Setting up Software for New Season (2023 – CenterStage edition)

Author: Dennis O'Brien
(dco6453@me.com)

Team: FTC-7462 (Not to Scale), Mentor

Table of Contents

1 Introduction

The full extent of setting up a new environment can be accessed using the FIRST Tech Challenge Software Development Kit link. However, this site does not mention how to layer on the *Road-Runner* and *Dashboard* software, created/maintained by ACME Robotics (Team #8367), that make the SDK a fully functioning environment for developing a highly competitive robot.

One requirement for the 2023 season is that we must use version 9.0 of the *FtcRobotController* software. Unfortunately, it appears as though ACME Robotics has not updated their web site for incorporating their software into this year's release. All references to their software is for version 8.2 of the *FtcRobotController*. So, these instructions detail how to start with version 9.0 and add in the 8.2 capabilities of *Road-Runner* and *Dashboard*.

Note: These instructions will be updated, time permitting, so that any references to version 8.2 will be replaced with version 9.0.

2 Setting up a New Repository

At the beginning of every season FIRST publishes a new version of the *FtcRobotController* software. It can be found on GitHub at FIRST-Tech-Challenge.

However, there is a similar piece of software called *Road-Runner*. *Road-Runner* is derived from this year's version of the *FtcRobotController* software and includes additional software needed to build more robust autonomous solutions.

This software, specifically *road-runner-quickstart*, must be forked, not cloned, into the *FIRST-4030* repository using the following steps:

1. Before you begin it is helpful to create a local folder on your PC to hold all of the software you will need to create a new repository for the season.

2. From the ACMERobotics GitHub site, select the *road-runner-quickstart* repository. (The site also include a *road-runner-ftc* repository but it is based on Kotlin, not Java.)

3. Select the *Create a new fork under* the *Fork* pulldown

4. Under the *Create a new fork*, make sure the *Owner* dropdown is set to *FIRST-4030*

5. Set the *Repository name* to a string that is indicative of the competition year (e.g., FTC-2023-<*SeasonName*>)

6. From the *FIRST-4030* repository, clone the new project onto your PC by select the *<> Code* button and copy the *HTTPS* URL to the clipboard

7. Within Android Studio, use the newly selected URL, and use *Git->Clone…* to clone the repository into a separate folder in PC your folder.

8. If a popup window appears saying *Project update recommended* then select the *upgraded* link and then select *Begin Upgrade*, followed by the *Run selected steps* button.

9. Refer back to the FIRST-Tech-Challenge and scroll down to the *README/Requirements* for the FtcRobotController to find the minimum level of Android Studio that we need to use.

10. In Android Studio, open the *About Android Studio* under the *Android Studio* menu dropdown to verify you are using the proper level.

Once the repository is completed you need to add in those persons who can access it using the following steps:

1. Select the *Insights* link at the top of the repository

2. Select the *People* button in the left-hand list of options

3. Add all other developers on the coding team with *Write* access

3 Installing the Dashboard

Supporting autonomous capability is dependent on using *Road-Runner*, supported by the *Dashboard*.

Note: The requirement for this year's competition is to use version **9.0** of the *FtcRobotController*. However, the instructions for installing *Road-Runner* was written for version 8.2 of the software.

What this means, for now, is that the *master* branch of the repo must be augmented with a previous version of *Road-Runner* so that *Method 2: Installing RR on your project* can be followed. So these instructions attempt to combine both sources of information.

Installing the *Dashboard* involves modifying files listed in the IDE under *Gradle Scripts*.

Update low level files using the following steps:

1. As a "programming best practice" it is useful to isolate unique work to an individual GitHub branch. This makes it easy to rollback your environment if you inadvertently introduce an error(s) that may be hard to unwind. In Android Studio, use the command *Git -> New Branch…* to create a local space to keep your changes. For the purpose of this document, name it *InstallDashboard*. (Make sure the *Checkout branch* dialog button is checked.)
2. Open the file *build.dependencies.gradle*.
3. At the top of the file add the following lines to the *repositories* block:

```
maven { url = 'https://maven.brott.dev/' }
flatDir {
   dirs rootProject.file('libs')
}
```

4. At the bottom of the file add the following lines to the *dependencies* block:

```
implementation 'com.acmerobotics.roadrunner:core:0.5.6'
implementation 'com.acmerobotics.dashboard:dashboard:0.4.15'
implementation 'org.apache.commons:commons-math3:3.6.1'
implementation 'com.fasterxml.jackson.core:jackson-databind:2.12.7'
```

   Note: Update the version numbers for the *dashboard* and *road-runner* with values from the right-hand column of each respective repository on GitHub.

5. Open the file *build.gradle* for the *Project:<project>*

6. At the bottom of the file add the following lines to the *repositories* block:

```
flatDir {
  dirs '../libs'
}
```

7. Open the file *build.gradle* for the *Module :TeamCode*
8. At the bottom of the file add the following line **below** the *dependencies* block:

```
apply from: '../build.dependencies.gradle'
```

9. Delete the following lines from the dependencies block:

```
implementation "com.acmerobotics.roadrunner:ftc:0.1.8"
implementation "com.acmerobotics.dashboard:dashboard:0.4.13"
```

10. Open the file *build.common.gradle*
11. At the bottom of the file add the following lines to the *repositories* block:

```
flatDir {
  dirs '../libs'
}
```

12. Select the *Sync Now* button at the top of the IDE
13. Build the project to make sure there are no errors before you add in some selected software.

Note1: Keep in mind, if ever you make a change to any gradle file then you **must** do a *Sync Now* to have that change recognized before doing a *Build->Make Project*.

Note2: As of 2024, the background image of the dashboard will appear once the Start button is selected. If the displayed image is not current for the current year then check the Git repository and update the gradle settings for an updated version number.

4 Installing Selected Road-Runner Routines

Up until this point, all work has been relative to version 9.0 of the *FtcRobotController*. Use the following commands to replace some version 9.0 routines with those from version 8.2:

1. Delete the *tuning* folder.

   Note: Normally you should use the *Refactor->Safe delete…* option so that you ensure that you do not introduce a ripple effect of broken links that may be difficult to unwind.

2. Delete the following files (in this order): *TankDrive*, *MecanumDrive*, *TwoDeadWheelLocalizer*, *ThreeDeadWheelLocalizer*, *Localizer*, and *Drawing*.

3. Move the file *PoseMessage* out of the messages folder and into *org.firstinspires.ftc.teamcode* folder.

4. Delete all other files in the *messages* folder.

5. Delete the *messages* folder.

6. Open *PoseMessage*.

7. Add the string *.geometry* after *roadrunner* in the *import* statement.

8. Replace
   .position.x with *.getX()*,
   .position.y with *.getY()*, and
   .heading.toDouble() with *.getHeading()*
   found in the body of the routine

9. From the ACMERobotics GitHub site, select the *road-runner-quickstart* repository.

10. Change the view of the repository from *master* to *quickstart1*.

11. Select the *<> Code* button and copy the *HTTPS* URL to the clipboard

12. Within Android Studio, use the newly selected URL, and use *Git->Clone…* to clone the repository into your folder on the PC.

    Note: Do not worry about the name of the repository. It will only remain around long enough to copy/paste software from it to your current project.

13. In the lower right-hand corner, change the checked out branch name from *master* to *origin/quickstart1*.

14. From the *TeamCode* folder, copy the *drive*, *trajectorysequence*, and *util* folders.

15. Return to your project and highlight the *org.firstinspires.ftc.teamcode* folder.

16. Select the *Paste* button. You will be prompted for a folder to paste the files in.

17. Select the *OK* button

18. You can select either the *Cancel* or *Add* button and select *Add*. This will place all files into the branch of the git repository on your PC.

    Note: Selecting *Cancel* will keep the files under the folder structure of the project but outside the repository. (Ultimately, you will have to add them to the git repository.)

19. Within the *drive* folder is a file called *DriveConstants*. The values in this file are all generic (i.e. not useful) and must be replaced with values specific to your robot for this year. Go thru the defining of constants found at this link found on the *Road-Runner* web page and generate new constants for this year's robot.

    Replace the *DriveConstants* file with the one you just generated.

    Note: We maintain a spreadsheet on our Google Drive of all of our past/current values that should also be used as a reference.

20. Surprisingly, the newly generated *DriveConstants* file does not contain entries for the orientation of the Control Hub and Expansion Hub. Add the following 2 lines to the DriveConstants file:

```
public static RevHubOrientationOnRobot.LogoFacingDirection
        LOGO_FACING_DIR =
            RevHubOrientationOnRobot.LogoFacingDirection.RIGHT;
public static RevHubOrientationOnRobot.UsbFacingDirection
        USB_FACING_DIR =
            RevHubOrientationOnRobot.UsbFacingDirection.UP;
```

21. Open *SampleMecanumDrive* and modify the names of the motors so that they match those in your configuration file.

5 Verifying the SDK via Road-Runner

Presumably all of the above work will result in working robot. The only way to know is to download the project onto a robot and calibrate it using *Road-Runner*.

Begin with opening a web browser to the Road-Runner web site. This is a "cookbook" of steps to follow that are pretty self-explanatory.

If this is the first time that you have used *Road-Runner* then select the *Before You Start!* Button and review the terminology and tips.

Select the High Level Overview header in the left-hand column. It will display the steps you will go thru.

The first step in calibrating a robot is to generate Drive Constants. After that, follow the steps in the order shown on the chart.

6 Preserving Changes Back into GitHub

Your local environment now needs to be rolled up into a comprehensive release and pushed back up to the *FIRST-4030* GitHub repository. All of the files in the project that appear in green have changed since the last commit. Those in red are new to the repository.

Within Android Studio, use the following steps to commit your changes to the team wide GitHub repository:

1. In the left-hand column of the project window, right-click on the *TeamCode* folder.
2. From the dropdown menu select *Git -> Add*.
3. From the dropdown menu select *Git -> Commit Directory...*.
4. A new window will be displayed showing all of the files that will be committed to the local *master* repository.
5. Make sure all of the files are selected. (The gradle files may not be selected but they should be.)
6. Add a brief text string in the *Commit Message* window that explains why the commit is being done.
7. Select the *Commit* button. (**DO NOT** select the *Commit and Push…* button since you are presumably working on a local branch and the "push" will not do what you think it should.)
8. Android Studio will analyze the code one last time and display its results. Assuming **no errors** are generated, then push the *Commit Anyway* button.
9. At this point all of your changes have been committed to your local *InstallDashboard* branch. These changes now need to move to your local *master* branch.
10. From the dropdown menu select *Git -> Branches...*.
11. Select the *master* branch and *Checkout*. (You will see the branch name change in the lower right-hand corner of the IDE.)
12. From the dropdown menu select *Git -> Merge...*.
13. Select the *InstallDashboard* branch.
14. Select the *Merge* button.
15. Move all changes up to the team wide repository by selecting *Git -> Push* button.
16. A new window will be displayed listing all files that will be committed. Assuming every looks right select the *Push* button.

7 Connecting Repository to Discord Server

Now that you have a repository it is useful to establish a webhook from the repository to our Discord *#github* channel. (Establishing a webhook results in a notice to everyone in the *#github* channel being notified whenever there is a change to the repository.)

Before you begin, it is assumed that you have the proper privileges (e.g., webhook, moderation, admin, etc.) in Discord to complete the process. (You will know if you have the proper level of privileges if you **can** complete step #1.)

Use the following steps to establish a webhook between our GitHub repository and Discord:

1. In Discord, right-click on *#github* and select *Edit Channel*. This should display a list of options, in a column, on the left-hand side.
2. Select *Integrations* and then *View Webhooks*.
3. Select the *New Webhook* button.
4. Expand the view of the new webhook and change the name of it to something meaningful to this year's work.
5. Select the *Save Changes* button to save the name change.
6. Select the *Copy Webhook URL* button
7. In GitHub, select the newly created repository
8. Select the *Settings* button at the top of the window
9. Select the *Webhooks* button in the left-hand set of buttons
10. Select the *Add webhook* button
11. Log in to the repository, if requested
12. Paste the string into the *Payload URL* field in GitHub
13. Append the string */github* to the *Payload URL*
14. Set the *Content type* to *application/json*
15. Select the *Send me everything* radio button under the *Which events would you like to trigger this webhook*
16. Select the *Add webhook* button
17. Select the *Starred* option in the upper right-hand page which will send a minimalist message to Discord to show that the webhook is working.

8 Troubleshooting

In building this procedure there were a number of errors/issues that arose that took many hours to track down. Here are those errors and possible fixes for them.

Error / Issue:	When building the software the error "Unsupported class file major version 63" occurred.
Solution:	Check all of the supporting libraries to make sure there versions are compatible. That is, comment out each library and substitute in one of a lower version. Be sure to do a *Sync Now* before compiling.

Error / Issue:	During checkout of the robot the *Start* and *Stop* button do not stay on when you hit the *Init* button.
Solution1:	Check the values in *DriveConstants* for legitimate numbers.
Solution2:	Make sure that the motor declarations in *SampleMecanumDrive* match those in the robot's config file..
Solution3:	Run the opmode using a Driver Station to see if there is a thrown exception