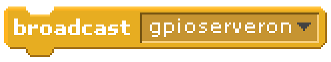
Rapid prototyping guide for Explorer Hat and Scratch

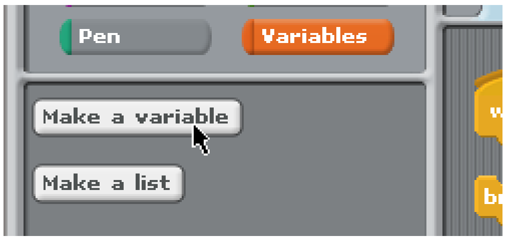
Setup

This initializes the GPIO server application that listens to broadcasts and updates variables in Scratch.

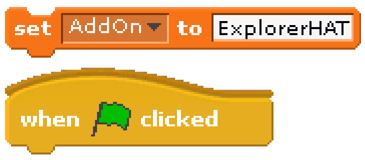
Broadcast message: **gpioserveron**



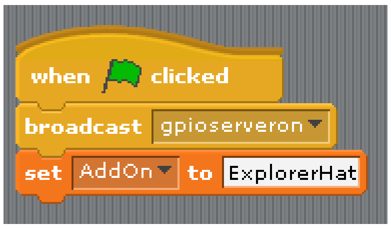
To set up the use of special ExplorerHAT options, create a variable called **AddOn**.



Create a block to assign it to the value **ExplorerHAT**.



Attach the **broadcast gpioserveron** block and the **set AddOn to ExplorerHAT** blocks to the when Flag is clicked event block. This makes sure that the script runs the initialization at the very beginning.

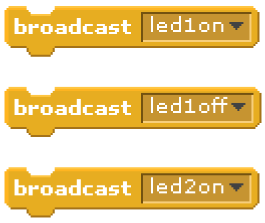


The name **ExplorerHAT** is case-sensitive. In general, with programming and code, it is best practice to pay close attention to both the spelling and capitalizations.

Input & Output Control

Basic input / output control using Scratch is accomplished using the broadcast message block. The easiest way is to create a new broadcast message for each command.

LED control

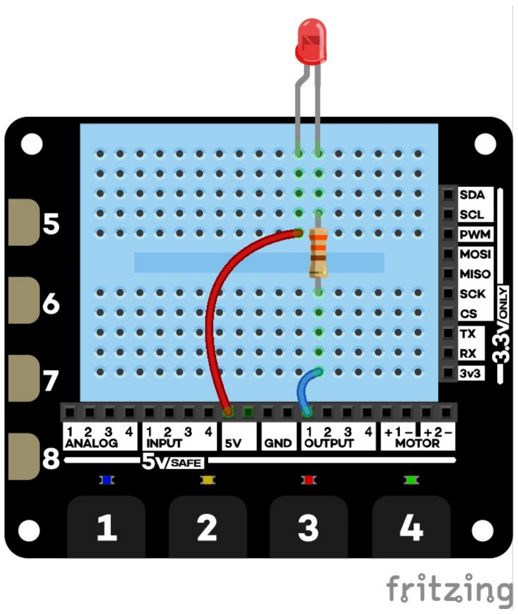

broadcast led1on
broadcast led1off
broadcast led2on

There are four coloured LEDs under the touch pads. These can be controlled with a simple broadcast message.

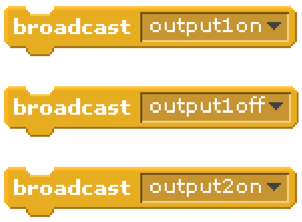
Broadcast message: led#[on/off]

OUTPUT control - (OUTPUT1 - OUTPUT4)

There are four controllable outputs on the ExplorerHAT. These are active *LOW OUTPUTs* (Sinks to GND).



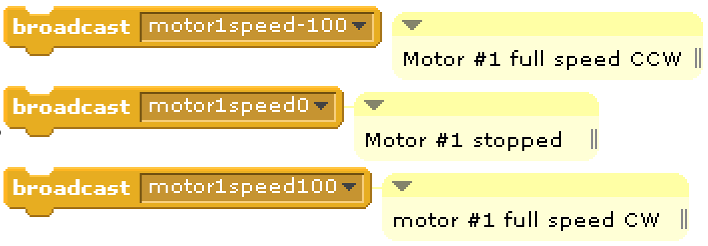
To turn on an LED, wire LED to 5V and connect negative (the shortest) leg to output pin (through a current limiting resistor)


broadcast output1on
broadcast output1off
broadcast output2on

Broadcast message: output#[on/off]

Motor control - Motor1 and Motor2

The speed of the motors can be controlled to spin CW and CCW by sending it a value between -100 and +100. Connect the two wires (red/black) from each motor to the + and - pins. You can also use these commands to control the solenoid.


broadcast motor1speed-100
Motor #1 full speed CCW
broadcast motor1speed0
Motor #1 stopped
broadcast motor1speed100
motor #1 full speed CW

The structure of the command is: Broadcast message: motor#speed[value]

Using variables to simplify the broadcast messages

You can also uses variables and the **join** block to create your own broadcast messages in the code. Here's an example that blinks a random LED on and off.

Here are all of the blocks that you're going to need for this.

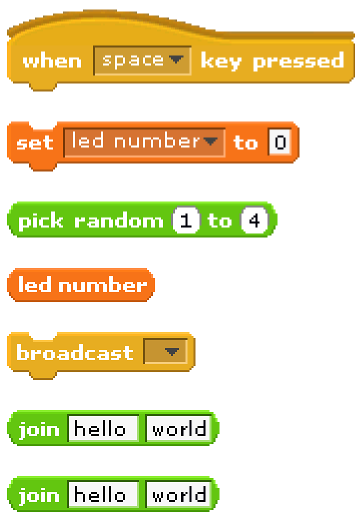
Create a variable called led number.

You're going to use the pick random block to randomly pick which LED to turn on.

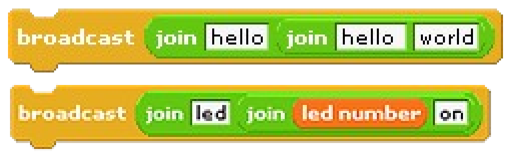
You need the **led number** variable block, the broadcast block, and two join blocks.

You're going to use the two join blocks to join three elements together into a compound broadcast message. Drag one **join** into the second space (world) of the first **join**.

Now, change the text so that it joins the text **led** with the variable **led number** and the text **on**.



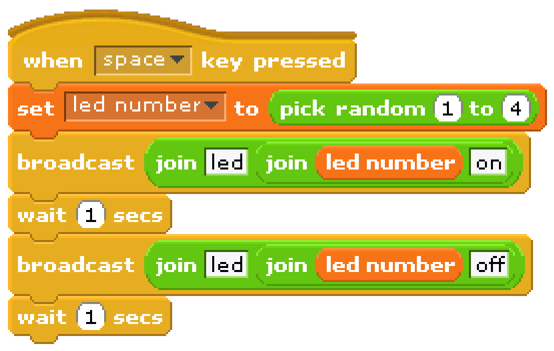
```
when [space] key pressed
set [led number] to [0]
pick random (1) to (4)
(led number)
broadcast [ ]
join [hello] [world]
join [hello] [world]
```



```
broadcast (join [hello] (join [hello] [world]))
broadcast (join [led] (join (led number) [on]))
```

Putting it all together. Here's the complete script that picks a random number from 1 to 4, and blinks that LED - turning it on for one second and then off for one second.

You can use the join block and **variables** to control any number of other features in scratch.



```
when [space] key pressed
set [led number] to (pick random (1) to (4))
broadcast (join [led] (join (led number) [on]))
wait (1) secs
broadcast (join [led] (join (led number) [off]))
wait (1) secs
```

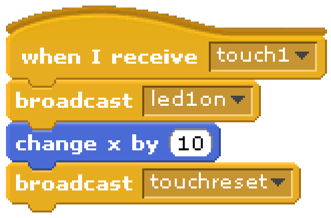
Touch Input

There are four points that use capacitive touch to detect a touch input. These are labelled 1, 2, 3, & 4. There are four additional clip points that can be used with alligator clips to connect to a variety of conductive objects to use as digital sensors.

A 'touch[#]' broadcast message is received when one of the tabs is touched. You can use the "**When I receive [ ]**" event block to define what happens when a touch is received.

You must clear the touch event by broadcasting a message: "**touchreset**" If you don't it won't be able to detect another touch event.

Example:
Moves sprite 10 spaces in x and turns on LED1



```
when I receive [touch1]
broadcast [led1on]
change x by (10)
broadcast [touchreset]
```

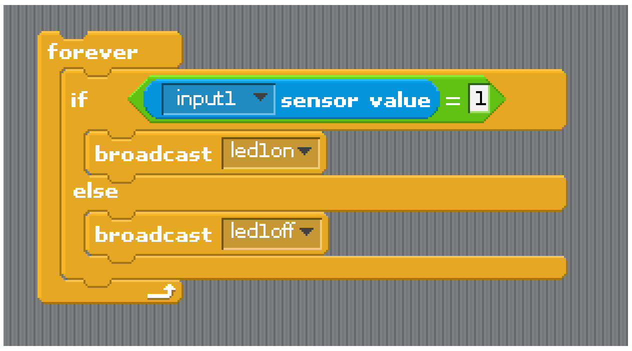
Connecting Other Inputs

There are four points that use capacitive touch to detect a touch input. These are labelled 1, 2, 3, & 4. There are four additional clip points that can be used with alligator clips to connect to a variety of conductive objects to use as digital sensors.

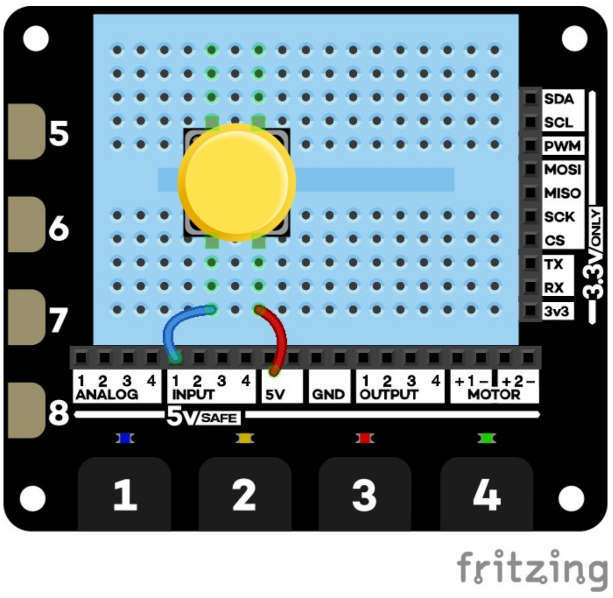
Digital Inputs

A digital input is something like a switch. It is either on (1) or off (0). The on state indicates that it's connected to 5V.

Here is a simple example using a button switch. Connect up the circuit shown. When you push the button, you connect the input pin to 5V.



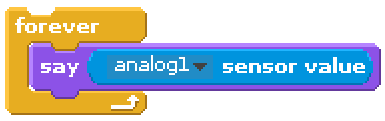
```
forever
  if <([input1] sensor value) = 1>
    broadcast [led1on]
  else
    broadcast [led1off]
```

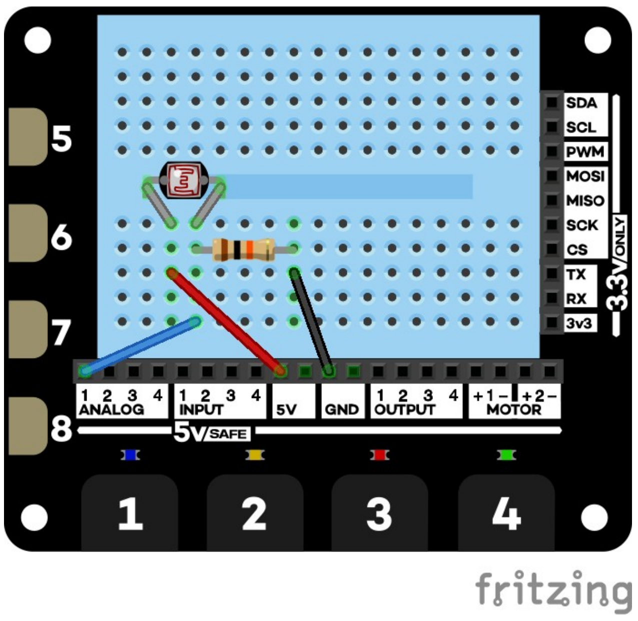


Analog inputs

An analog input can measure not only on and off, but variations in between. It converts a voltage into a value that you can use in the code.

Here is a wiring example of connecting up a light dependant resistor (LDR).





The circuit diagram for this looks like this. Vout of the sensor is connected to the **analog1** input pin. When connected like this, the LDR works as a variable voltage divider. As the light changes, the voltage between the two resistors changes between 0V and 5V.

Under the sensor block, you'll find an option for **analog1** and **adc1**. How do the values compare?

This is taken and modified from: https://sparkfuneducation.com/how-to/explorerhat-and-scratch-on-raspberry-pi.html