

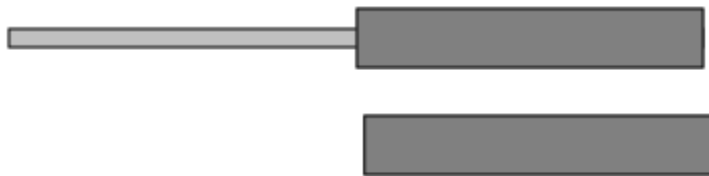
Creating a custom control using FXML

sfx comes with a palette of built-in controls that feature a wide range of use cases. But sometimes you would like to further customize your robot dashboard with controls that you create yourself. There are two strategies for creating custom controls, either:

1. FXML - a XML-based markup language for describing your own controls using a declarative language without needing programming
2. Java-based controls can have more complex requirements and behaviors

In this lesson we'll look at creating FXML-based controls. For creating Java controls, see the [Java tutorial](#).

Creating a pneumatic piston position indicator using FXML



Suppose you need to display the position of a pneumatic piston to make it clear to the operators the state of the piston. Ideally you would draw the piston and show the piston rod either in or out of the case. Showing it graphically might be much clearer than just having an indicator that was either red or green. FXML allows you to create more complex drawings and animate them based on, in this case, a boolean value. The illustration above shows the piston in the in and out position.

In this example we use a 2x3 grid pane to model the piston. When it is extended, the left middle cell contains a light grey panel that is visible. When it is retracted, that panel is hidden.

Creating a custom control using FXML

Creating the FXML file

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <?import java.lang.*?>
4 <?import java.util.*?>
5 <?import javafx.geometry.*?>
6 <?import javafx.scene.*?>
7 <?import javafx.scene.control.*?>
8 <?import javafx.scene.layout.*?>
9 <?import javafx.scene.shape.*?>
10
11 <dashfx.controls.bases.BooleanControlBase fx:id="base" xmlns:fx="http://javafx.com/fxml">
12   <ui>
13     <GridPane prefHeight="63.0" prefWidth="488" xmlns:fx="http://javafx.com/fxml">
14       <children>
15         <Pane visible="{base.value}" prefHeight="200.0" prefWidth="200.0" style="-fx-back
16         <Pane prefHeight="200.0" prefWidth="200.0" style="-fx-background-color: gray;-fx-b
17       </children>
18     </GridPane>
19     <columnConstraints>
20       <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />
21       <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />
22     </columnConstraints>
23     <rowConstraints>
24       <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
25       <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
26       <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
27     </rowConstraints>
28   </ui>
29 </dashfx.controls.bases.BooleanControlBase>
30
```

FXML is a declarative markup language for describing the graphical properties of your widget. It lets you specify values for shape, color, position or other properties to describe your widget.

FXML controls have the following structure:

1. Header
2. Base
3. UI & Bindings

The header is made of the XML header and JVM imports to avoid fully qualifying all nodes

```
<?xml version="1.0" encoding="UTF-8"?>
<?import java.lang.*?>
<?import java.util.*?>
<?import javafx.geometry.*?>
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
```

Creating a custom control using FXML

```
<?import javafx.scene.layout.*?>
<?import javafx.scene.shape.*?>
```

The **base** is required for SFX controls and abstracts the data management away. The following are valid bases:

- BooleanControlBase - any boolean-like values (true/false, 1/0, etc...)
- NumberControlBase - any floating point or integral value. Also coerces numbers in strings.
- RangedNumberControlBase - The same as a NumberControlBase but with a min and max value
- StringControlBase - any string value or the toString of any other values

In this case we need a boolean so we use the boolean control base (note we assign it an ID so we can bind to it later):

```
<dashfx.controls.bases.BooleanControlBase fx:id="base" xmlns:fx="http://javafx.com/fxml">
</dashfx.controls.bases.BooleanControlBase>
```

All current bases have a **ui** property that describes what to show on screen, and this is where we put all the **ui**:

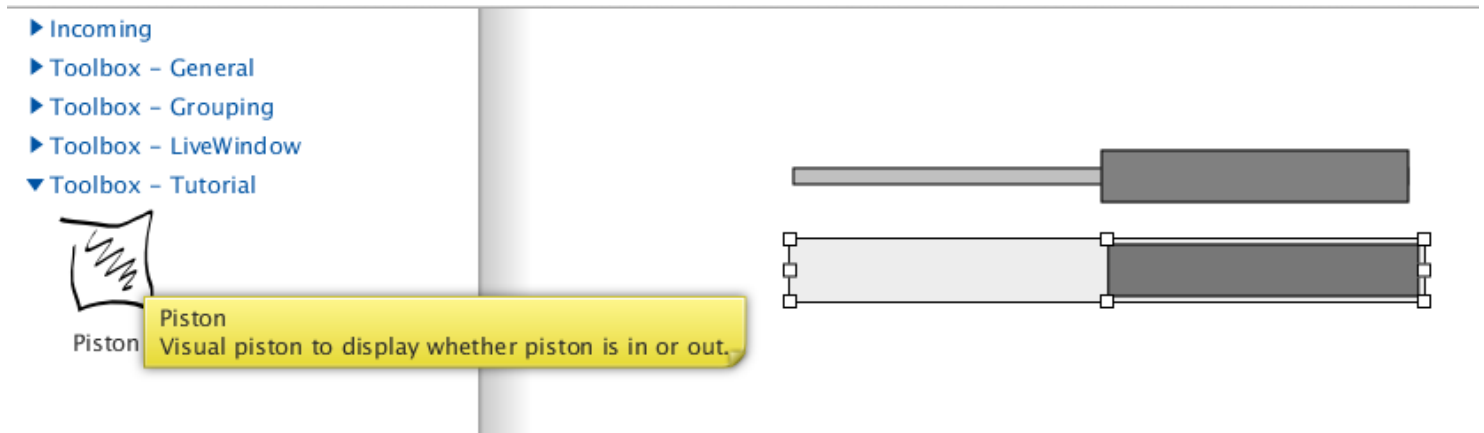
```
<ui>
  <GridPane prefHeight="50" prefWidth="500">
    <children>
      <Pane visible="{base.value}"
        prefHeight="200.0" prefWidth="200.0"
        style="-fx-background-color: silver;-fx-border-color: black;"
        GridPane.columnIndex="0"
        GridPane.columnSpan="2147483647"
        GridPane.rowIndex="1" />
      <Pane prefHeight="200.0" prefWidth="200.0"
        style="-fx-background-color: gray;-fx-border-color: black;"
        GridPane.columnIndex="1"
        GridPane.rowIndex="0"
        GridPane.rowSpan="2147483647" />
    </children>
    <columnConstraints>
      <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />
      <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />
    </columnConstraints>
  </GridPane>
</ui>
```

Creating a custom control using FXML

```
</columnConstraints>
<rowConstraints>
  <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
  <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
  <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
</rowConstraints>
</GridPane>
</ui>
```

This contains normal FXML and was built with the JavaFX Scene Builder. Note the first pane's visibility property is bound to the base's value, thus enabling the piston to appear retracted when the value is false, and extended when true. This is all the FXML needed to define this control.

Registering your control with a manifest



To add your control to the dashboard, you need to package it in a plugin. For FXML controls you can simply put it in a folder or pack it in a jar. Either way, we need a manifest file that describes what the plugin contains. Manifests are written in YAML and can contain multiple controls and other options. For our needs, we will start with this file (please generate your own UUID. uuidgenerator.net is where the provided UUID was generated)

```
API: 0.1
Name: Tutorial plugin
Description: Contains all the plugins from the tutorial
Version: 1.0.0
# Please generate your own unique UUID and replace it below
Plugin ID: b673b0fe-716a-40ce-b446-70e34aafc509
Controls:
```

Creating a custom control using FXML

```
-  
Name: Piston  
Description: Visual piston to display whether piston is in or out.  
Source: /piston.fxml  
Category: Tutorial  
Defaults:  
  value: false
```

This says:

- we are using plugin API version 0.1 (the current version)
- the name of the plugin is "Tutorial plugin" which can be identified by the UUID
- it has one control, the Piston control, which is in the Tutorial category
- it's described by the given fxml file.

Note most fields are optional and many have been left off this simple example. Save the manifest as manifest.yml. NOTE: YAML expects spaces for indentation, not tabs.

Now register this with SFX. Create a new folder in the sfx/plugins folder in your copy of sfx with the following layout:

```
tutorial  
  manifest.yml  
  piston.fxml
```

Now when you start sfx, you should be able to use the control. It is recommended to launch from the terminal with the command:

```
java -jar sfx.jar
```

in case there are any errors. All plugins can be viewed under settings>plugins.