FLOWBOTICS STUDIO

PROGRAMMING GUIDE

RobotShop.com
Putting robotics at your service™

Contents

1 Introduction

Going Beyond Sequencing

About This Guide

Before reading this guide you should read the guide for at least one of the projects. In those guides you'll learn how to use the sequencer and see how to create patterns of movement.

To give your robot a bit more intelligence you need to apply a bit of programming and allowing it to respond to external input that extends beyond just pressing the Play button.

In this guide we'll describe how you can make your robot respond to external stimuli such as controllers or inputs on the SSC-32 board.

2 Getting Started

Preparing To Edit a Project

Preparing a Project

To do any kind of programming with the project you'll need to use the FlowStone editor. You will recall from the FlowBotics studio guide that you can edit a project by going to the Project Browser, selecting the project and clicking on the Edit Selected button on the toolbar.

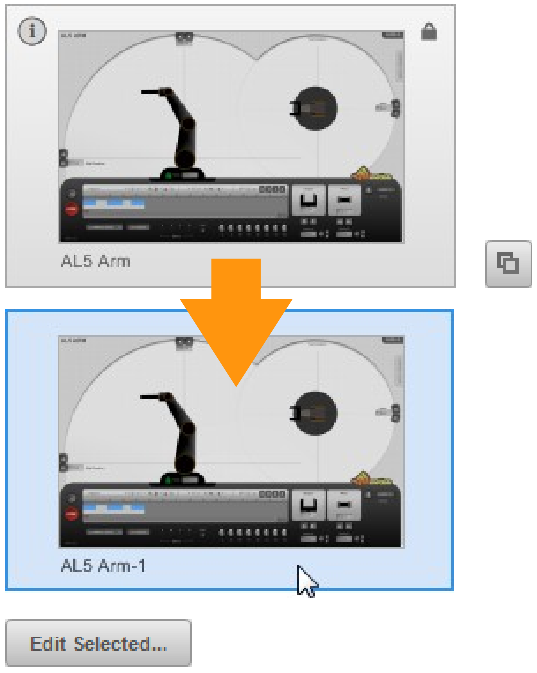
Creating a Copy

Before you do this you'll need to duplicate the project as the original is locked. By duplicating the project you make your own copy that you can modify.

STEP 1

STEP 2

STEP 3



3 The User Programming Area

Adding Intelligence

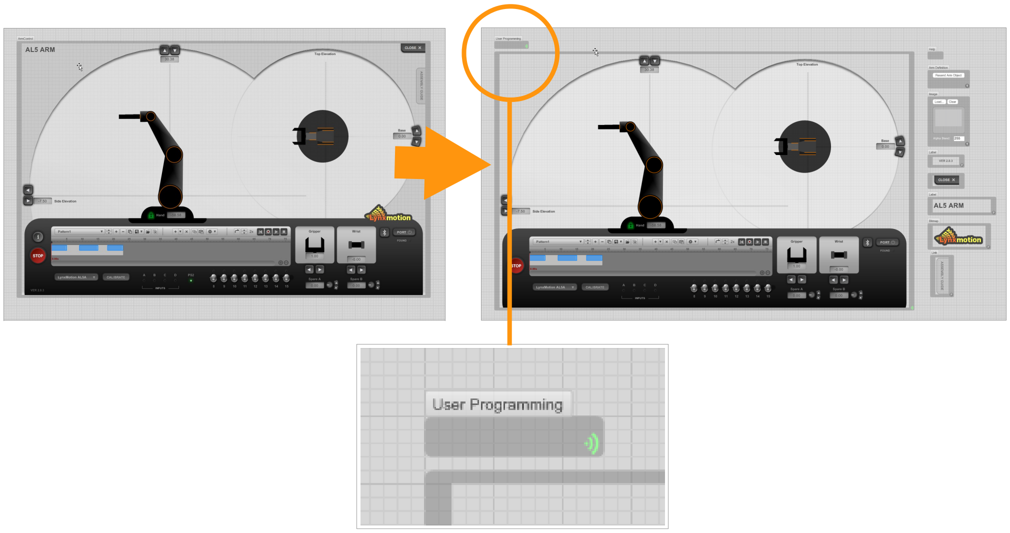
About

Once you're in the editor you could edit the project and change it to do exactly what you want. This is one of the great benefits of the projects being built with FlowStone.

However, if you're just starting that's quite a big step to take so to help you find your feet we have created a simplified user programming area. It's worth reading the short FlowStone in a Nutshell section of the FlowBotics Studio guide before you continue so you're familiar with some of the terminology.

Locating the Programming Module

To navigate to the user programming area double click on the main module of the project. This will take you one level down. You should now see a small module lat the top labelled ‘User Programming’. Double-click on the User Programming module to go inside



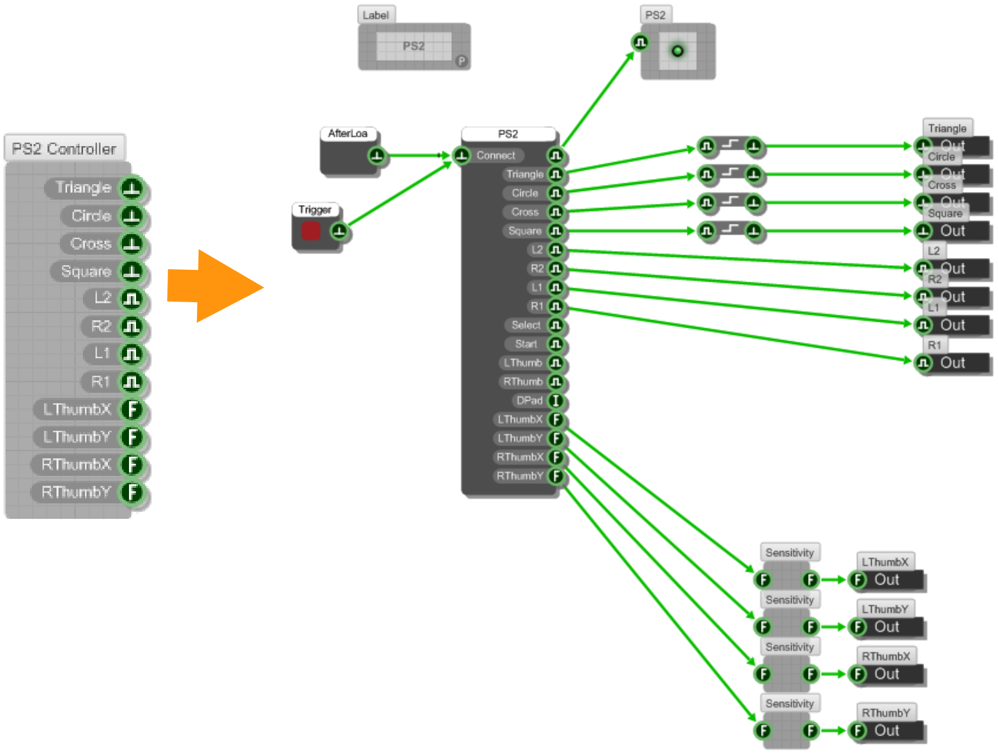
Here you can add an array of supported hardware from digital interfaces, games controllers, sensors etc. You will see that there are already some examples in here of how to control this project. You can use these as a basis to create your own program to control your robot.

There is a PS2 Games controller module, some Scripting Modules, Parameter Modules and Notify Modules.

PS2 Module

The PS2 module connects to the LynxMotion PS2 USB games controller and provides you with various inputs coming from the PS2 controller.

Go inside the module and you'll see how it is defined. If you have a controller connected before opening the project then the module is set up to connect to the controller automatically on opening. If not then plug in your controller and click the red Trigger Button.

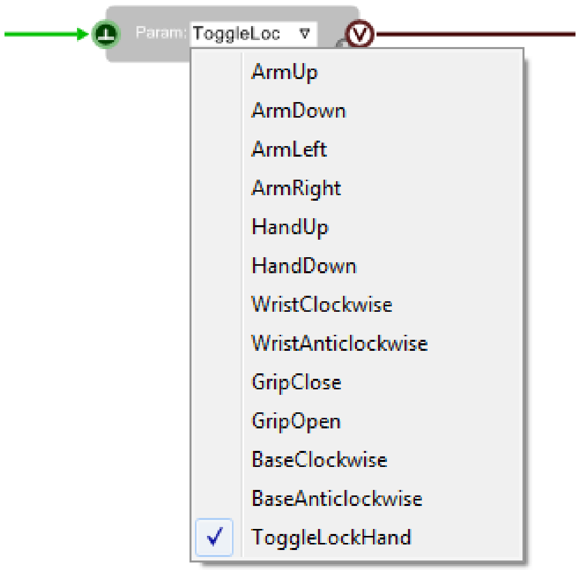


FlowStone has built in support for other controllers including Xbox and Wii. You can find these in the Toolbox to the left of the editor window. See the FlowStone manuals for more information on these.

Parameter Modules

These are modules we have created to give you easy access to the underlying parameters for each robot project. They have a drop down menu to select the parameter to be changed. They can either be triggered by an input or set a specific value depending on the function.

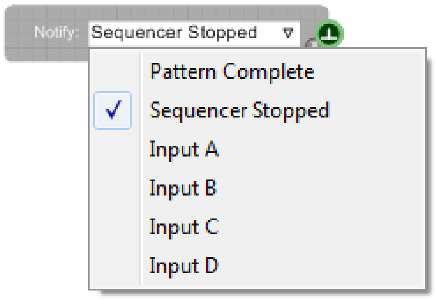
The module below is from the AL5 Arm project. You can see that this allows you to change the various arm parameters by nudging them up or down.



Notify Modules

These are also modules that we created just for the FlowBotics Studio pre-built projects. They feed back information from the sequencer state or the state of the four SSC-32 inputs. There is a drop down list to select the parameter you want to respond to.

The module below shows one of these modules and the 6 options we've set up for you to choose from.



Scripting Modules

These modules contain simplified Ruby scripts (Ruby is a text based programming language inside FlowStone – see the FlowStone user manual for more details). These scripts can send out messages to the sequencer to tell it what to do next. For example you can control the Playback state, jump to a pattern or change the speed or loop mode.

Sequencer Command List

play	stop	reset
loopfalse	looptrue	

goto{pattern name} - e.g gotoWalk or gotoCrouch where Walk and Crouch are pattern names

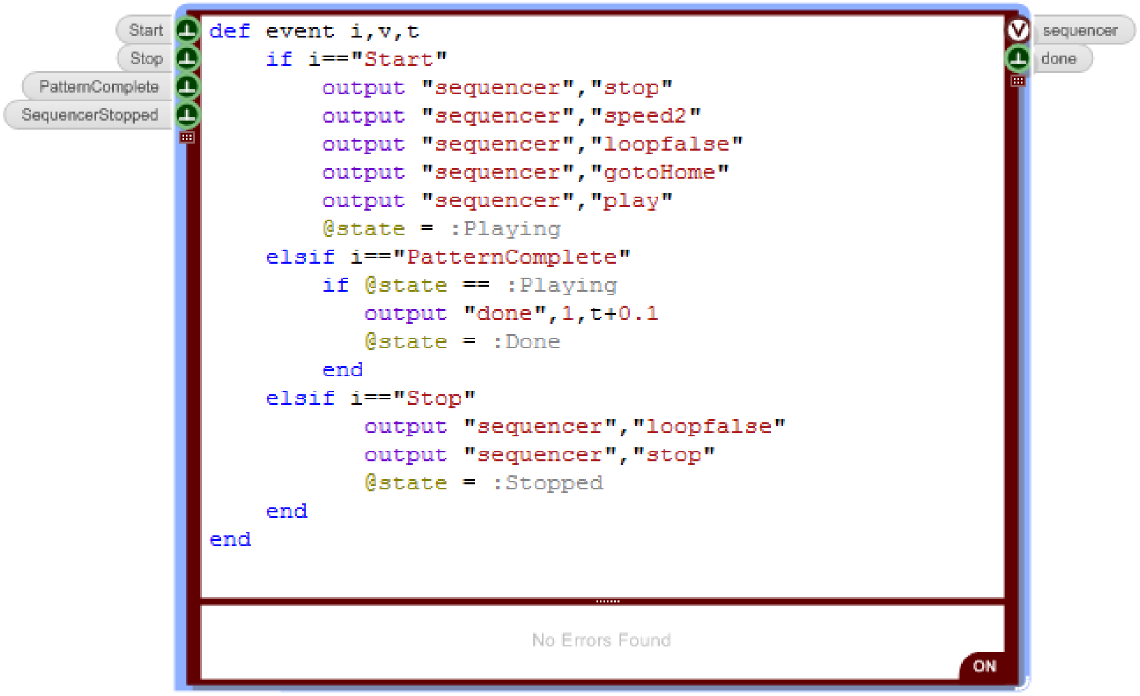
speed{0.25,0.5,1,2,or 4} - e.g. speed0.25 or speed4

speedUp

speedDown

Example Script

Here's a simple example script for you to see how this all works.



```
def event i,v,t
    if i=="Start"
        output "sequencer","stop"
        output "sequencer","speed2"
        output "sequencer","loopfalse"
        output "sequencer","gotoHome"
        output "sequencer","play"
        @state = :Playing
    elsif i=="PatternComplete"
        if @state == :Playing
           output "done",1,t+0.1
           @state = :Done
        end
    elsif i=="Stop"
           output "sequencer","loopfalse"
           output "sequencer","stop"
           @state = :Stopped
    end
end
```

The red component is a Ruby component. On the left edge of the component the green circles are the inputs. These could come from a games controller like the PS2 or an input form a sensor or IO board or just a simple software button.

These inputs are labelled: Start, Stop, PatternComplete & SequencerStopped. The names are then used to reference which input has been triggered.

For example, the statement **if i == “Start”** will run the code in that section once the Start input has been triggered.

You then then see some **output** statements. These send the required sequencer commands to the 'sequencer' output on the right-hand side of the module.

For example, the statement **output ‘sequencer’,”stop”** stops the sequencer.

The subsequent statements set the playback speed to 2 (“speed2”), turn off the looping (“loopfalse”), selects the pattern ‘Home’ (“gotoHome”) and starts playing that pattern (“play”).

You will also see an **@state** variable, this is used to tell the script what state it is currently in. For example, **@state = :Playing**

Following this there is the 'PatternComplete' statement which responds to triggers at the PatternComplete input. This is connected to a Notify Module (PatternComplete) that tells the code when the current playing pattern has finished playing. This is useful to decide when to play the next pattern.

In this case once the 'Home' pattern has finished playing it checks the state is correct (**@state == :Playing**), and then outputs a trigger to the 'done' output (**output "done",1,t+0.1**). The output connector is labelled 'done', the '1' sends a trigger out and the 't+0.1' waits a 10th of a second to ensure all of the other commands have finished.

This trigger can then be used to start another event, play another pattern, output to a hardware output device or whatever you decide.

Finally there is another input called 'Stop' which sends a stop command to the sequencer when triggered.

From this you can see that controlling your robot with real programming is not too complicated thanks to the Ruby scripting inside the FlowStone editor.

This really is just the tip of the iceberg though. For more information please refer to the FlowStone User Guide and Component Reference.