SHUFFLEBOARD

Last Updated: 08-10-2019

Table of Contents

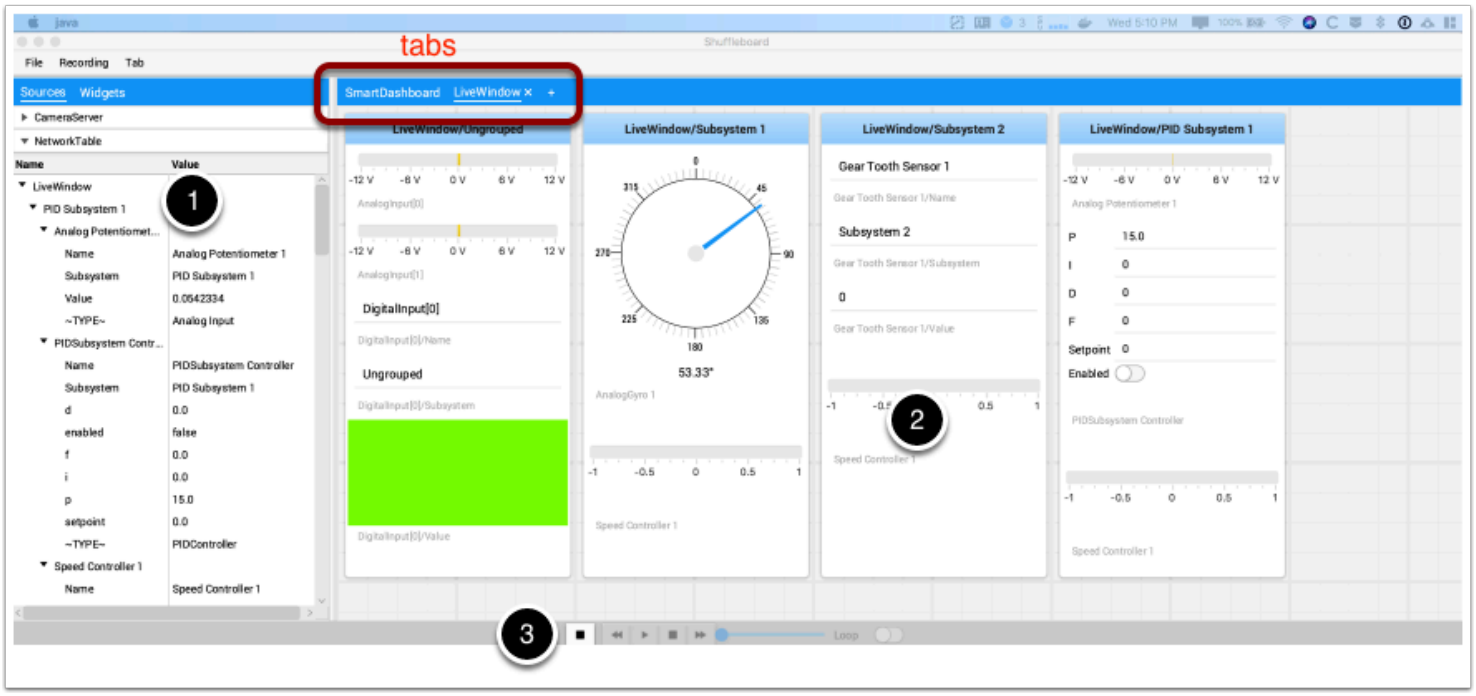
Getting started with Shuffleboard

Tour of Shuffleboard

Shuffleboard is a dashboard for FRC based on newer technologies such as JavaFX that are available to Java programs. It is designed to be used for creating dashboards for C++ and Java programs. If you've used SmartDashboard in the past then you are already familiar with many of the features of Shuffleboard since they fundamentally work the same way. But Shuffleboard has many features that aren't in SmartDashboard. Here are some of the highlights:

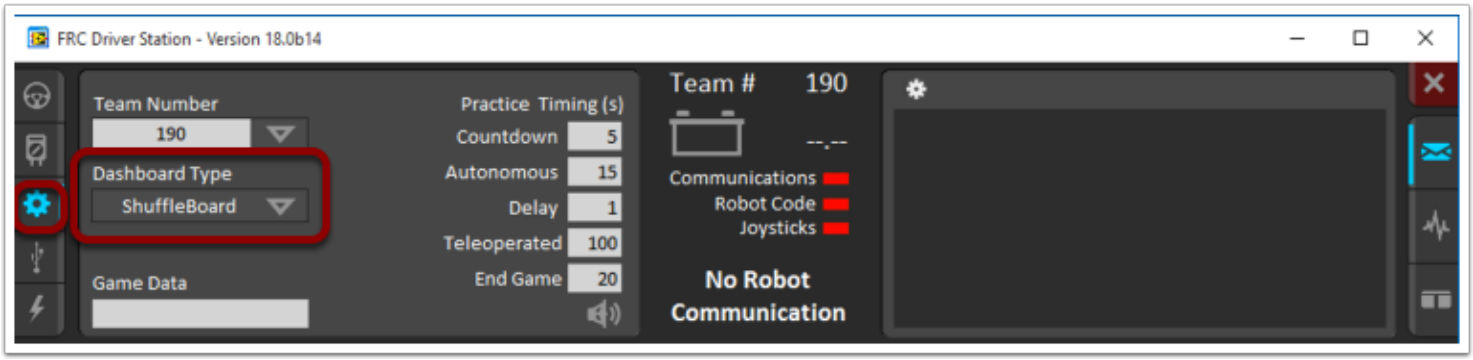
- Graphics is **based on JavaFX**, the Java graphics standard. Each of the components has an associated style sheet so it becomes possible to have different "skins" or "themes" for Shuffleboard. We supply default light and dark themes.
- Shuffleboard supports **multiple sheets for the display of your data**. In fact you can create a new sheet (shown as a tab in the Shuffleboard window) and indicate if and which data should be autopopulated on it. By default there is a Test tab and a SmartDashboard tab that are autopopulated as data arrives. Other tabs might be for robot debugging vs. driving.
- Graphical **display elements (widgets) are laid out on a grid** to keep the interface clean and easy to read. You can change the grid size to have more or less resolution in your layouts and visual cues are provided to help you change your layout using drag and drop. Or you can choose to turn off the grid lines although the grid layout is preserved.
- Layouts are saved and the previous layout is instantiated by default when you run shuffleboard again.
- There is a **record and playback** feature that lets you review the data sent by your robot program after it finishes. That way you can carefully review the actions of the robot if something goes wrong.
- **Graph widgets are available for numeric data** and you can drag data onto a graph to see multiple points at the same time and on the same scale.
- You can extend Shuffleboard by writing your own widgets that are specific to your team's requirements. Documentation on extending it are on the Shuffleboard Github Wiki.

Shuffleboard



1. **Sources area**: Here are data sources from which you can choose values from NetworkTables or other sources to display by dragging a value into one of the tabs
2. **Tab panes**: This is where you data is displayed from the robot or other sources. In this example it is Test-mode subsystems that are shown here in the LiveWindow tab. This area can show any number of tabbed windows, and each window has it's own set of properties like grid size and auto-populate.
3. **Record/playback controls**: set of media-like controls where you can playback the current session to see historical data

Starting Shuffleboard



You can start Shuffleboard in one of three ways:

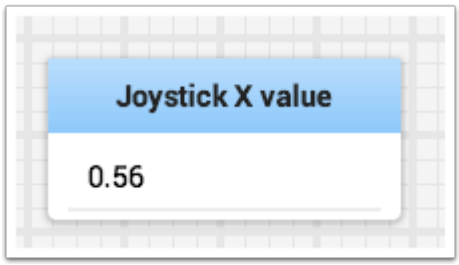
1. You can automatically start it when the Driver Station starts by setting the "Dashboard Type" to Shuffleboard in the settings tab as shown in the picture above.
2. You can run it by double-clicking on the .jar file in <user-directory>/WPILib/tools/ shuffleboard.jar. This is useful on a development system that does not have the Driver Station installed such as a Mac or Linux system.
3. You can start it from the command line by typing the command:
   java -jar shuffleboard.jar
   from *home-dir/* WPILib/tools directory. This is often easiest on a a development system that doesn't have the Driver Station installed.

Getting robot data onto the dashboard

The easiest way to get data displayed on the dashboard is simply to use methods in the SmartDashboard class. For example to write a number to Shuffleboard write:

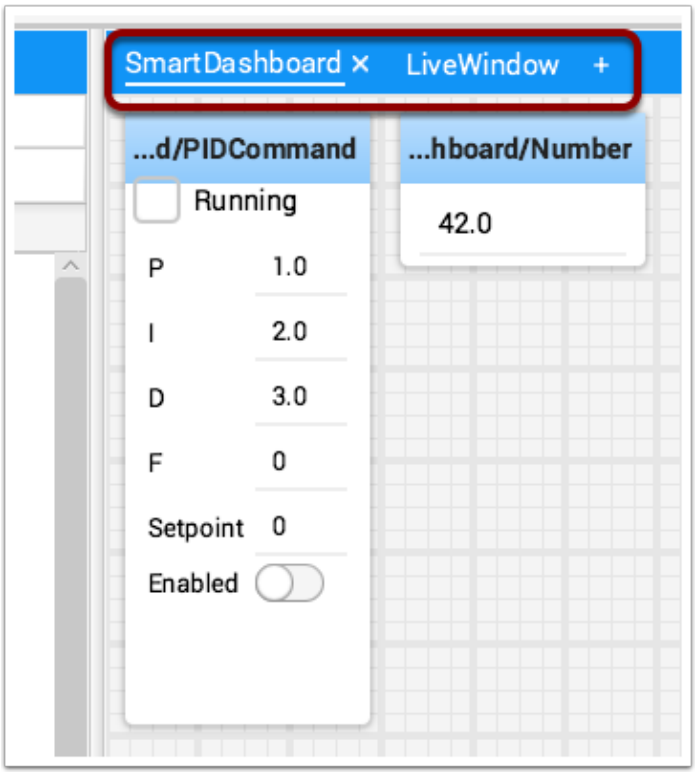
```
SmartDashboard.putNumber("Joystick X value", joystick1.getX());
```

to see a field displayed with the label "Joystick X value" and a value of the X value of the joystick. Each time this line of code is executed, a new joystick value will be sent to Shuffleboard. *Remember: you must write the joystick value whenever you want to see an updated value. Executing this line once at the start of the program will only display the value once at the time the line of code was executed.*


Joystick X value
0.56

Displaying data from your robot

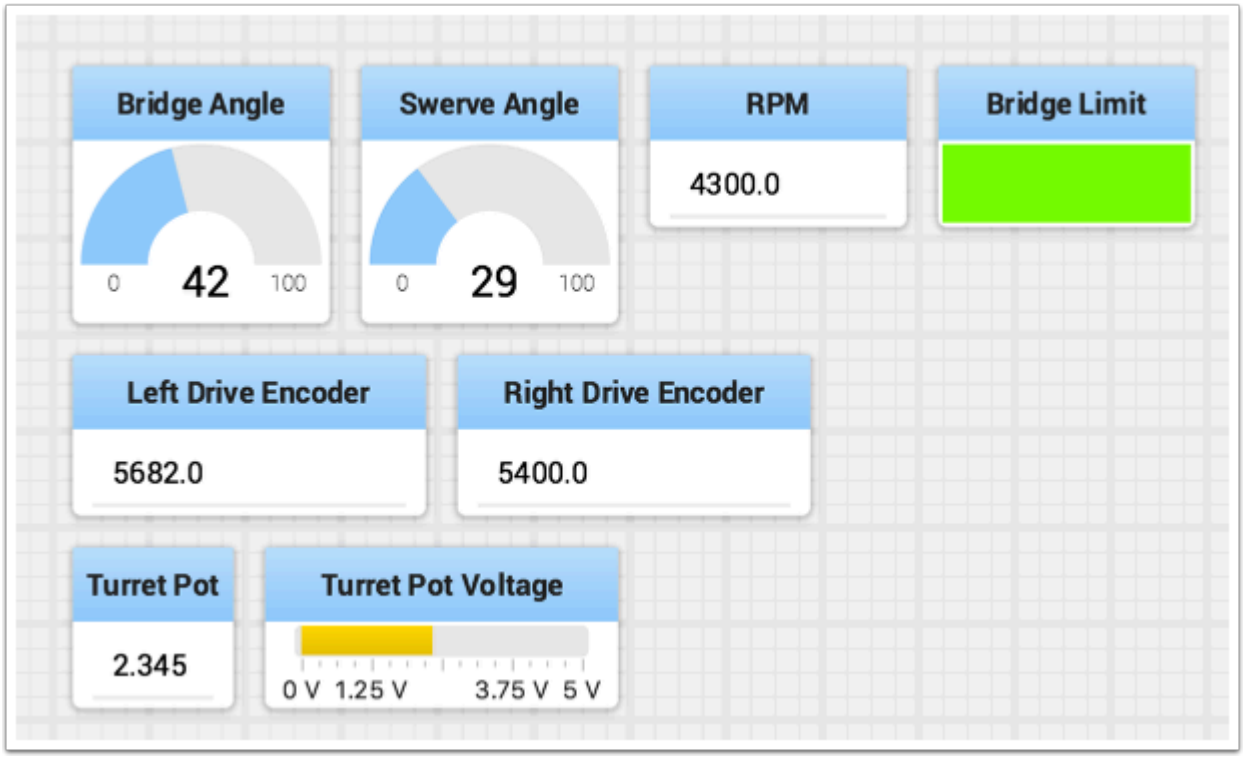
Your robot can display data in regular operating modes like Teleop and Autonomous modes but you can also display the status and operate all the robot subsystems when the robot is switched to Test mode. By default you'll see two tabs when you start Shuffleboard, one for Teleop/ Autonomous and another for Test mode. The currently selected tab is underlined as can be seen in the picture below.



Often debugging or monitoring the status of a robot envolves writing a number of values to the console and watching them stream by. With Shuffleboard you can put values to a GUI that is automatically constructed based on your program. As values are updated, the corresponding GUI element changes value - there is no need to try to catch numbers streaming by on the screen.

Displaying values in normal operating mode (autonomous or teleop)

```
protected void execute() {
    SmartDashboard.putBoolean("Bridge Limit", bridgeTipper.atBridge());
    SmartDashboard.putNumber("Bridge Angle", bridgeTipper.getPosition());
    SmartDashboard.putNumber("Swerve Angle", drivetrain.getSwerveAngle());
    SmartDashboard.putNumber("Left Drive Encoder", drivetrain.getLeftEncoder());
    SmartDashboard.putNumber("Right Drive Encoder", drivetrain.getRightEncoder());
    SmartDashboard.putNumber("Turret Pot", turret.getCurrentAngle());
    SmartDashboard.putNumber("Turret Pot Voltage", turret.getAverageVoltage());
    SmartDashboard.putNumber("RPM", shooter.getRPM());
}
```



You can write Boolean, Numeric, or String values to Shuffleboard by simply calling the correct method for the type and including the name and the value of the data, no additional code is required.

- Numeric types such as char, int, long, float or double call SmartDashboard.putNumber("dashboard-name", value).
- String types call SmartDashboard.putString("dashboard-name", value) and

- Boolean types call SmartDashboard.putBoolean("dashboard-name", value)

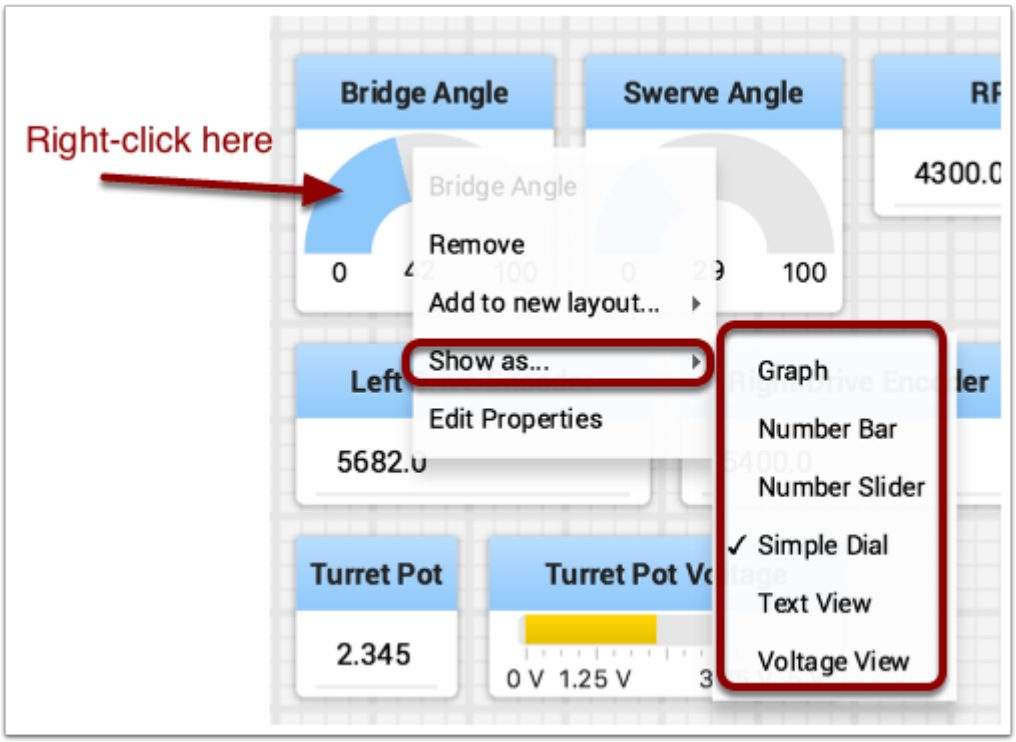
Note: the methods are all part of the SmartDashboard class. The values will be displayed automatically in both Shuffleboard or SmartDashboard. You can even have an instance of Shuffleboard and SmartDashboard running and the values will be written to both of them.

Each time your program calls SmartDashboard.put<type>() a new value is written to that widget in the dashboard, so to keep changing values up to date, be sure to call the put<type>() function whenever the value changes. As you can imagine this is a great way of debugging and getting status of your robot as it is operating.

The SmartDashboard class is just shorthand for writing to the SmartDashboard subtable in NetworkTables.

Changing the display type of data

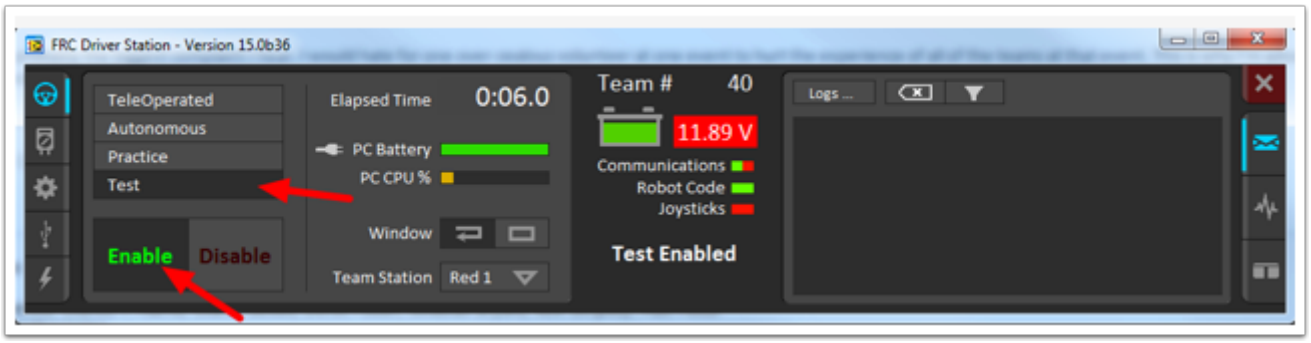
Depending on the data type of the values being sent to Shuffleboard you can often change the display format. In the previous example you can see that number values were displayed as either decimal numbers, a dial to better represent angles, and as a voltage view for the turret potentiometer. To set the display type right-click on the tile and select "Show as...". You can choose display types from the list in the popup menu.



Displaying data in Test mode

You may add code to your program to display values for your sensors and actuators while the robot is in Test mode. This can be selected from the Driver Station whenever the robot is not on the field. The code to display these values is automatically generated by RobotBuilder or manually added to your program and is described in the next article. Test mode is designed to verify the correct operation of the sensors and actuators on a robot. In addition it can be used for obtaining setpoints from sensors such as potentiometers and for tuning PID loops in your code.

Setting test mode

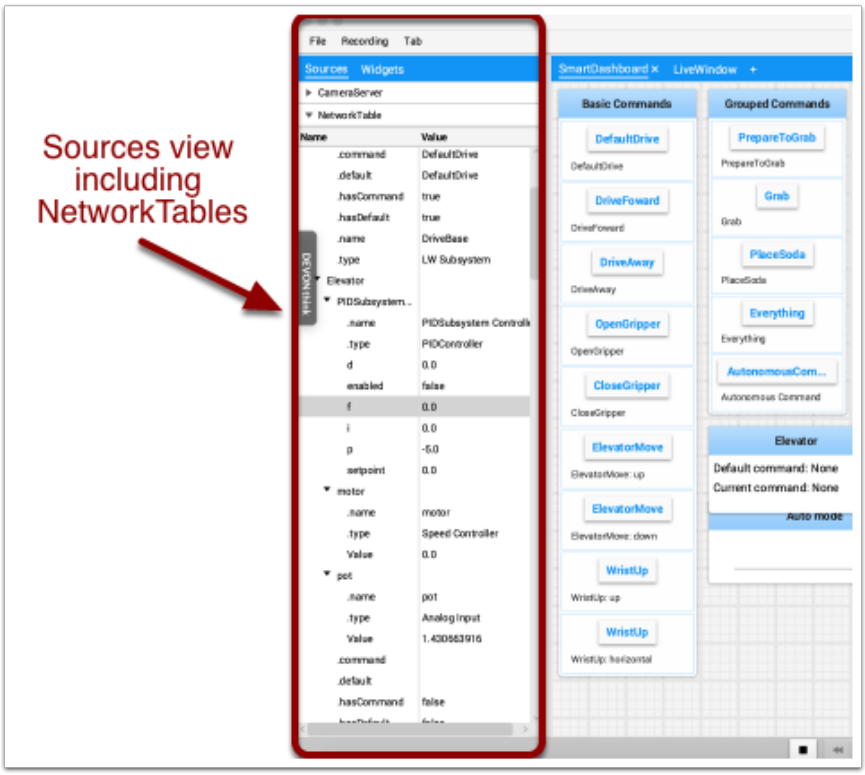


Enable Test Mode in the Driver Station by clicking on the "Test" button and setting "Enable" on the robot. When doing this, Shuffleboard will display the status of any actuators and sensors used by your program organized by subsystem.

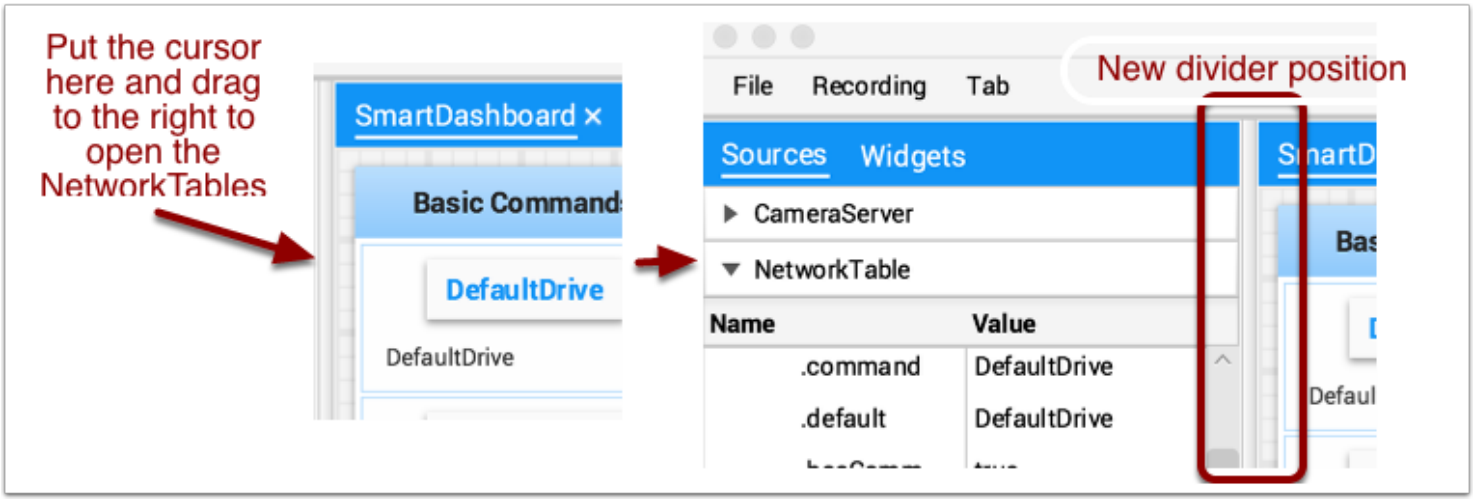
Getting data from the Sources view

Normally network table data automatically appears on one of the tabs and you just rearrange and use that data. Sometimes you might want to recover a value that was accidentally deleted from the tab or display a value that is not part of the SmartDashboard/ NetworkTable key. For these cases the values can be dragged onto a pane from NetworkTable view under Sources on the left side of the window. Choose the value that you want to display and just drag it to the pane and it will be automatically created with the default type of widget for the data type.

Shuffleboard



Note: sometimes the Sources view is not visible on the left - it is possible to drag the divider between the tabbed panes and the Sources so the sources is not visible. If this happens move the cursor over the left edge and look for a divider resizing cursor, then left click and drag out the view. In the two images below you can see where to click and drag, and when finished the divider is as shown in the second image.

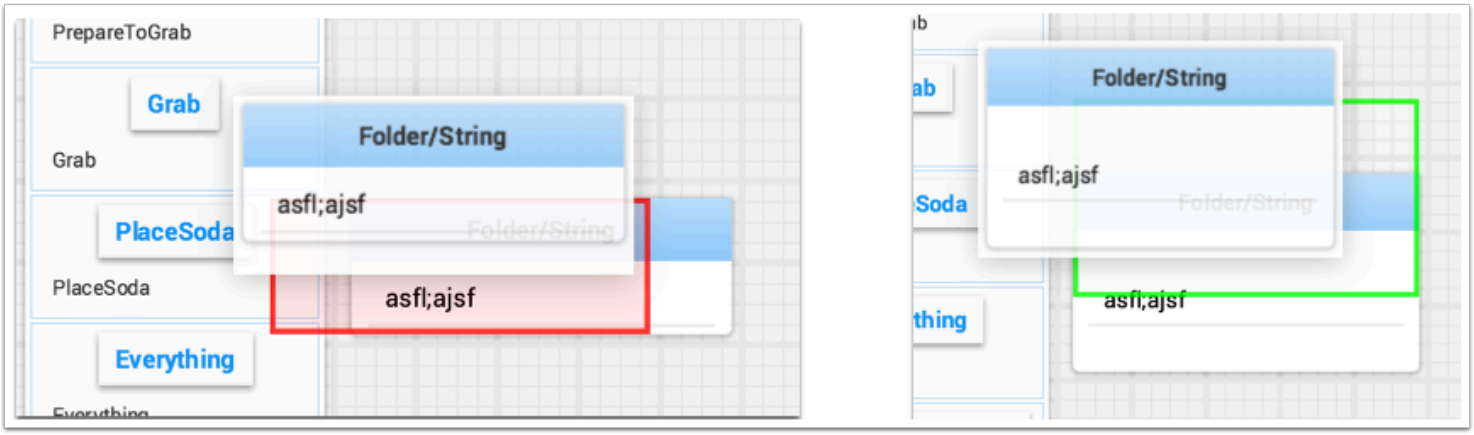


Working with widgets

The visual displays that you manipulate on the screen in Shuffleboard are called widgets. Widgets are generally automatically displayed from values that the robot program publishes with NetworkTables.

Moving widgets

Widgets can be moved simply with drag and drop. Just move the cursor over the widget, left-click and drag it to the new position. When dragging you can only place widgets on grid squares and the size of the grid will effect the resolution of your display. When dragging a red or green outline will be displayed. Green generally means that there is enough room at the current location to drop the widget and red generally means that it will overlap or be too big to drop. In the example below a widget is being moved to a location where it doesn't fit.

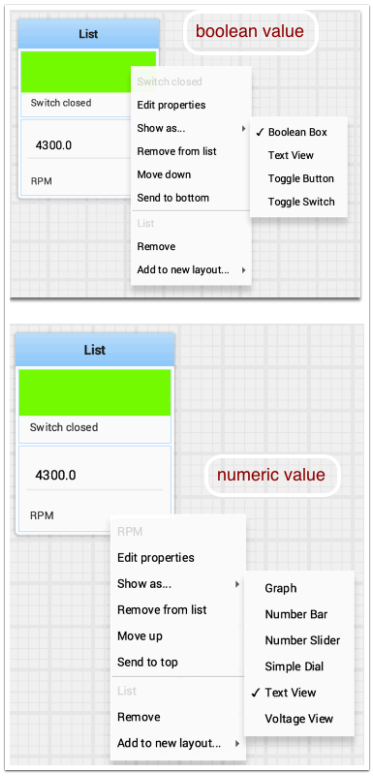


Resizing widgets

Widgets can be resized by clicking and dragging the edge or corner of the widget image. The cursor will change to a resize-cursor when it is in the right position to resize the widget. As with moving widgets, a green or red outline will be drawn indicating that the widget can be resized or not. The example below shows a widget being resized to a larger area with the green outline indicating that there is no overlap with surrounding widgets.

Changing the display type of widgets

Shuffleboard is very rich in display types depending on the data published from the robot. It will automatically choose a default display type, but you might want to change it depending on the application. To see what the possible displays are for any widget, right-click on the widget and select the "Show as..." and from the popup menu, choose the desired type. In the example below are two data values, one a number and the other a boolean. You can see the different types of display options that are available to each. The boolean value has only two possible values (true/false) it can be shown as a boolean box (the red/green color), or text, or a toggle button or toggle switch. The. number value can be displayed as a graph, number bar, number slider, dial, text, or a voltage view depending on the context of the value.



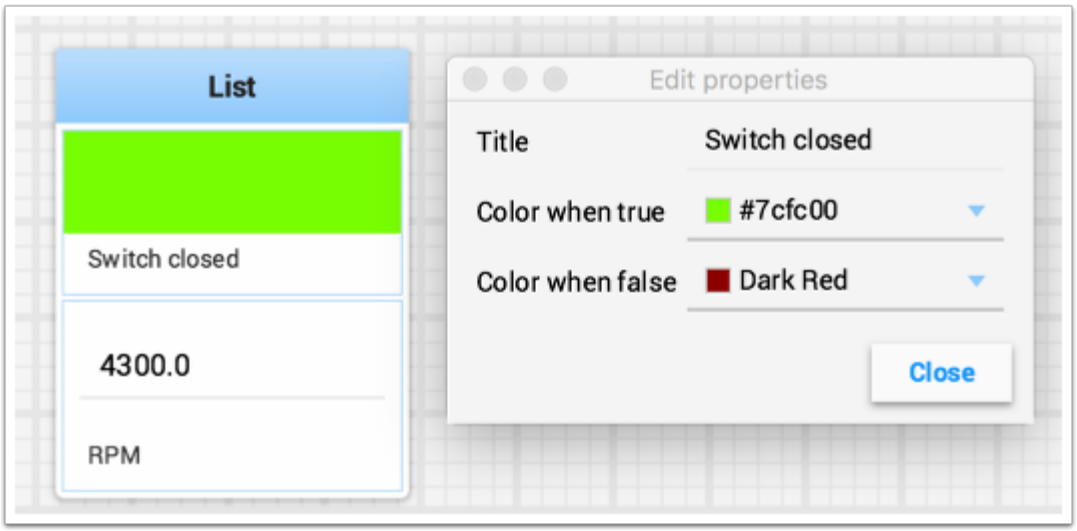
Changing the title of widgets

You can change the title of widgets by double-clicking in their title bar and editing the title to the new value. If a widget is contained in a layout, then right-click on the widget and select the properties. From there you can change the widget title that is displayed.

Changing widget properties

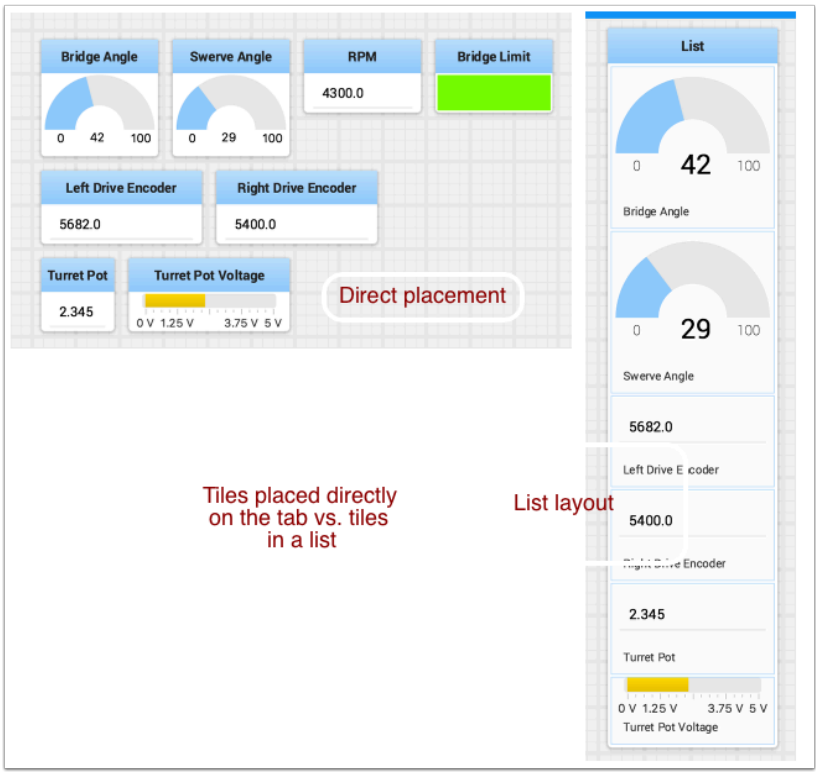
You can change the appearance of a widget such as the range of values represented, colors or some other visual element. In cases where this is possible right-click on the widget and select "Edit properties" from the popup menu. In this boolean value widget shown below, the widget title, true color and false color can all be editted.

Shuffleboard

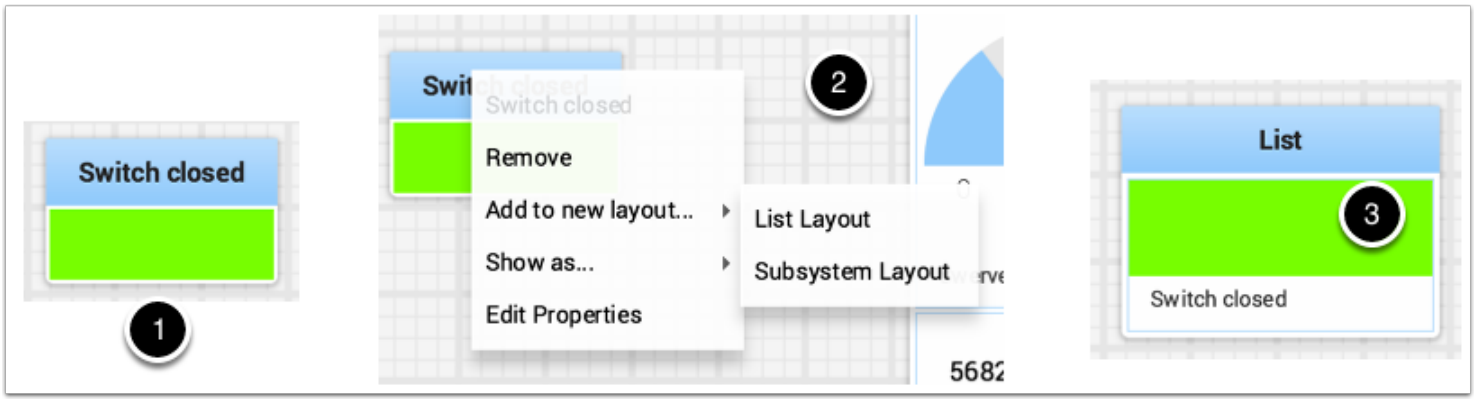


Working with Lists

Lists are sets of tiles that are be logically grouped together in a vertical layout. When tiles are added to a list, it becomes visually obvious that the tiles are related in function. In addition tiles in lists take up less screen space than the same tiles directly on the desktop because tiles in a list don't display the dark window title, but instead display a smaller textual label documenting the contents of the list.



Creating a list



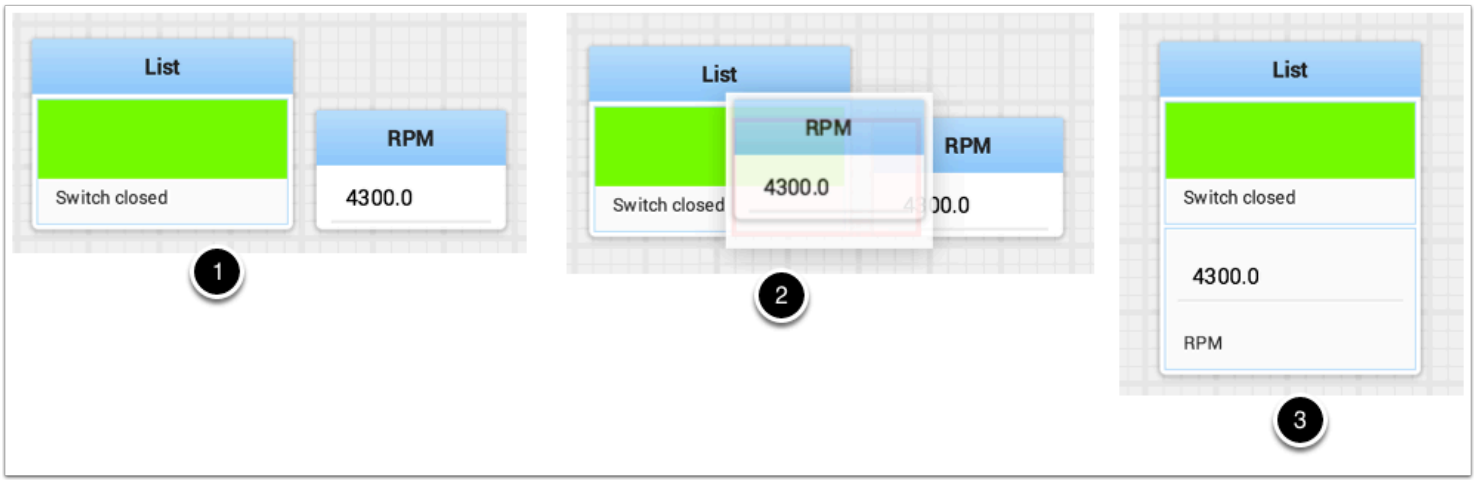
A list can be created by:

1. right-clicking on the first tile that should go into the list.
2. Select the "Add to new layout..." option from the popup menu.
3. A new list will be created called "List" and the tile will be at the top of it. Notice that tiles in lists do not have the window title at the top, but instead have the text that was in the window title.

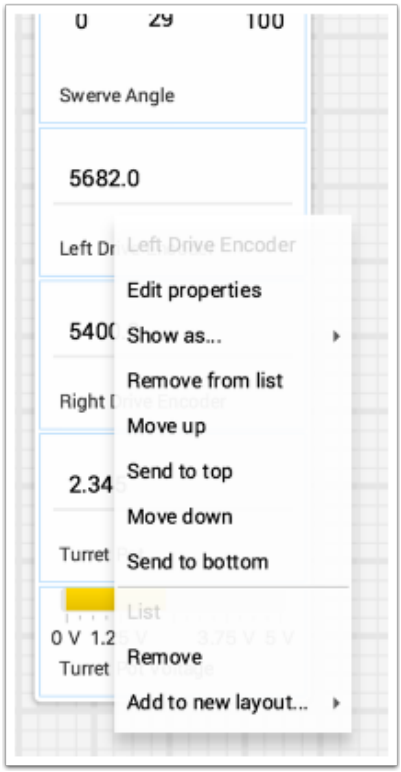
Adding tiles to a list

Add tiles to a list layout by:

1. identify the list and the tile to be added.
2. drag the new tile onto the list.
3. the tile will be added to the list. If there is no room as the list is currently sized, the tile will be added off the end of the list and a vertical scrollbar will be added if it's not already there.



Rearranging tiles in a list



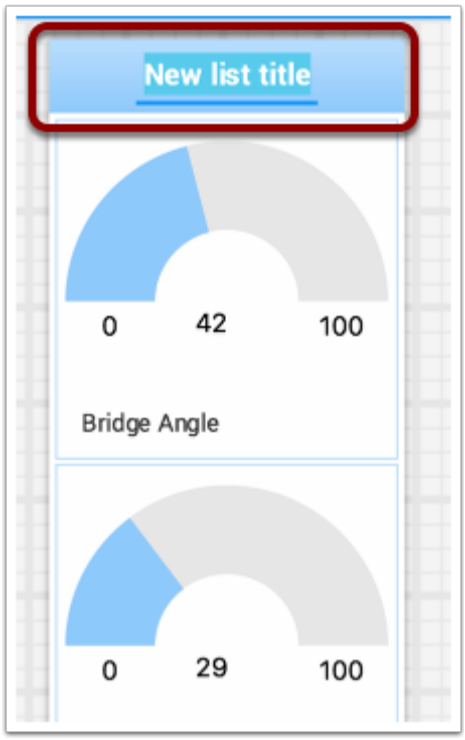
Tiles in a list an be rearranged by right-clicking on the tile to be moved and selecting:

1. "Move up" to move the tile before the previous tile

2. "Move down" to move the tile after the next tile
3. "Send to top" to move the tile to the top of the list
4. "Send to bottom" to move the tile to the bottom of the list
5. "Remove from list" to delete the tile from the list.

Renaming a list

You can rename a list by double-clicking on the list title and changing the name. Click outside fo the title to save the changes

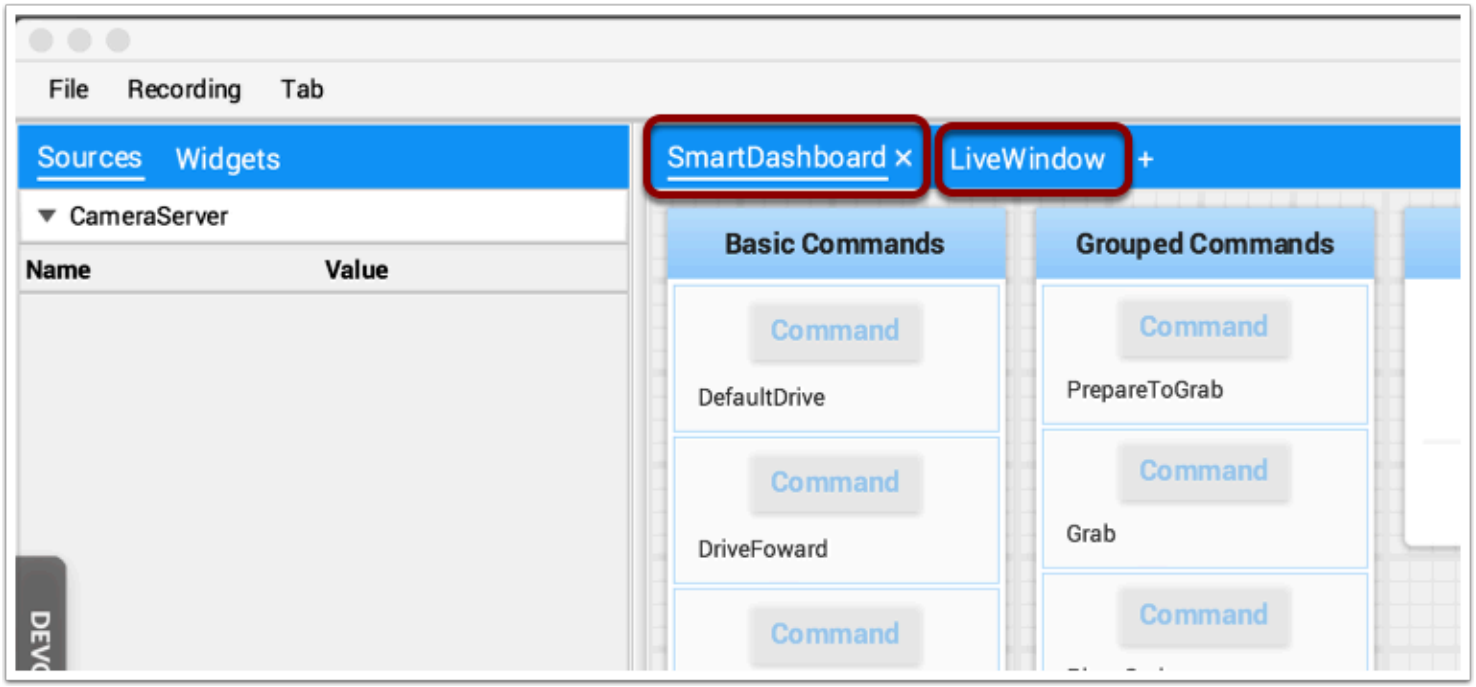


Creating and manipulating tabs

The tabbed layout the Shuffleboard uses help separate different "views" of your robot data and make the displays more useful. You might have a tab the has the display for helping debug the robot program and a different tab for use in competitions. There are a number of options that make tabs very powerful. You can control which data from NetworkTables or other sources appears in each of your tabs using the auto-populate options described later in this article.

Default tabs

When you open Shuffleboard for the first time there are two tabs, labeled SmartDashboard and LiveWindow. These correspond to the two views that SmartDashboard had depending on whether your robot is running in Autonomous/Teleop or Test mode. In shuffleboard both of these views are available any time.



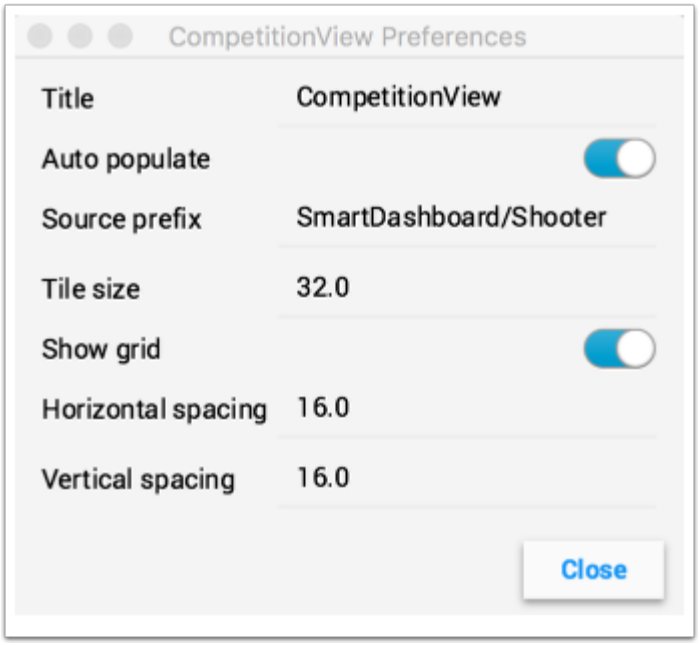
On the SmartDashboard tab all the values that are written using the SmartDashboard.putType() set of methods. On the LiveWindow tab all the autogenerated debugging values are shown.

Switching between tabs

You can switch between tabs clicking on the tab label at the top of the window. In the case above, simply click on SmartDashboard or LiveWindow to see the values that are associated with each tab.

Adding tabs and deleting tabs

You can add additional tabs by clicking on the plus(+) symbol just to the right of the last tab. Once you create a new tab you can set the label by double-clicking on the label in the tab and editing it. You can also right-click on the tab or use the Tab menu to bring up the tab preferences and from that window you can change the name by editing the Title field.



Setting the tab to auto-populate

One of the most powerful features with tabs is to have them automatically populate new values based on a source prefix that is supplied in the tab Preferences pane. In the above example the Preferences pane has a Source prefix of "SmartDashboard/Shooter" and Auto populate is turned on. Any values that are written using the SmartDashboard class that specifies a sub-key of Shooter

will automatically appear on that tab. *Note: keys that match more than one Source prefix will appear in both tabs.* Because those keys also start with SmartDashboard/ and that's the Source prefix for the default SmartDashboard tab, those widgets will appear in both panes. To only have values appear in one pane, you can use NetworkTables to write labels and values and use a different path that is not under SmartDashboard. Alternatively you could let everything appear in the SmartDashboard tab making it very cluttered, but have specific tabs for your needs that will be better filtered.

Using the tab grid and spacing

Each tab can have it's own Tile size (number of pixels per large square). So some tabs might have coarser resolution for easier layout and others might have a fine grid. The Tile size in the Tab preferences overrides any global settings in the Shuffleboard preferences. In addition, you can specify the padding between the drawing in the widget and the edge of the of the widget. If you program user interfaces these parameters are usually referred to as horizontal and vertical gap (hgap, vgap).

Moving widgets between tabs

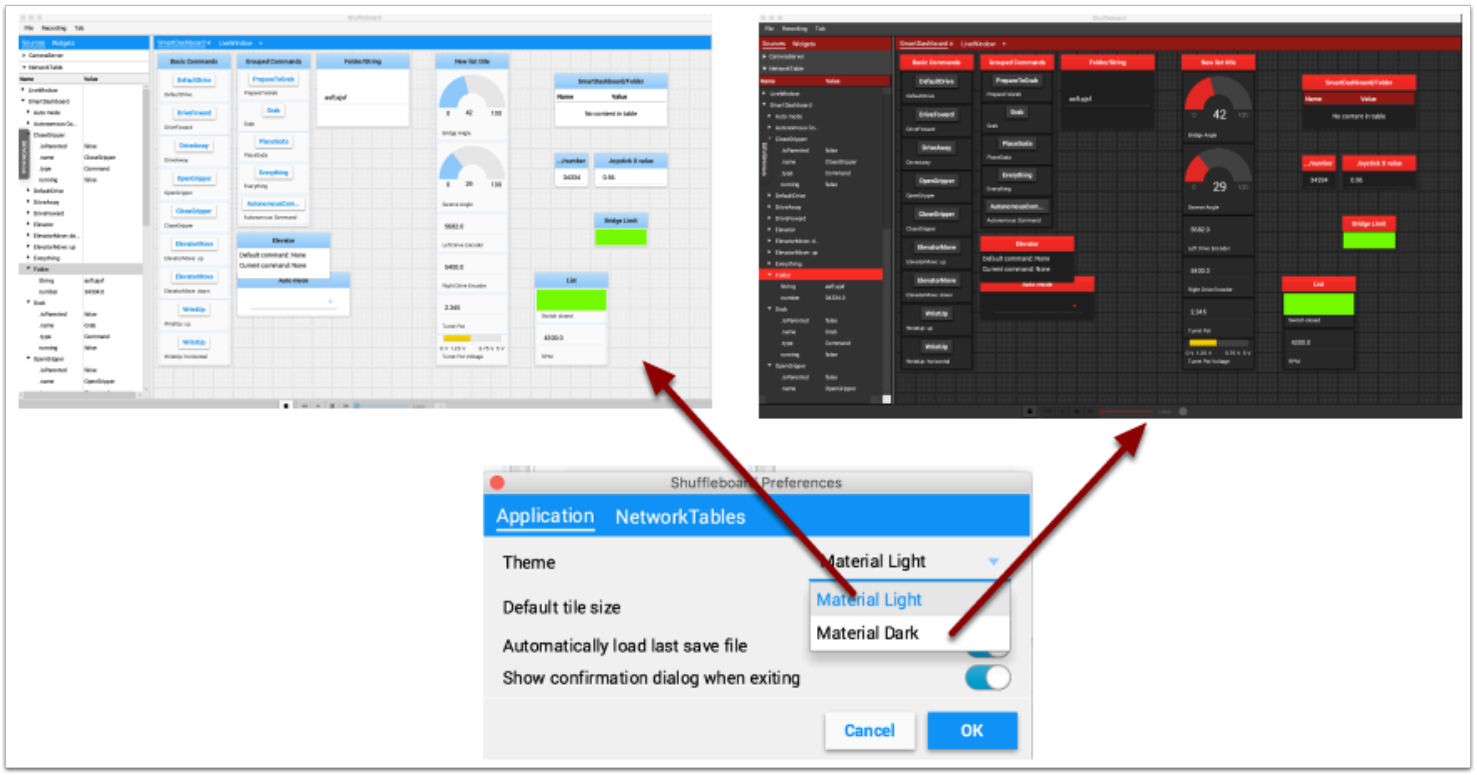
Currently there is no way to easily move widgets between tabs without deleting it from one tab and dragging the field from the sources hierarchy on the left into the new pane. We hope to have that capability in a subsequent update soon.

Setting global preferences for Shuffleboard

There are a number of settings that set the way Shuffleboard looks and behaves. Those are on the Shuffleboard Preferences pane that can be accessed from the File menu.

Setting the theme

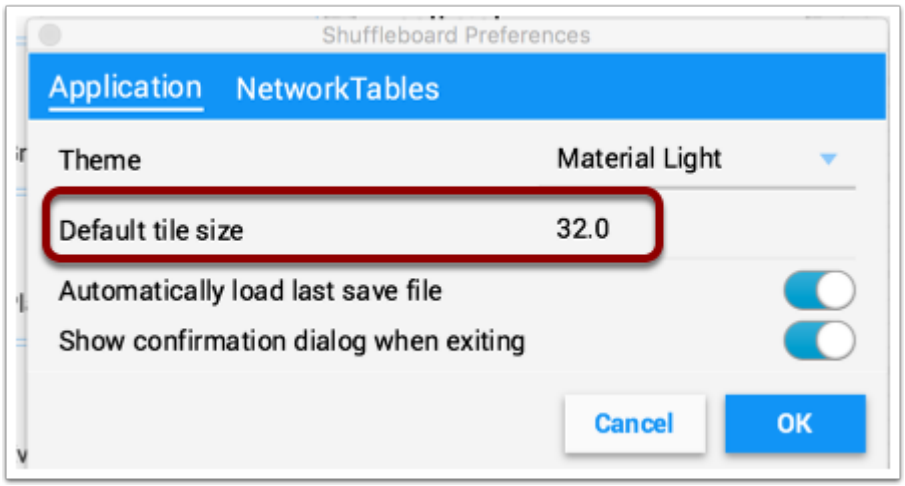
Shuffleboard supports two themes, Material Dark and Material Light and the setting depends on your preferences. This uses css styles that apply to the entire application and can be changed any time.



Setting the default tile size

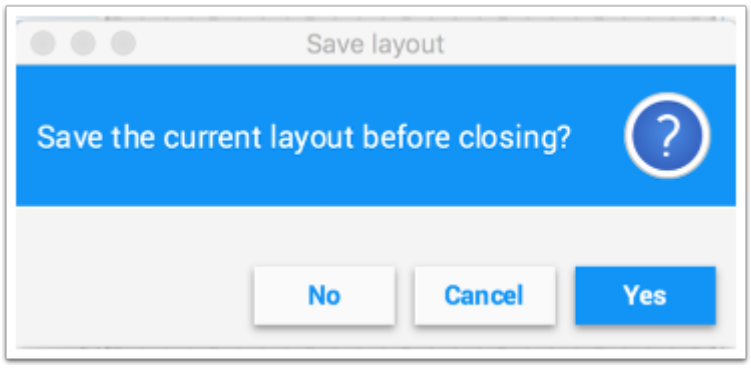
Shuffleboard positions tiles on a grid when you are adding or moving them yourself or when they are auto-populated. You can set the default tile size when for each tab or it can be set globally for all the tabs created after the default setting is changed. Finer resolution in the grid results in finer

control over placement of tiles. This can be set in the Shuffleboard Preferences window as shown below.

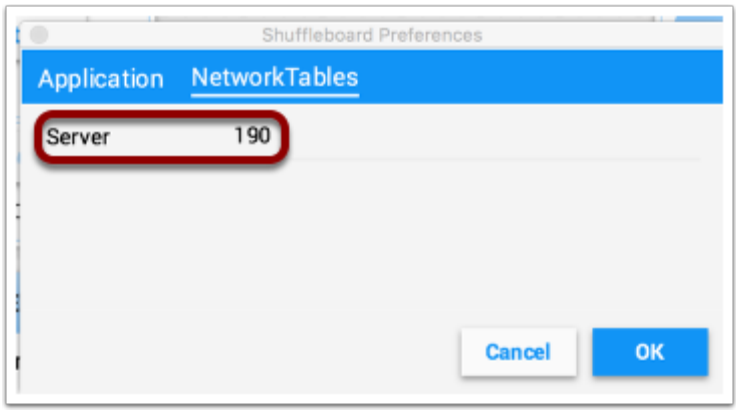


Working with the layout save files

Shuffleboard will default to saving your layout when you exit the application. You can also save it explicitly using the File / Save and File / Save as... menu options. The preferences window has options to cause the previous layout to be automatically applied when Shuffleboard starts. In addition, Shuffleboard will always display a "Save layout" window to remind you to save the layout. You can choose to turn off the automatic prompt on exit, but be sure to save the layout manually in this case so you don't loose your changes.



Setting the team number



In order for Shuffleboard to be able to find your NetworkTables server on your robot, specify your team number in the "NetworkTables" tab on the Preferences pane. If you're running Shuffleboard with a running Driver Station, the Server field will be auto-populated with the correct information. If you're running on a computer without the Driver Station, you can manually enter your team number or the robotRIO network address.

Shuffleboard FAQ, issues, and bugs

Shuffleboard as well as most of the **other control system components** were developed with Java 11 and **will not work with Java 8.** Be sure before reporting problems that your computer has Java 11 installed and is set as the default Java Environment.

Frequently Asked Questions

How do I report issues, bugs or feature requests with Shuffleboard?

Bugs, issues, and feature requests can be added on the Shuffleboard Github page by creating an issue. We will try to address them as they are entered into the system. Please try to look at existing issues before creating new ones to make sure you aren't duplicating something that has already been reported or work that is planned. You can find the issues on the shuffleboard site.

How can I add my own widgets or other extensions to Shuffleboard?

The Shuffleboard wiki has a large amount of documentation on extending the program with custom plugins. In the future we will be posting a sample plugin project that can be used for additional custom widgets, but for now the wiki documentation is complete.

How can I build Shuffleboard from the source code?

You can get the source code by downloading, cloning, or forking the repository on the GitHub site. To build and run Shuffleboard from the source, make sure that the current directory is the top level source code and use one of these commands:

Application	Command (for Windows systems run the gradlew.bat file)
Running Shuffleboard	./gradlew :app:run
Building the APIs and utility classes for plugin creation	./gradlew :api:shadowJar
Building the complete application jar file	./gradlew :app:shadowJar

Interacting with Shuffleboard from a robot program

Using tabs

Shuffleboard is a *tabbed interface.* Each tab organizes widgets in a logical grouping. By default, Shuffleboard has tabs for the legacy SmartDashboard and LiveWindow - but new tabs can now be created in Shuffleboard directly from a robot program for better organization.

Creating a new tab

```
ShuffleboardTab tab = Shuffleboard.getTab("Tab Title");
```

```
ShuffleboardTab& tab = Shuffleboard::GetTab("Tab Title");
```

Creating a new tab is as simple as calling a single method on the Shuffleboard class, which will create a new tab on Shuffleboard and return a handle for adding your data to the tab. Calling `getTab` multiple times with the same tab title will return the same handle each time.

Selecting a tab

```
Shuffleboard.selectTab("Tab Title");
```

```
Shuffleboard::SelectTab("Tab Title");
```

This method lets a tab be selected by title. This is case-sensitive (so "Tab Title" and "Tab title" are two individual tabs), and only works if a tab with that title exists at the time the method is called, so calling `selectTab("Example")` will only have an effect if a tab named "Example" has previously been defined.

This method can be used to select *any* tab in Shuffleboard, not just ones created by the robot program.

Caveats

Tabs created from a robot program differ in a few important ways from normal tabs created from the dashboard:

- Not saved in the Shuffleboard save file
- No support for autopopulation - users are expected to specify the tab contents in their robot program
- Have a special color to differentiate from normal tabs

Sending data

Unlike SmartDashboard, data cannot be sent directly to Shuffleboard without first specifying what tab the data should be placed in.

Sending simple data

Sending simple data (numbers, strings, booleans, and arrays of these) is done by calling `add` on a tab. This method will set the value if not already present, but will not overwrite an existing value.

```
Shuffleboard.getTab("Numbers")
  .add("Pi", 3.14);
```

```
Shuffleboard::GetTab("Numbers")
  .Add("Pi", 3.14);
```

If data needs to be updated (for example, the output of some calculation done on the robot), call `getEntry()` after defining the value, then update it when needed or in a `periodic` function

```
class VisionCalculator {
  private ShuffleboardTab tab = Shuffleboard.getTab("Vision");
  private NetworkTableEntry distanceEntry =
      tab.add("Distance to target", 0)
         .getEntry();

  public void calculate() {
    double distance = ...;
    distanceEntry.setDouble(distance);
  }
}
```

Making choices persist between reboots

When configuring a robot from the dashboard, some settings may want to persist between robot or driverstation reboots instead of having drivers remember (or forget) to configure the settings before each match.

Simply using `addPersistent` instead of `add` will make the value saved on the roboRIO and loaded when the robot program starts.

Note: this does not apply to sendable data such as choosers or motor controllers.

```
Shuffleboard.getTab("Drive")
  .addPersistent("Max Speed", 1.0);
```

Sending sensors, motors, etc.

Analogous to `SmartDashboard.putData`, any `Sendable` object (most sensors, speed controllers, and SendableChoosers) can be added to any tab

```
Shuffleboard.getTab("Tab Title")
  .add("Sendable Title", mySendable);
```

Retrieving data

Unlike `SmartDashboard.getNumber` and friends, retrieving data from Shuffleboard is also done through the NetworkTableEntries, which we covered in the previous article.

```
class DriveBase extends Subsystem {
  private ShuffleboardTab tab = Shuffleboard.getTab("Drive");
  private NetworkTableEntry maxSpeed =
      tab.add("Max Speed", 1)
         .getEntry();

  private DifferentialDrive robotDrive = ...;

  public void drive(double left, double right) {
    // Retrive the maximum speed from the dashboard
    double max = maxSpeed.getDouble(1.0);
    robotDrive.tankDrive(left * max, right * max);
  }
}
```

This basic example has a glaring flaw: the maximum speed can be set on the dashboard to a value outside [0, 1] - which could cause the inputs to be saturated (always at maximum speed), or even reversed! Fortunately, there is a way to avoid this problem - covered in the next article.

Configuring widgets

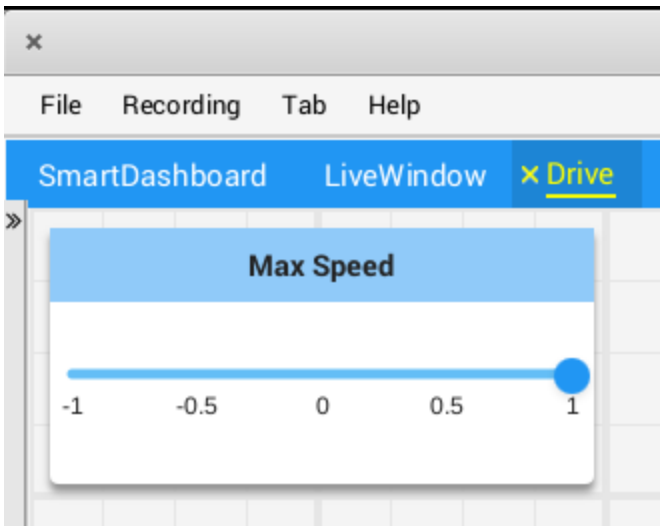
Robot programs can specify exactly which widget to use to display a data point, as well as how that widget should be configured. As there are too many widgets to be listed here, consult the docs for details.

Specifying a widget

Call `withWidget` after `add` in the call chain:

```
Shuffleboard.getTab("Drive")
  .add("Max Speed", 1)
  .withWidget(BuiltInWidgets.kNumberSlider) // specify the widget here
  .getEntry();
```

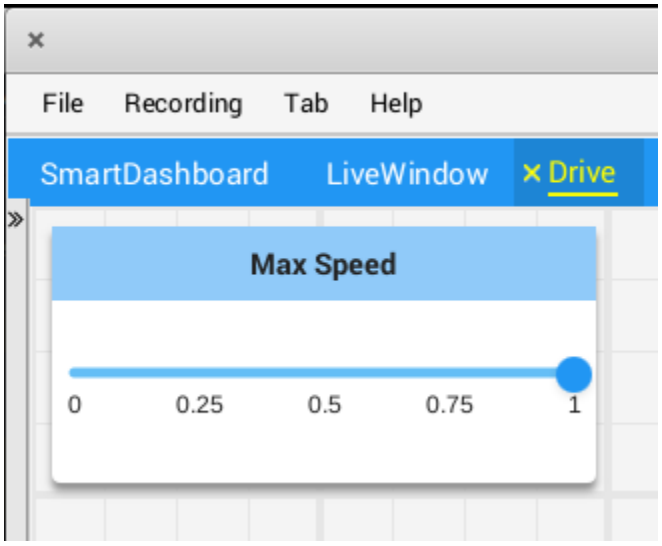
In this example, we configure the "Max Speed" widget to use a slider to modify the values instead of a basic text field.



Setting widget properties

Since the maximum speed only makes sense to be a value from 0 to 1 (full stop to full speed), a slider from -1 to 1 can cause problems if the value drops below zero. Fortunately, we can modify that using the `withProperties` method

```
ShuffShuffleboard.getTab("Drive")
    .add("Max Speed", 1)
    .withWidget(BuiltInWidgets.kNumberSlider)
    .withProperties(Map.of("min", 0, "max", 1)) // specify widget properties here
    .getEntry();
```



Notes

Widgets can be specified by name; however, names are case- and whitespace-sensitive ("Number Slider" is different from "Number slider" and "NumberSlider"). For this reason, it is recommended to use the built in widgets class to specify the widget instead of by raw name. However, a custom widget can only be specified by name or by creating a custom `WidgetType` for that widget.

Widget property names are neither case-sensitive nor whitespace-sensitive ("Max" and "max" are the same). Consult the documentation on the widget in the BuiltInWidgets class for details on the properties of that widget.

Organizing widgets

Setting widget size and position

Call `withSize` and `withPosition` to set the size and position of the widget in the tab.

`withSize` sets the number of columns wide and rows high the widget should be. For example, calling `withSize(1, 1)` makes the widget occupy a single cell in the grid. Note that some widgets have a minimum size that may be greater than the specified size, in which case the widget will use the smallest supported size.

`withPosition` sets the row and column of the top-left corner of the widget. Rows and columns are both 0-indexed, so the topmost row is number 0 and the leftmost column is also number 0. If the position of any widget in a tab is specified, every widget should also have its position set to avoid overlapping widgets.

```
Shuffleboard.getTab("Pre-round")
  .add("Auto Mode", autoModeChooser)
  .withSize(2, 1) // make the widget 2x1
  .withPosition(0, 0); // place it in the top-left corner
```

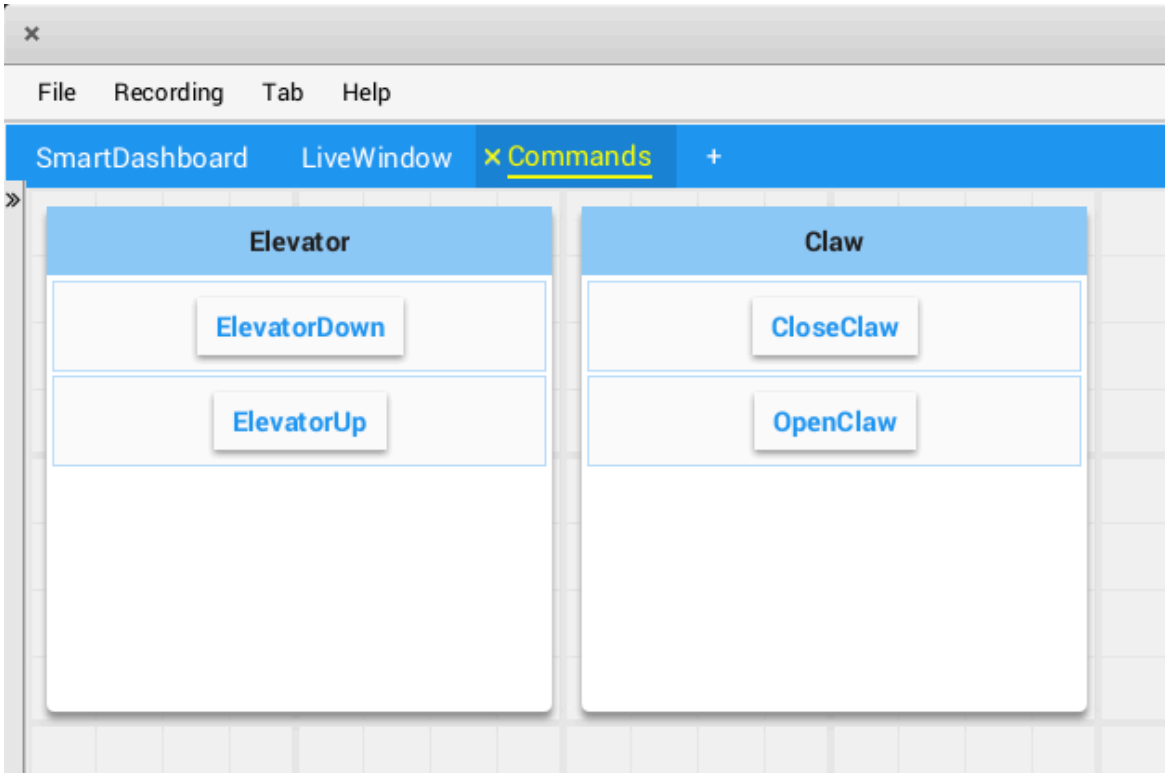
Adding widgets to layouts

If there are many widgets in a tab with related data, it can be useful to place them into smaller subgroups instead of loose in the tab. Much like how the handle to a tab is retrieved with `Shuffleboard.getTab`, a layout inside a tab (or even in another layout) can be retrieved with `ShuffleboardTab.getLayout`.

Just like widgets, layouts can have their custom properties, size, and position specified from the robot program.

```
ShuffleboardLayout elevatorCommands = Shuffleboard.getTab("Commands")
  .getLayout("Elevator", BuiltInLayouts.kList)
  .withSize(2, 2)
  .withProperties(Map.of("Label position", "HIDDEN")); // hide labels for commands
elevatorCommands.add(new ElevatorDownCommand());
elevatorCommands.add(new ElevatorUpCommand());
```

```
// Similarly for the claw commands
```



Controlling data recording

Starting in the 2019 season, Shuffleboard will **not** record data by default. Recording can be manually controlled through the dashboard, or directly from the robot.

Starting recording at startup

If you want to start recording data as soon as possible (such as to collect data on gyro and accelerometer drift, or keep track of pre-round autonomous selections), call `startRecording()` in `robotInit()`

Note that this method has no effect if Shuffleboard is already recording data.

```
class Robot extends TimedRobot {
  @Override
  public void robotInit() {
    Shuffleboard.startRecording();
  }
}
```

Marking important events

Shuffleboard supports marking important events in recordings. These can be any kind of event - scoring a game piece, acquiring a vision target, or changing match periods, just to name a few.

Events have a name, an optional description, and an importance level. The name of an event should be terse; if more detail is required, add a description for the event. The importance is a subjective measure of how important the event, but in general less important information (such as a minor sensor deviation) should have a lower importance level than more critical information (such as dangerously high current draw on a motor). See the documentation on the `EventImportance` class for further detail on the available importance levels.

```
class Robot extends TimedRobot {
  @Override
  public void robotInit() {
    Shuffleboard.addEventMarker("Robot initialized", EventImportance.kTrivial);
```

```
  }

  @Override
  public void autonomousInit() {
    Shuffleboard.addEventMarker("Match start", EventImportance.kNormal);
  }
}
```

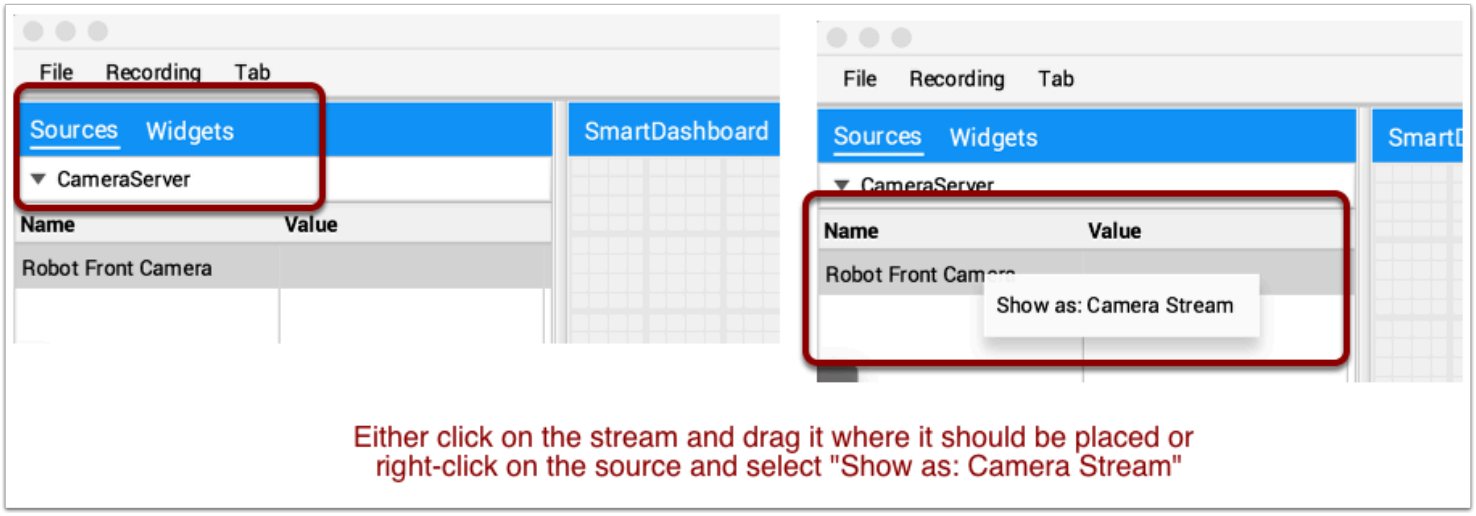
Advanced applications

Displaying camera streams

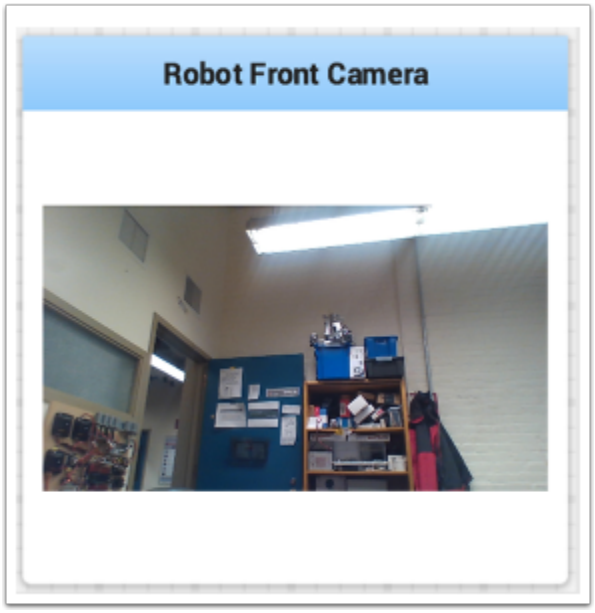
Camera streams from the robot can be viewed on a tab in Shuffleboard. This is useful for viewing what the robot is seeing to give a less obstructed view for operators or helping visualize the output from a vision algorithm running on the driver station computer or a coprocessor on the robot. Any stream that is running using the CameraServer API can be viewed in a camera stream widget.

Adding a camera stream

To add a camera to your dashboard select "Sources" and view the "CameraServer" source in the left side panel in the Shuffleboard window as shown in the example below. A list of camera streams will be shown, in this case there is only one camera called "Robot Front Camera". Drag that to the tab where it should be displayed. Alternatively the stream can also be placed on the dashboard by right-clicking on the stream in the Sources list and selecting "Show as: Camera Stream".



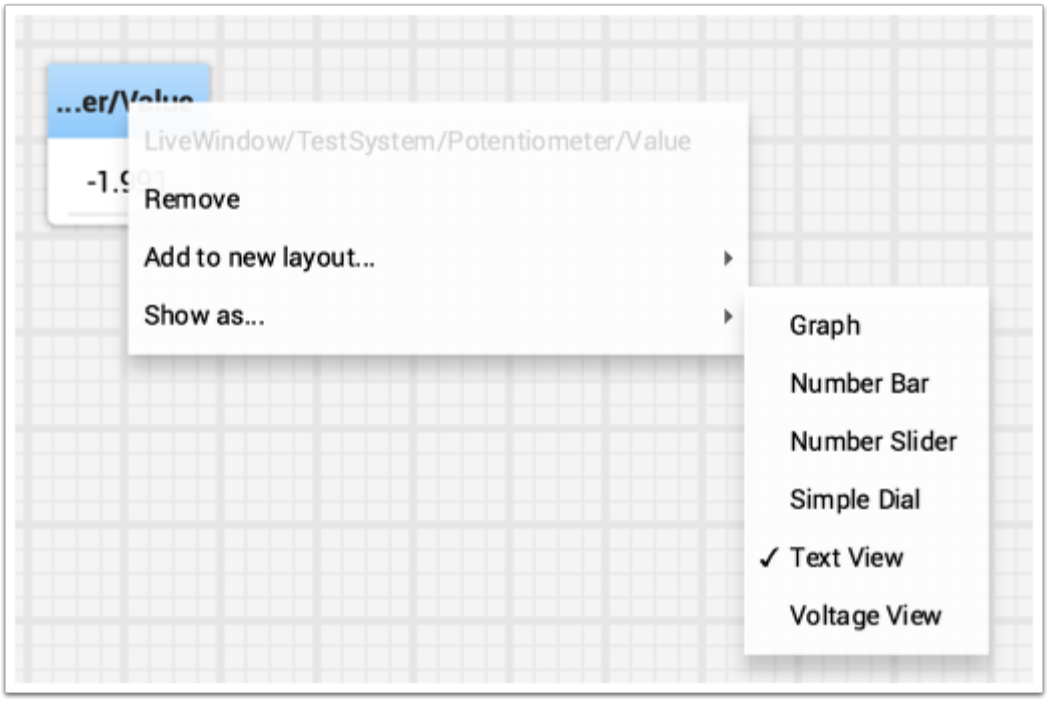
Once the camera stream is added it will be displayed in the window. It can be resized and moved where you would like it. *Be aware that sending too much data from too high a resolution or too high a frame rate will cause high CPU usage on both the roboRIO and the laptop.*


Robot Front Camera

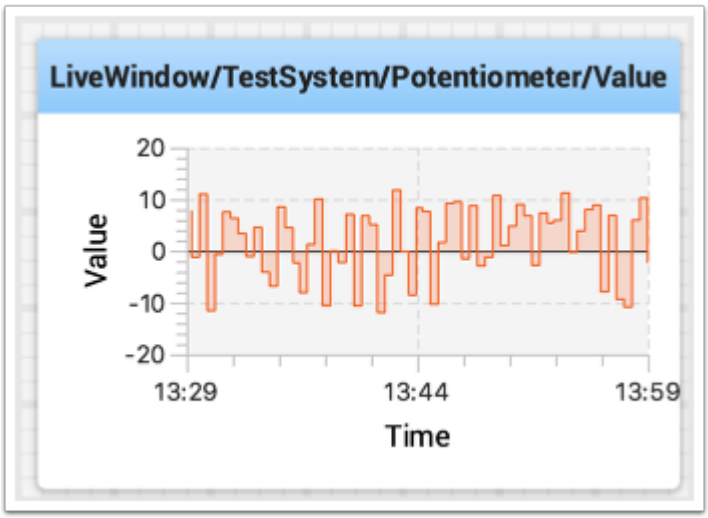
Working with graphs

With Shuffleboard you can graph numeric values over time. Graphs are very. useful to see how sensor or motor values are changing as your robot is operating. For example the sensor value can be graphed in a PID loop to see how it is responding during tuning.

To create a graph, choose a numeric value and right-click in the heading and select "Show as..." and then choose graph.

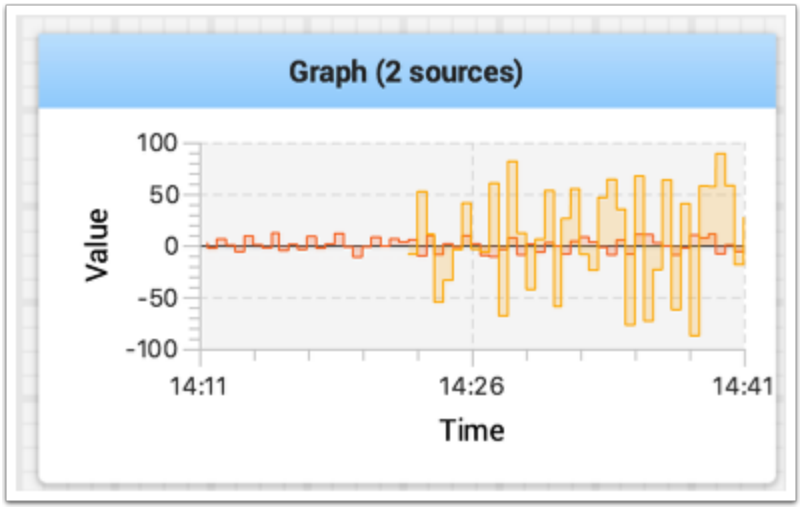


The graph widget shows line plots of the value that you selected. It will automatically set the scale and the default time interval that the graph will show will be 30 seconds. You can change that in the setting for the graph (see below).

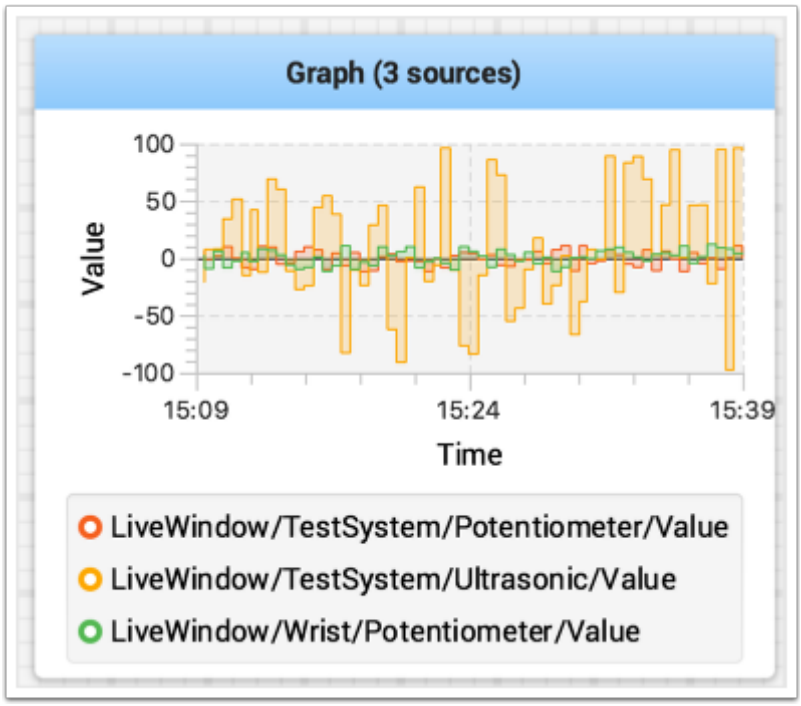


Adding an additional data value to the graph

For related values it is often desirable to show multiple values on the same graph. To do that, simply drag additional values from the NetworkTable source view (left side of the Shuffleboard window) and drop it onto the graph and that value will be added as shown below. You can continue to drag additional values onto the graph.



You can resize the graph vertically to view the legend if it is not displayed as shown in the image below. The legend shows all the sources that are used in the plot.

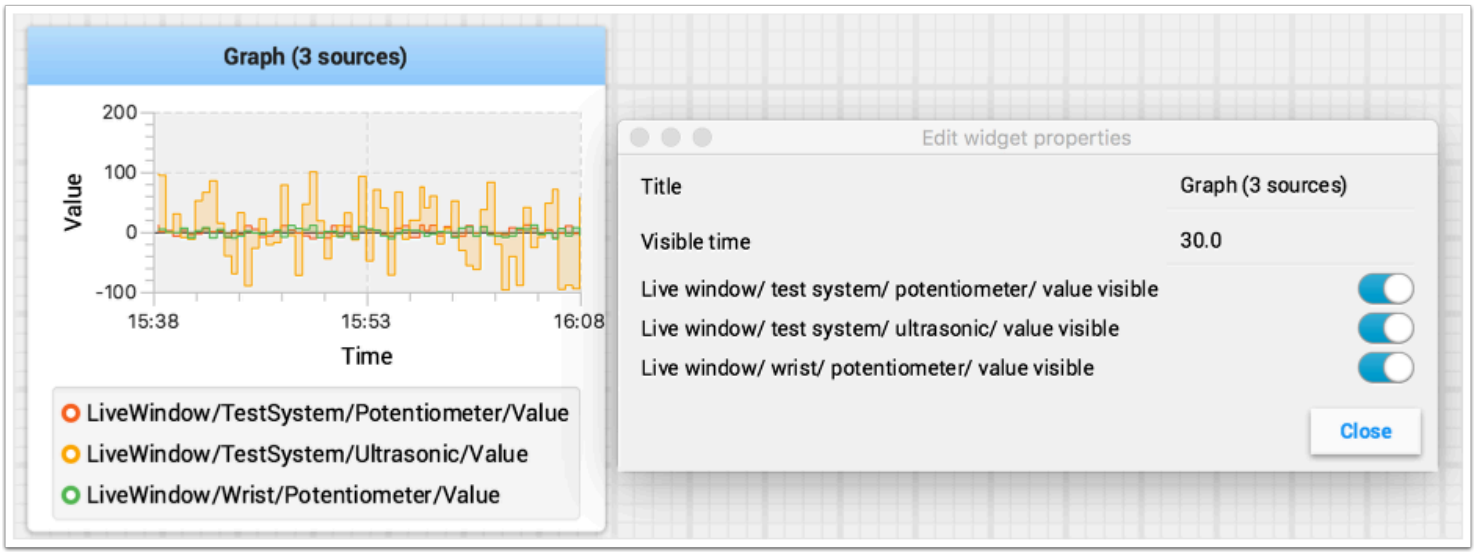


Setting properties for the graph

You can set the number of seconds that are shown in the graph by changing the "Visible time" in the graph widget properties. To access the properties, right-click on the graph and select "Edit properties".

In addition to setting the visible time the graph can selectively turn sources on and off by turning the switch on and off for each of the sources shown in the properties window (see below).

Shuffleboard

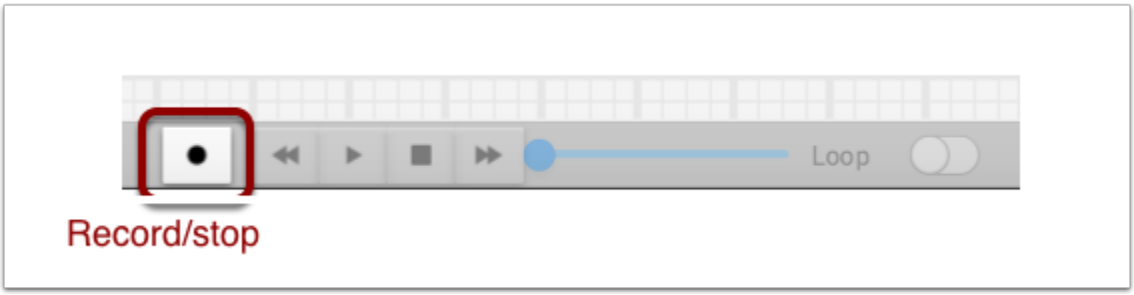


Using record and playback

Shuffleboard can log all widget updates during a session. Later the log file can be "played back" to see what happened during a match or a practice run. This is especially useful if something doesn't operate as intended during a match and you want to see what happened. Each recording is captured in a recording file.

Creating a recording

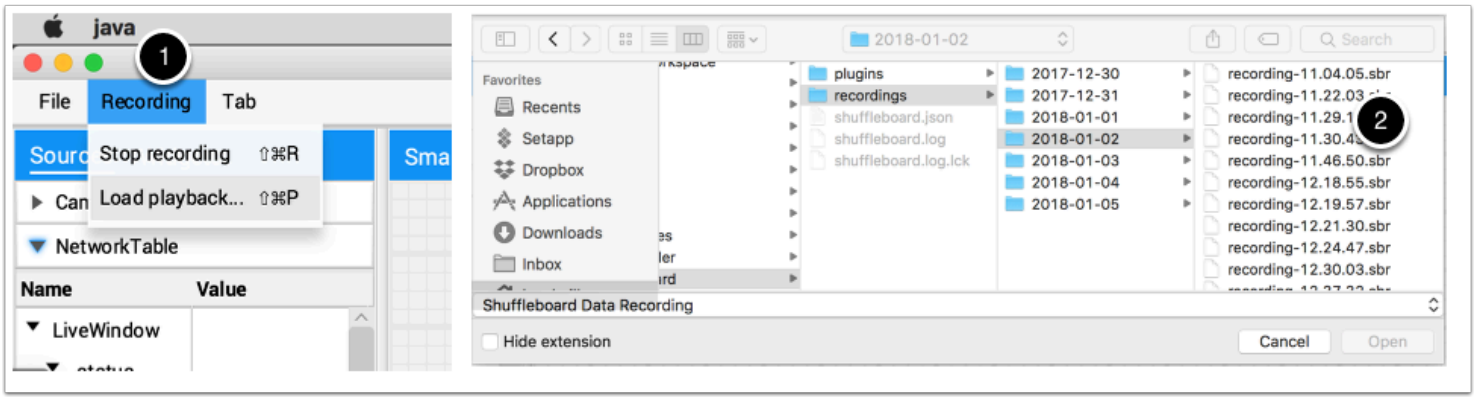
When shuffleboard starts it begins recording to all the NetworkTable values are recorded and continues until stopped by hitting the record/stop button in the recorder controls as shown below. If a new recording is desired, such as when a new piece of code or mechanical system is being tested, stop the current recording if it is running, and click the record button. Click the button again to stop recording and close the recording file. If the button is round (as shown) then click it to start a recording. If the button is a square, then a recoding is currently running so click it to stop the recording.



Playing back a recording

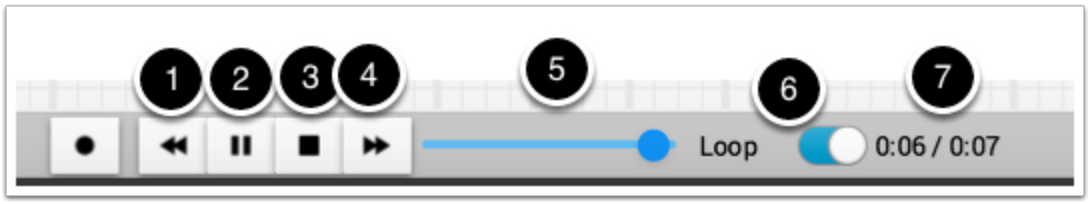
Previous recordings can be played back by:

1. selecting the "Recording" menu then click "Load playback".
2. Choose a recording from the from the directory shown. Recordings are grouped by date and the file names are the time the recording was made to help identify the correct one. Select the correct recording from the list.



Controlling the playback

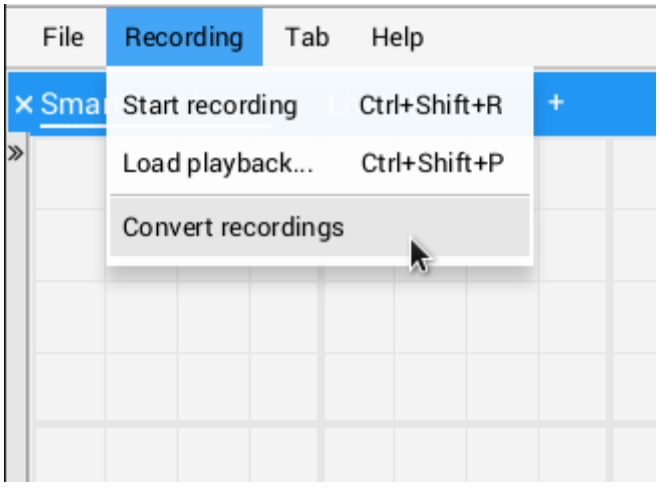
Selecting the recoding file will begin playback of that file. While the recording is playing the recording controls will show the current time within the recording as well as the option to loop the recording while watching it. When the recording is being played back the "transport" controls will allow the playback to be controlled.



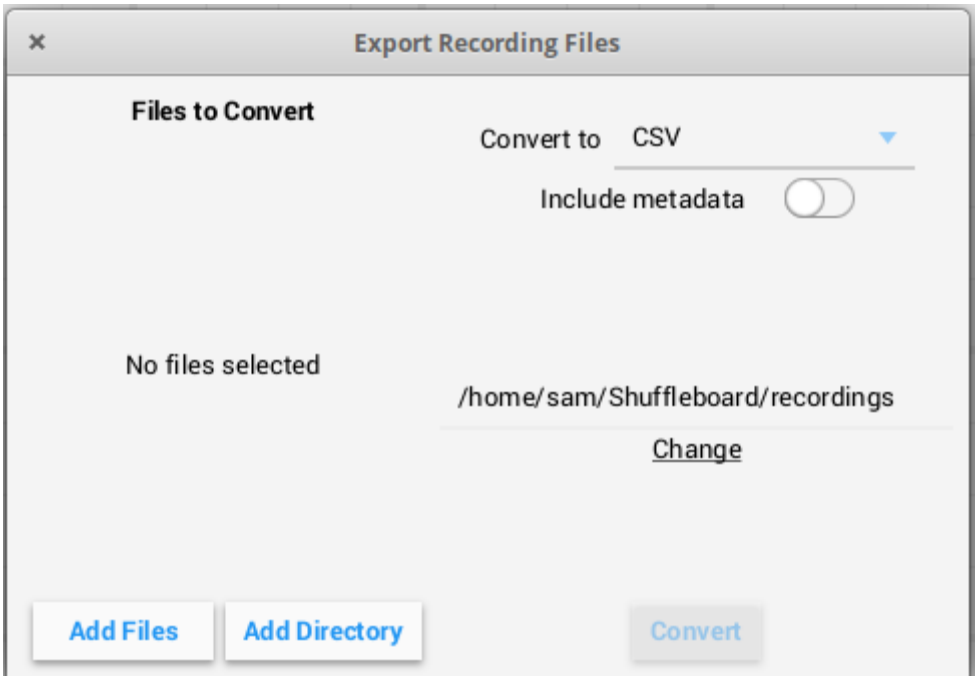
The controls work as follows:

1. The left double-arrow button backs up the playback to the last changed data point
2. The play/pause controls starts and stops the playback
3. The square stop button stops playback and resumes showing current robot values
4. The right double-arrow skips forward to the next changed data value
5. The scrubber is a slider that allow direct positioning to any point in time to view different parts of the recording
6. The loop switch turns on playback looping, that is, the playback will run over and over until stopped
7. The time shows the current point within the recording and the total time of the recording

Converting a recording file to a different format



Shuffleboard recordings are in a custom binary format for efficiency. To analyze recorded data without playing it back through the app, Shuffleboard supports data converters to convert the recordings to an arbitrary format. Only a simple CSV converter is shipped with the app, but teams can write custom converters and include them in Shuffleboard plugins.



Multiple recordings can be converted at once. Individual files can be selected with the "Add Files" button, or all recording files in a directory can be selected at once with the "Add Directory" button.

Converted recordings will be generated in the `~/Shuffleboard/recordings` directory, but can be manually selected with the "Change" button.

Different converters can be selected with the dropdown in the top right. By default, only the CSV converter is available. Custom converters from plugins will appear as options in the dropdown.

Additional notes

Graphs won't display properly while scrubbing the timeline but if it is playing through where the graph history can be captured by the graph then they will display as in the original run.

Working with Commands and Subsystems

When using the Command-based framework Shuffleboard makes it easier to understand what the robot is doing by displaying the state of various commands and subsystems in real-time.

Displaying subsystems

To see the status of a subsystem while the robot is operating in either automous or telop modes, that is what it's default command is and what command is currently using that subsystem send a subsystem instance to Shuffleboard:

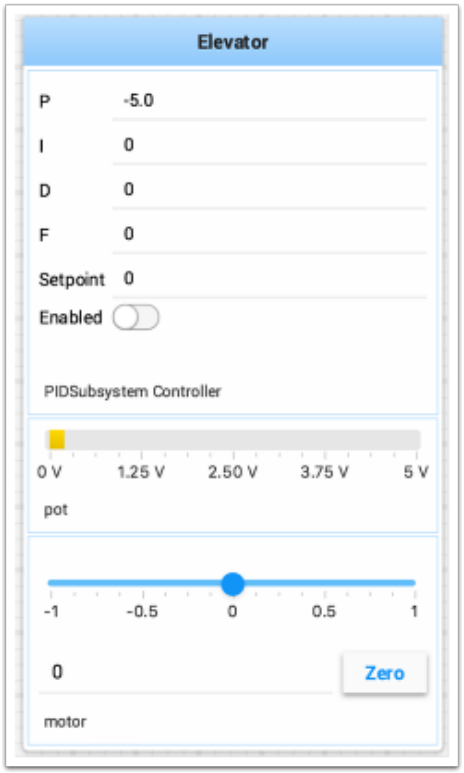
```
Java:
SmartDashboard.putData(subsystem-reference);

C++:
SmartDashboard::PutData(subsystem-pointer);
```

Shuffleboard will display the subsystem name, the default command associated with this subsystem, and the currently running command. In this example the default command for the Elevator subsystem is called "AutonomousCommand" and it is also the current command that is using the Elevator subsystem.

Testing and viewing subsystems in Test mode

In Test mode (Test/Enabled in the driver station) subsystems maybe displayed in the LiveWindow tab with the sensors and actuators of the subsystem. This is ideal for verifying of sensors are

working and seeing the values that they are returning. In addition the actuators can be operated, for example motors can be operated using sliders to set the speed and direction. For PIDSubsystems the P, I, D, and F constants are displayed along with the setpoint and an enable control. This is useful for tuning PIDSubsystems by adjusting the constants, putting in a setpoint, and enabling the embedded PIDController. The the mechanisms response can be observed. Then the parameters can be changed, the PIDController re-enabled until a reasonable set of parameters is found.



More information on tuning PIDSubsystems can be found here: [Testing and tuning PID loops.] Using RobotBuilder will automatically generate the code to get the subsystem displayed in Test mode. The code that is necessary to have subsystems displayed is shown below where *subsystem-name* is a string containing the name of the subsystem:

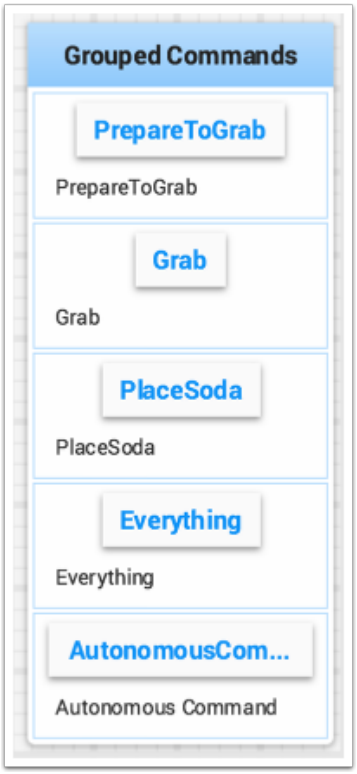
```
setName(subsystem-name);
```

Displaying commands

Using commands and subsystems makes very modular robot programs that can easily be tested and modified. Part of this is because commands can be written completely independently of other commands and can therefor be easily run from Shuffleboard. To write a command to Shuffleboard use the SmartDashboard.putData method as shown here:

```
SmartDashboard.putData("ElevatorMove: up", new ElevatorMove(2.7);
```

Shuffleboard will display the command name and a button to execute the command. In this way individual commands and command groups can easily be tested without needing special test code in a robot program. In the image below there are a number of commands contained in a Shuffleboard list. Pressing the button once runs the command and pressing it again stops the command. To use this feature the robot must be enabled in teleop mode.



Testing and tuning PID loops

One challenge in using sensors to control mechanisms is to have a good algorithm to drive the motors to the proper position or speed. The most commonly used control algorithm is called PID control. There is a good set of videos (look for the robot controls playlist) that explain the control algorithms described here The PID algorithm converts sensor values into motor speeds by:

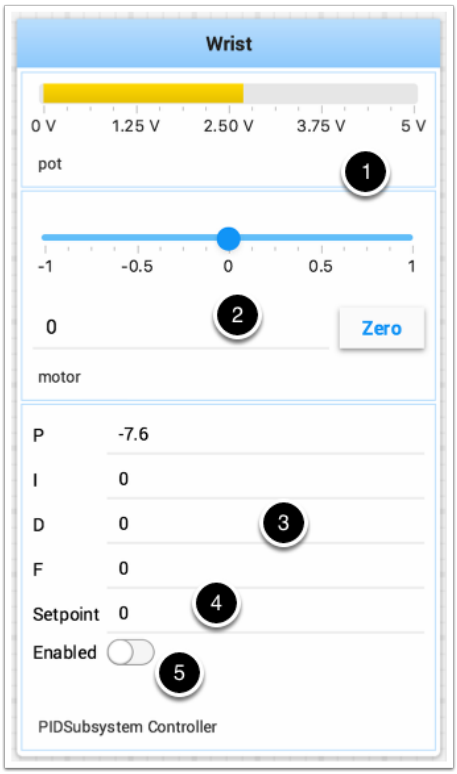
1. Reading sensor values to determine how far the robot or mechanism from the desired setpoint. The setpoint is the sensor value that corresponds to the expected goal. For example, a robot arm with a wrist joint should be able to move to a specified angle very quickly and stop at that angle as indicated by a sensor. A potentiometer is a sensor that can measure. rotational angle. By connecting it to an analog input, the program can get a voltage measurement that is directly proportional to the angle.
2. Compute an error (the difference between the sensor value and the desired value). The sign of the error value indicates which side of the setpoint the wrist is on. For example negative values might indicate that the measured wrist angle is larger than the desired wrist angle. The magnitude of the error is how far the measured wrist angle is from the actual wrist angle. If the error is zero, then the measured angle exactly matches the desired angle. The error can be used as an input to the PID algorithm to compute a motor speed.
3. The resultant motor speed is them used to drive the motor in the correct direction and a speed that hopefully will reach the setpoint as quickly as possible without overshooting (moving past the setpoint).

WPILib has a PIDController class that implements the PID algorithm and accepts constants that correspond to the Kp, Ki, and Kd values. The PID algorithm has three components that contribute to computing the motor speed from the error.

1. P (proportional) - this is a term that when multiplied by a constant (Kp) will generate a motor speed that will help move the motor in the correct direction and speed.
2. I (integral) - this term is the sum of successive errors. The longer the error exists the larger the integral contribution will be. It is simply a sum of all the errors over time. If the wrist isn't quite getting to the setpoint because of a large load it is trying to move, the integral term will continue to increase (sum of the errors) until it contributes enough to the motor speed to get it to move to the setpoint. The sum of the errors is multiplied by a constant (Ki) to scale the integral term for the system.
3. D (differential) - this value is the rate of change of the errors. It is used to slow down the motor speed if it's moving too fast. It's computed by taking the difference between the current error value and the previous error value. It is also multiplied by a constant (kd) to scale it to match the rest of the system.

Tuning the PID controller

Tuning the PID controller consists of picking constants that will give good performance. Shuffleboard helps this process by displaying the details of a PID subsystem with a user interface for setting constant values and testing how well it operates. This is displayed while the robot is operating in test mode (done by setting "Test" in the driver station).



This is the test mode picture of a wrist subsystem that has a potentiometer as the sensor (pot) and a speed controller connected to the motor. It has a number of areas that correspond to the PIDSubsystem.

1. The analog input voltage value from the potentiometer. This is the sensor input value.
2. A slider that moves the wrist motor in either direction with 0 as stopped. The positive and negative values correspond to moving up or down.
3. The PID constants as described above (F is a feedforward value that is used for speed PID loops)
4. The setpoint value that corresponds the to the pot value when the wrist has reached the desired value

5. Enables the PID controller - that is, starts it looping at regular intervals reading the pot value, computing the error, applying the P, I, and D terms and setting the motor value.

Try various values to get the desired motor performance. You can look at the video linked to at the beginning of this article or other sources on the internet to get the desired performance.

Viewing hierarchies of data

Dragging a key with other keys below it (deeper in the hierarchy) displays the hierarchy in a tree, similar to the NetworkTables sources on the left.

Currently this feature is unavailable, but we expect to have it soon in an update.