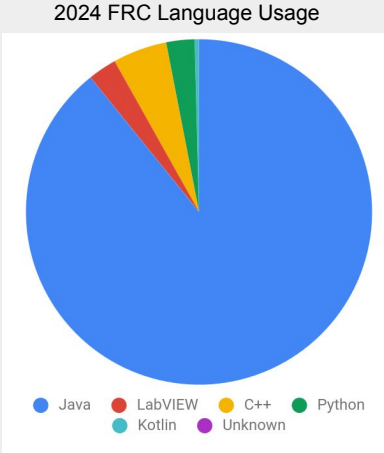
- Controller hosted development and other web-based tools
- Fewer loop overruns and much reduced probability of out-of-memory issues
- Performant integrated vision processing

Topics

- **Supported programming languages**
- **Desktop development**
- **Controller Hosted development**
- **Blockly**
- **Frameworks and OpModes**
- **Telemetry**
- **Dashboards and tools**
- **“Smart Motor” API**
- **Miscellania**

2027 Official WPILib Language Options

- 4 officially supported WPILib languages: C++, Java, Python, and Blockly
  - Roughly equivalent feature sets
  - Modern language versions (C++20, Java 21, Python 3.12+)
- Primary driver of language selection is network effect
  - School classes (AP CS Java)
  - Mentor experience
  - What other teams in your area use
  - Online support base (teams worldwide)
  - What other teams at competitions use
- In 2024, 89% of FRC teams used Java
  - Python was at ~3% in its first official year (about the same as LabVIEW); expecting to see future growth as an officially supported language and via Blockly onramp path



WPI Lib

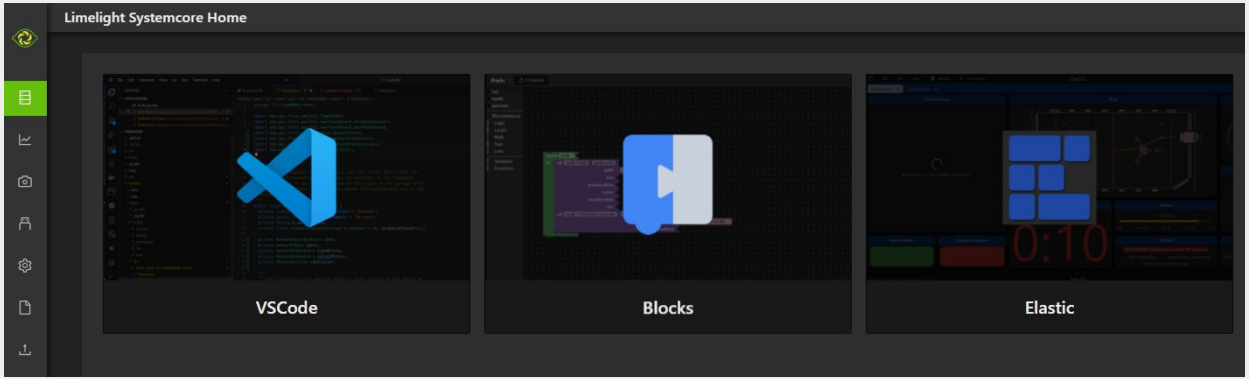
Desktop Development

- Similar to current WPILib
- Java, Python, and C++
- All languages use Visual Studio Code for IDE
  - More integrated Python experience
- Baseline: continue to use GradleRIO for Java and C++ build/deploy system
  - Open to C++ transitioning to another build system, depending on community contributions
- Desktop simulation support
- Want to further streamline download+install experience while maintaining offline install capability



Controller Hosted Development

- **Java, Python, and Blockly**
  - C++ not supported due to expected poor experience
- **Java and Python use VSCode (or derivative) from browser**
  - Currently expect single user access, still investigating possibilities for multi-user
  - Currently planning to support storage of multiple “Workspaces” with some sort of “Deploy” button to determine currently running code
- **Likely no simulation support**



WPI Lib

Blockly

- **Graphical coding experience using Google Blockly**
- **Generates WPILib Python code**
  - Users can view the code associated with Blockly in “real-time”
  - Can “generate” into a Python code project but can’t turn Python back into Blockly
- **Aim to balance “limiting the ceiling” with simplifying the experience to make things simpler**
- **Expecting to support splitting code into multiple subsections to make editing more manageable**

WPILib Frameworks

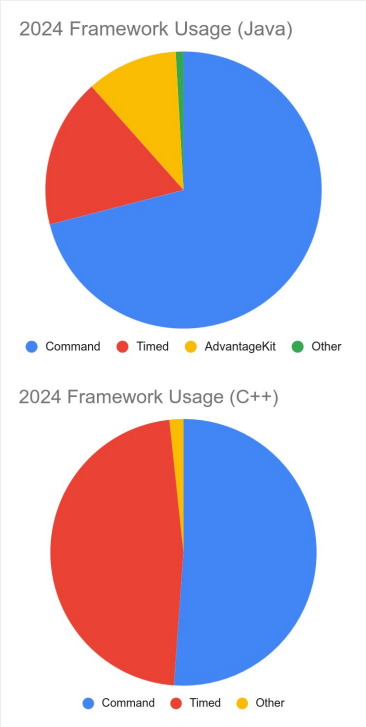
WPILib plans to offer two primary frameworks for robot code structure:

- Periodic + Linear Opmodes
  - Periodic: Runs a function every 20 ms
  - Linear: Runs a function; looping handled by user
  - Easy to get started with, but unwieldy for complex programs
- Command Based
  - Genesis was team approaches to adding structure
  - Structures code into “subsystems” and “commands”
  - Execution model is cooperative multitasking
  - Enables more complex programs, but has steeper learning curve

In 2024, 63% of all FRC teams used command-based Java for their competition robot programs

A team-created Java framework called AdvantageKit became popular in 2024. It’s important to note that AdvantageScope is a generic debugging/visualization tool and does not require the use of AdvantageKit.

For 2027, we will extend the existing frameworks with opmodes support



Opmodes - Background

Purpose: Enable operator to choose what code to run for a particular robot mode (e.g. auto or teleop)

FRC today

- Singleton Robot class defines actuators/sensors
- Explicitly defined modes with lifecycle methods e.g. Robot.teleopInit(), autoPeriodic()
- DS has enable/disable and mode selector (teleop/auto/test/practice)
- Many teams select what autonomous code to run using SendableChooser or similar via a dashboard

FTC today

- Hardware map defines actuators/sensors
- Teams define multiple teleop and periodic opmodes using annotated classes
- DS provides opmode selector; if auto opmode selected, teleop opmode can be preselected to run after auto completes
- OpModes have a blocking initialization step that allows actuator movement

Opmodes - Plan

General operation

- Singleton Robot class defines actuators/sensors and provides robotPeriodic, disabledPeriodic
- Opmodes can be named and grouped
- DS has enable/disable, mode selector (teleop/auto/test/match), and opmode selector(s) filtered by robot mode
- No separate opmode initialization step; robot is either enabled or disabled

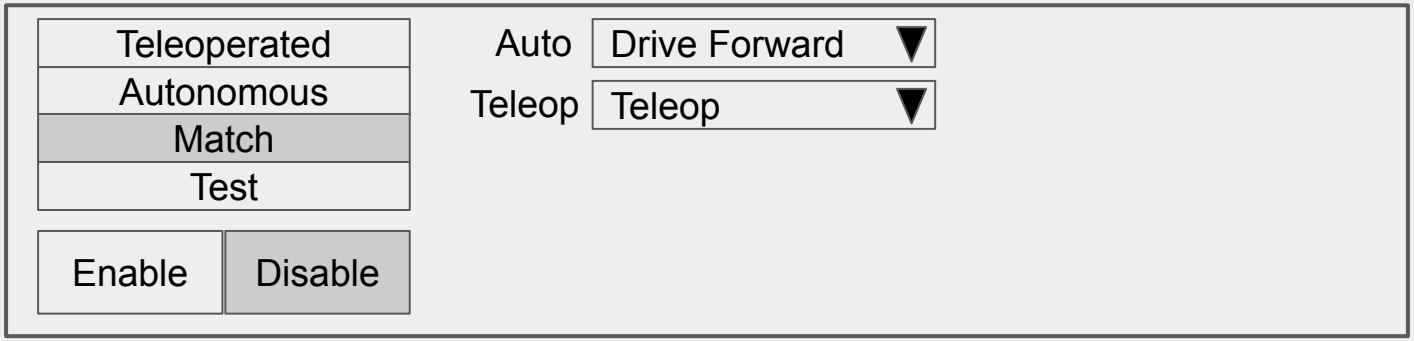
Periodic/Linear structure

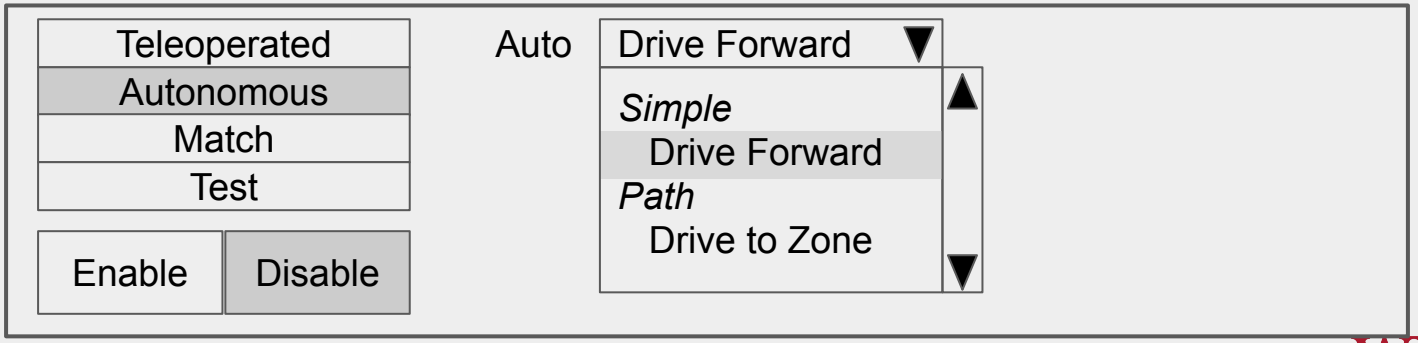
- Opmodes are defined via annotated classes; Robot object is passed to the constructor
- Only one opmode class is constructed/active at a time; in a match, the teleop opmode object is not constructed until after close() is called on the auto opmode object (at the end of auto)

Command-based structure

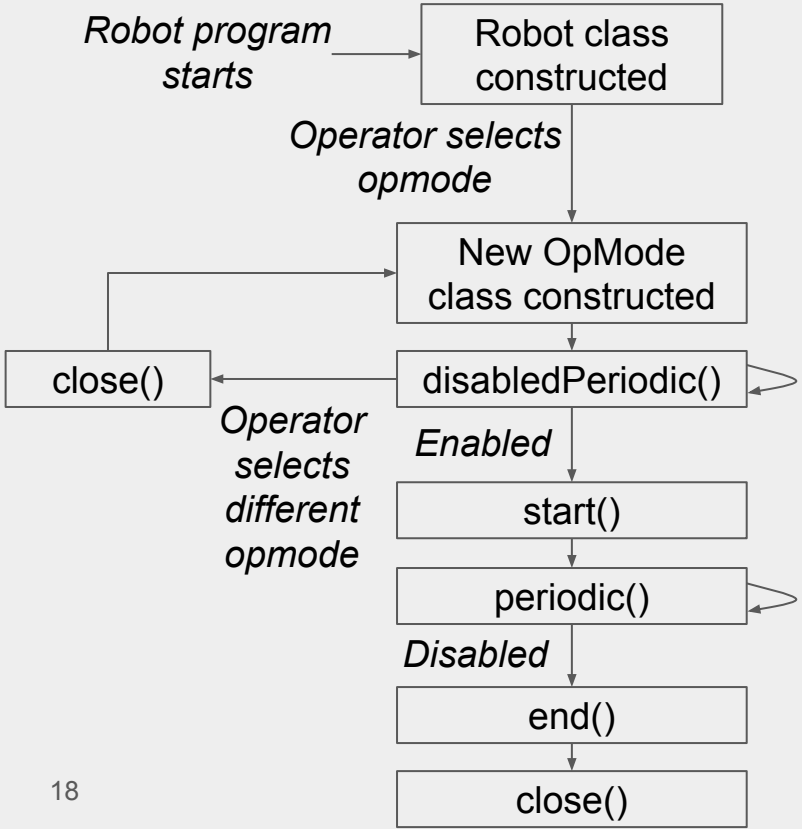
- Opmodes are defined via factory functions and provide selected, disabled, running triggers
- Commands/triggers are tied to particular opmodes via these triggers or sugaring
- Subsystem periodic functions always called periodically (in disabled and in all opmodes)
- Note: it is not possible to mix command-based and periodic/linear opmodes in a single robot project

Driver Station Concept





Periodic Opmode (Linear is similar)




```
@Autonomous(name="My Auto", group="Drive")
public class MyAuto extends PeriodicOpMode {
  private final Robot robot;
  private final Timer timer = new Timer();

  public MyAuto(Robot robot) {
    this.robot = robot;
  }

  @Override
  public void start() {
    timer.start();
  }

  @Override
  public void periodic() {
    if (!timer.hasElapsed(2.0)) {
      robot.drivetrain.Drive(...);
    }
  }
}
```

Command-Based Opmodes

```
// A simple autonomous opmode
CommandOpModes.autonomous(“Simple Auto”).running.whileTrue(Autos.simpleAuto(this));
```

```
// A complex autonomous opmode that loads a path when selected in the DS (while still disabled)
String pathName = “My path”;
var opmode = CommandOpModes.autonomous(pathName, “Paths”);
opmode.selected.onTrue(Commands.runOnce(() -> Paths.loadPath(pathName)));
opmode.running.whileTrue(Autos.followPath(this, pathName));
```

```
// A teleop opmode with joystick and button controls
var opmode = CommandOpModes.teleoperated(“teleop”);
var driverController = new CommandXboxController(1);

// Deploy the intake with the X button
driverController.x().whileRunning(opmode).onTrue(intake.intakeCommand());

// Control the drive with split-stick arcade controls
drive.setDefaultCommand(opmode, drive.arcadeDriveCommand(...));
```

Telemetry - Background

Purpose: Live display and data logging (for offline analysis) of sensors and other robot data

FRC today

- Tree structured, native data; 1000+ data values becoming common
- Dashboards: users choose view of subset of values and widgets to display them; only one dashboard (Shuffleboard) has code-driven layout
- SmartDashboard, Shuffleboard: mostly imperative, some callback (putData/Sendable), bidirectional; only complex data type support is Sendable (no Struct/Protobuf)
- Epilogue: annotation, with compile-time generation
- NetworkTables and DataLog: low level, relatively complex APIs
- Third-party: AdvantageKit (annotation & imperative), DogLog (imperative)

FTC today

- Telemetry: imperative, flat structure, string-based, dashboard display control

Telemetry - Plan

Annotation: Epilogue

- Change from codegen to introspection for user-friendliness

Imperative: New Telemetry API (replaces SmartDashboard and Shuffleboard APIs)

- Tree structure; TelemetryTable provides subtree view (ala NetworkTable)
- Telemetry static class for root table
- Output only (inputs use a separate Tunable API)
- Primary entry point: log(name, value) method, overloaded for all data types
- Supports setting per-entry properties
- Supports complex data types and user-defined classes
  - TelemetryLoggable interface (similar to Sendable, but imperative, not callback-based)
  - Struct and Protobuf serialization
  - Extensible via backend data type handlers (e.g. for unit types)
- Backends (e.g. NT, DataLog, mock, discard) dynamically configurable at any level of tree
- No integrated code-driven dashboard layout support

Dashboards and Tools

- Retire SmartDashboard, Shuffleboard, RobotBuilder, PathWeaver
- Add controller-hosted (web) versions of Elastic and AdvantageScope
  - Desktop versions will still be maintained
- Make it easier for teams to create custom web-based, controller-hosted dashboards (e.g. using FRC Web Components)
- Exploring options for controller-hosted (web) path-planning and system identification tools
  - Fairly significant development effort needed for this, so expect near-term to be desktop only
- LiveWindow functionality replaced by SystemCore web interface
- Concept for RobotBuilder v2.0, but development currently stalled

“Smart Motor” API

Single API that covers common characteristics of a “smart motor” to enable better examples and documentation.

- Utilize Units API to reduce issues related to different devices using different “native units”
- Reduces developer friction when swapping between different motor/controller types
- Allows for documentation, examples, and potentially even APIs that are applicable across motor controller types

Miscellania

- Package renames/restructure (edu.wpi.first.wpilibj → org.wpilib)
- Math
  - More powerful pose estimators
- Java units
  - Remove mutable units
  - Due to complexity to teams, not planning on migrating broader Java API to units-only
- Vendor dependencies
  - Need to figure out how to nicely bundle/package for controller-hosted development
- Docs
  - Split tutorial and reference documentation
- Improved alert/warnings (fewer console prints)

Alpha Testing

- https://forms.office.com/r/kh4yq3ggH8
- **Complete before May 2nd for first round beginning in June. Additional round including MotionCore expected in September**