

jRIO

Written by Nathaniel Lattimer of the Midpark Robotics Team (#1248)

What is jRIO

jRIO is a comprehensive cRIO and hardware simulator written in Java, for Java. It is a library that is designed to replicate that of the library provided by FIRST so that it requires minimal setup for the developers to test their code and get on with the exciting and wonderful process of development. This library is also designed to replicate the actual state and actions of the cRIO even down to the iterative periodic frequency documented at 50 Hz.

Note: All documentation assumes that you are currently using Netbeans 7.1+ on your machine. For other IDE's, use equivalent steps to integrate jRIO into your project.

How to Setup a Project for jRIO

The jRIO library cannot be run in a normal FRC Java project directly so, to get it to operate properly, a new, regular *Java Application* project must be created.

1. Create a new *Java Application* Project in Netbeans
2. Right click the Project and go to *Properties*
3. Go to *Libraries* and click *Add JAR/Folder* and add your local *jRIO.jar* file.
4. Copy all source files from the *FRC Robot* Project that you wish to emulate into the main package of your *Java Application* Project.
5. Go to the main class file and start the configuration phase.

Configuring jRIO for Emulation

The great thing about jRIO is how it allows one to create their own cRIO configuration to match that of their robot! For example, one configurable attribute is how many module slots the cRIO has; 4 or 8 which allows teams with a 4 slot cRIO to better simulate their code on jRIO; minimal configuration needed code-wise!

jRIO requires that the cRIO module count, the RobotBase launch point, the DashboardDisplay, and any modules that the cRIO is to use are all specified explicitly before the actual jRIO is launched. Please refer to the following example to better see how this process is done.

Note: This following example configuration is for a robot with only one Digital Sidecar in the default slot (2). Please refer to Configuration Guide at the end of this document for a detailed listing on configuration options.

```
package testrobot;  
import org.jRIO.DashboardDisplay.DashboardDisplay;
```

```

import org.jRIO.mainframe.DigitalSidecar;
import org.jRIO.mainframe.Main;
import org.jRIO.mainframe.cRIO;

public class TestRobot{

    public static void main(String[] args){
        new DashboardDisplay(); //creates a new Dashboard Display

        cRIO.setModuleCount(false); // false for 8 slot cRIO

        new DigitalSidecar(); //will set to the default digital sidecar module slot

        //assuming that RobotTemplate is the main Robot class for your FRC Robot Project
        Main.setRobotBase(new RobotTemplate());

        Main.start(); //hands control off to the robot for simulation
    }
}

```

The first thing that is always recommended to do is to create the `DashboardDisplay` first. A lot of hardware is going to try to integrate into the Dashboard at runtime or at construction time and it would save a lot of trouble to create the Dashboard first. Please note that one must only create a new `DashboardDisplay` object to use it properly; there exists no need to save the object, just leaving it instantiated is actually recommended. The constructors should handle the integration of **all** components correctly and gracefully. The `cRIO` module count setting should be done before any modules are attached because the default location for modules are set when the module count is declared and would cause *NullPointerExceptions* otherwise. All modules that are to be added only need to be created in memory, just like the `DashboardDisplay`. This declaration just creates a new `DigitalSidecar` in the default slot. To specify a slot, use the notation: *new DigitalSidecar(slotNum);*

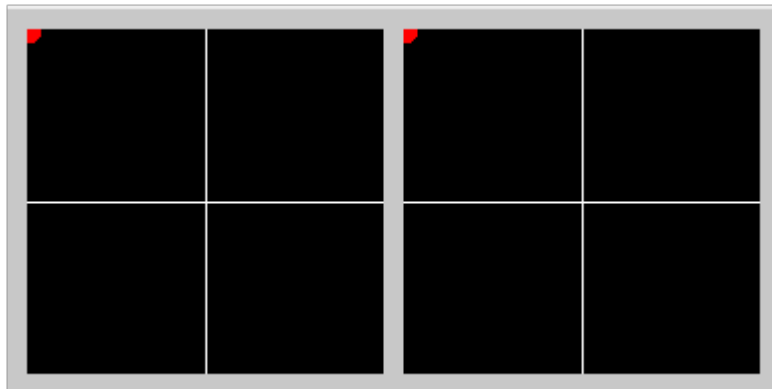
The `RobotBase` should be declared just before the `jRIO` emulation is to start. This example assumes that you left the FIRST FRC Java Project class name as default (*RobotTemplate*) and creates an object in memory which is then passed to the *Main* class through the method *setRobotBase*. This method **must** be called **after** the `cRIO` is configured and **before** the emulation is started, otherwise, *NullPointedExceptions* will populate the output window.

Finally, the command: *Main.start();* may be used to start the emulation. Just simply compile and run the project for emulation! Please read *Interpreting the Dashboard* for further instruction.

Interpreting the Dashboard

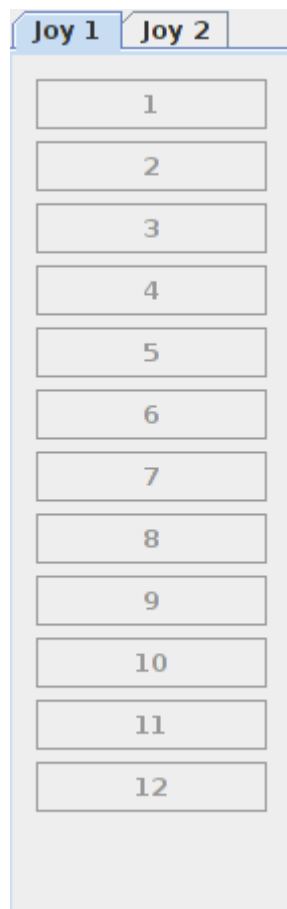
The Dashboard is designed to give a quick and compact display of all the hardware that would be required on the robot as interpreted from the code provided.

Joysticks:



The red circles are indicative of the current focus of the joysticks.

Joystick Buttons:



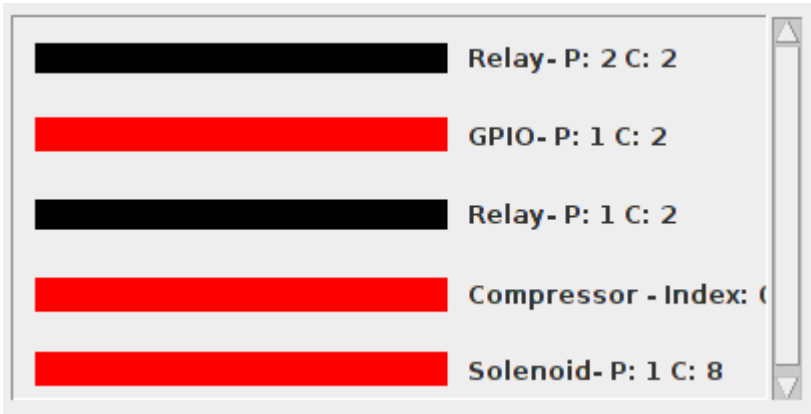
Greyed out in periods of non-use.

Speed Controller Output Displays:











Speed Controller Type – Pin number – Module Channel

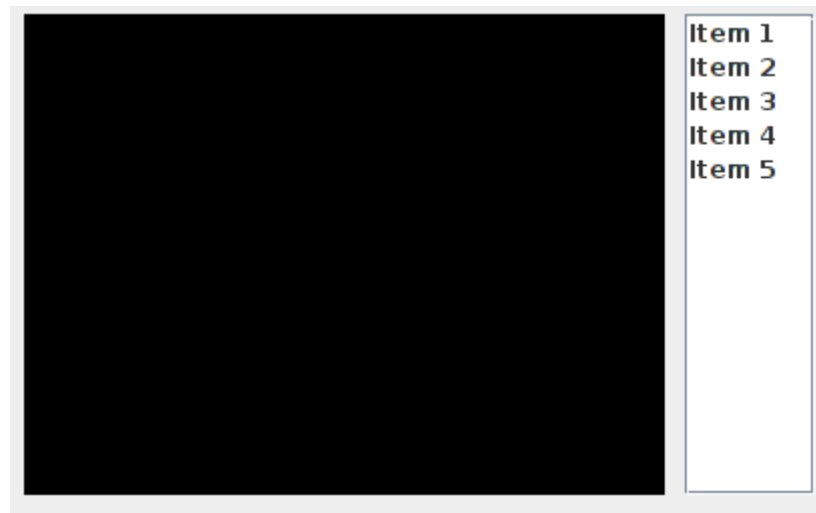
Pneumatic Displays:



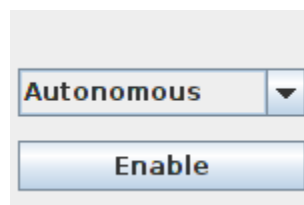
Refer to this key for values:

Relay Key		GPIO/Solenoid State		Compressor State	
	Off State		Forward	 False	 Off
	On State		Reverse	 True	 On

Camera Display:



Enable/Disable Autonomous/Teleop Switch:



When the button is pushed in, the robot is in the state specified by the combo box. It is recommended to disable then switch states instead of switching states while enabled, however this is still possible.

Configuration Guide

The robot has quite a bit of configuration options to choose from before running the emulation. Here is a guide depicting in greater detail the different ways to configure.

DigitalSidecar

- *DigitalSidecar()*
 - Creates a new DigitalSidecar on the default digital module channel
- *DigitalSidecar(int)*
 - Creates a new DigitalSidecar on the specified module channel
 -

SolenoidBreakout

- *SolenoidBreakout()*
 - Create a new SolenoidBreakout on the default solenoid module channel
- *SolenoidBreakout(int)*
 - Create a new SolenoidBreakout on the specified module channel

Main.setRobotBase(RobotBase)

- Use either Iterative or Simple Robot
- Command based robot not supported

Main.start()

- Starts the simulation of the robotbase code

cRIO.setModuleCount(boolean)

- Is true if the module count is 4 otherwise it is 8
- Must be used before modules can be added

cRIO.addJoystick(Joystick)

- Do not use; for simulator's use only
- A maximum of 2 joysticks may be created