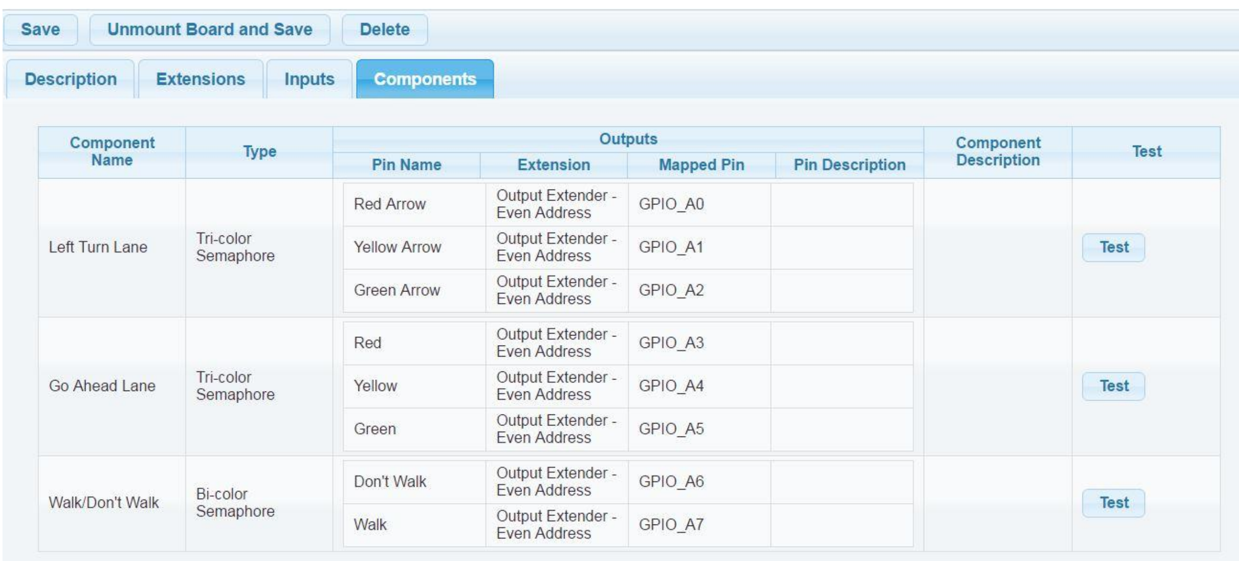
Introducing Breezy4Pi: an easier way to control your Raspberry Pi

The introduction of the Raspberry Pi, an inexpensive, powerful single board computer, has contributed to the explosion of today's makerspace. Add a little hardware, some Java, Python, or C++ code and suddenly, you have a robocar, a low cost media center, a Wi-Fi connected weather station or, in my case, controlling a model railroad.

In 2015, I posed a question, "can I create a web application, written in Java, for the Raspberry Pi that would make it easier for people to program controller sequences for their hardware without knowing a programming language; with mouse clicks and drop-down menus?" The answer was a resounding "yes" and thus, Breezy4Pi was born.

Breezy4Pi enables a user to dynamically model the inputs and outputs of hardware mounted on the Raspberry Pi, map the inputs to trigger the execution of macros which, in turn, manipulate components such as semaphores and motors that have been mapped to the hardware's outputs.   The mapping is done using mostly drop-down menus while the creation of macros and event triggers are akin to filling in the cells of a spreadsheet, once again using mostly drop-down menus. Although, Breezy4Pi is young, I'm finding it to be my preferred way of programming the train yard control panels and traffic light simulators on my model railroad.



Component Name	Type	Outputs				Component Description	Test
		Pin Name	Extension	Mapped Pin	Pin Description		
Left Turn Lane	Tri-color Semaphore	Red Arrow	Output Extender - Even Address	GPIO_A0			Test
		Yellow Arrow	Output Extender - Even Address	GPIO_A1			
		Green Arrow	Output Extender - Even Address	GPIO_A2			
Go Ahead Lane	Tri-color Semaphore	Red	Output Extender - Even Address	GPIO_A3			Test
		Yellow	Output Extender - Even Address	GPIO_A4			
		Green	Output Extender - Even Address	GPIO_A5			
Walk/Don't Walk	Bi-color Semaphore	Don't Walk	Output Extender - Even Address	GPIO_A6			Test
		Walk	Output Extender - Even Address	GPIO_A7			

Hardware attached to the Raspberry Pi is treated as boards, both mounted and unmounted. A mounted board means that it is configured and operational. In this example, we are modeling part of a traffic intersection with various pins of an I/O extender such as a MCP23S17 being mapped to semaphore component LEDs. Clicking on the 'Test' button runs a small test program that helps the user verify the component's physical connection.

Create New Event

When		Changes To	Run	Enabled
Mounted Board	Input			
PiRyte Multi S Rev 1 - inputs only	Track 1 Button	Low	Track One Normal	☑
PiRyte Multi S Rev 1 - inputs only	Track 2 Button	Low	Track Two Normal	☑
PiRyte Multi S Rev 1 - inputs only	Track 3 Button	Low	Track Three Normal	☑
PiRyte Multi S Rev 1 - inputs only	Mainline Button	Low	Mainline Normal	☑
PiRyte Multi S Rev 1 - inputs only	Track 4 Button	Low	Track Four Normal	☑

Events are triggered in response to changes in input conditions. Here, these five inputs will each start the execution of an individual macro when it transitions from high to low.

Save Delete Stop

Description Macro Definition

Line	Tag	Mounted Board	Component	Function	Parameters
1	Start	PiRyte Multi S Rev 1 Traffic Semaphores	Go Ahead Lane	Turn On	**Pin Name:** Green
2		PiRyte Multi S Rev 1 Traffic Semaphores	Walk/Don't Walk	Turn On	**Pin Name:** Don't Walk
3		PiRyte Multi S Rev 1 Traffic Semaphores	Left Turn Lane	Pulse	**Pin Name:** Green Arrow **Duration (milleseconds):** 6001 **Wait Until Done:** TRUE
4		PiRyte Multi S Rev 1 Traffic Semaphores	Left Turn Lane	Pulse	**Pin Name:** Yellow Arrow **Duration (milleseconds):** 4001 **Wait Until Done:** TRUE
5		PiRyte Multi S Rev 1 Traffic Semaphores	Left Turn Lane	Blink Timed	**Pin Name:** Yellow Arrow **On Time (milleseconds):** 500 **Duration (milleseconds):** 3950 **Wait Until Done:** TRUE
6		PiRyte Multi S Rev 1 Traffic Semaphores	Left Turn Lane	Turn On	**Pin Name:** Red Arrow

Here is a partial example of a macro. Each component on each mounted board has a set of functions that the macro can execute such as, 'Turn On', Pulse, etc… Everything is symbolically referenced so that a user doesn't need to know how a component is mapped to an I/O extender pin but merely that one wants to blink the "Walk" light, for example.

In conclusion, combining the maturity and power of Java with the power of the Raspberry Pi has yielded a powerful sequence controller that will only become more powerful over time. Add in a REST interface and one could query from a remote machine how hardware is attached to the Raspberry Pi, how it is configured and even execute macros remotely. Best of all, Breezy4Pi is free for personal use. Check it out at www.Breezy4Pi.com and https://github.com/tomtibbetts/Breezy.