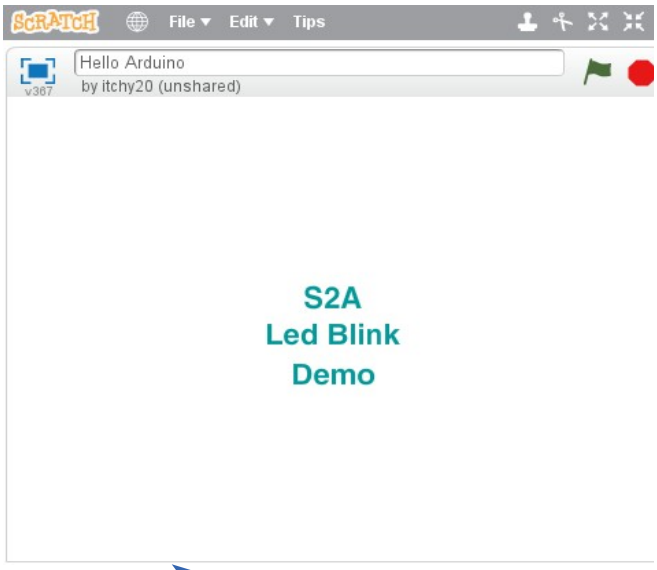
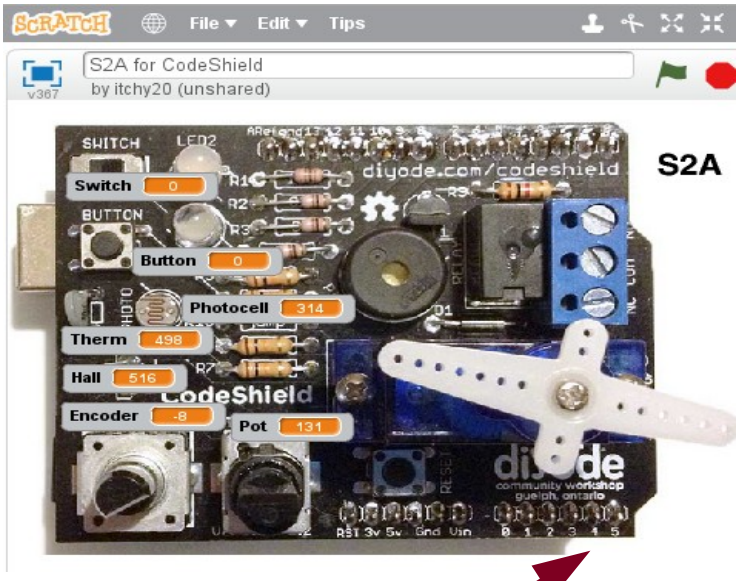
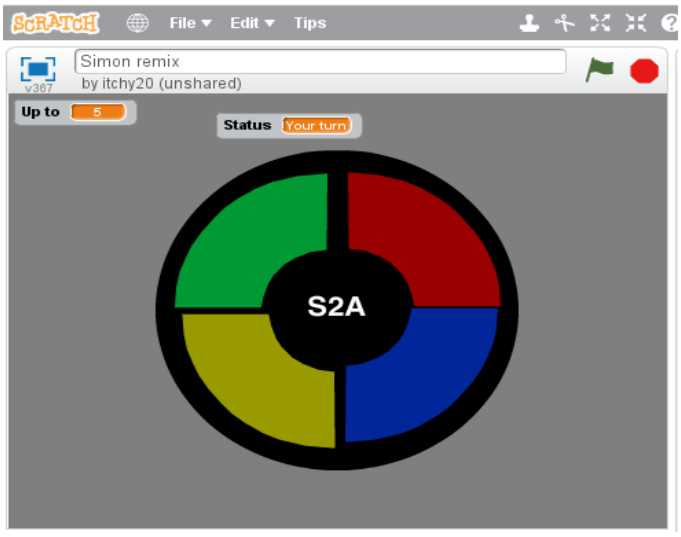
S2A

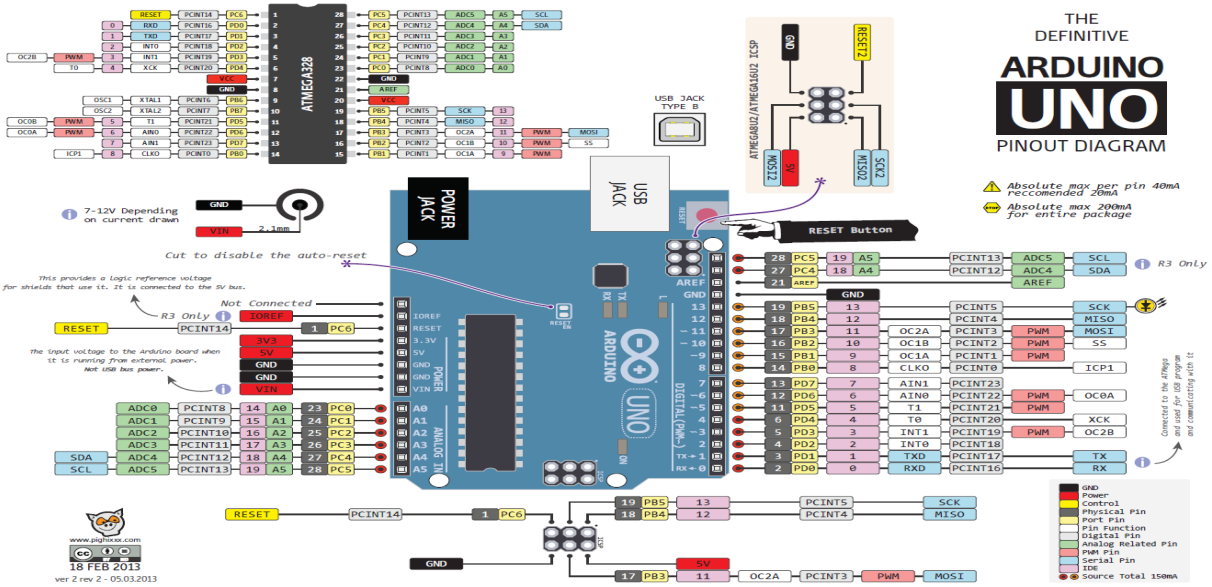
The Scratch 2.0 Hardware Extension For Arduino



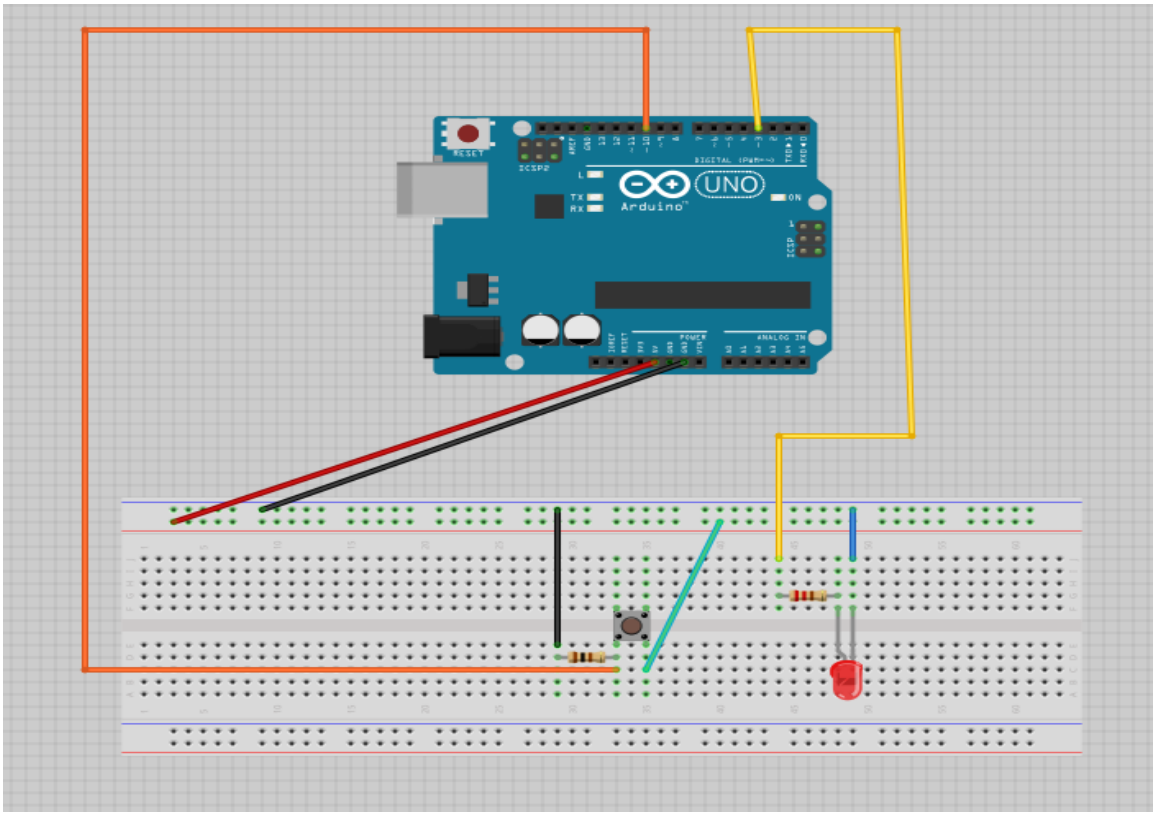




Your *Own* Scratch Creation



Arduino Pinout Diagram



Single LED and Switch Wiring Example for the Simon Demo

Table of Contents

1 What is S2A?....1
2 What is Included?....1
3 What Do You Need to Use this Package?....1
4 S2A Software Architecture....1
  4.1 s2a....2
  4.2 arduino_serial....2
  4.3 arduino_translator....3
  4.4 scratch_translator....3
5 An Example: Building an S2A Application for the Simon Memory Game....3
  5.1 Application Design....3
  5.2 Build and Test the Arduino Circuit....4
  5.3 Specify and Create the Scratch Processing Blocks....4
  5.4 Create the s2a Map Configuration File....7
  5.5 Starting s2a....10
  5.6 Test End to End Communications....11
  5.7 Create The Scratch Program....12
6 The Demos....12
  6.1 How to Use The Demos....12
  6.2 CodeShield....13
  6.3 Pin 13 LED Blinker....13
  6.4 Simon Memory Game....13
7 Included Files and Directory Structure....13
8 Acknowledgements....14
9 Contact Information....14

1 What is S2A?

S2A is a PC based python application that allows you to connect a Scratch 2.0 program to an Arduino micro-controller, all without writing a single line of additional code. S2A performs all translations by using a simple, easily maintainable and flexible user application specific text file.

Currently, S2A supports the following Arduino functionality through configuration file mapping:

- Pin mode setup (input/output)
- Digital Read
- Digital Write
- Analog Read
- Analog Write
- Reading a rotary encoder (pins A0 and pins A1)
- Servo library (pin 5)
- Tone library (freq, and duration – pin 3)

The pins for the rotary encoder, servo and tone libraries can be modified by recompiling the JSON Client sketch provided as part of this package. A future release will provide commands to specify the pins through the configuration file.

2 What is Included?

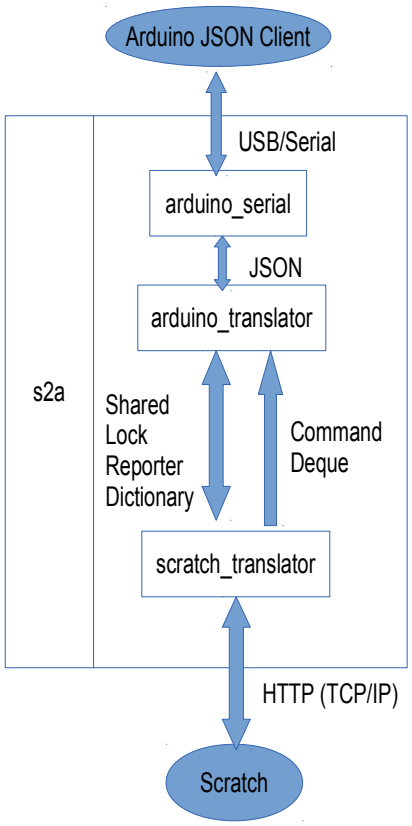
- S2A Python scripts for both Python2 and Python3.
- The Arduino JSON Client sketch.
- All necessary configuration and Scratch application files for CodeShield support, an Arduino Pin 13 LED blinking light tester, and a Simon Memory game.

3 What Do You Need to Use this Package?

- Python version 2.7+ or 3.3+ installed on your computer (see python.org to download)
- PySerial installed (see http://pyserial.sourceforge.net/ for installation instructions)
- An Arduino Uno or Leonardo micro-controller.
- And of course, access to Scratch 2.0.
- This package has been tested on both Ubuntu Linux and Windows 8.

4 S2A Software Architecture

The architecture is illustrated in the following drawing:



s2a Software Architecture

4.1 s2a

This class is the "main" class that instantiates all other classes needed for operation. It instantiates the shared resources required for safe inter-thread communications. It supplies all references needed by the other classes at the time of their instantiation..

4.2 arduino_serial

This class provides serial link communications between s2a and the Arduino micro-controller. It is in a separate class so that support for other types of communications (WiFi, Ethernet, etc.) can be easily supported in the future.

4.3 arduino_translator

This class handles all data translation to and from the Arduino micro-controller. After instantiation, this class spawns a separate thread that will continuously poll the Arduino for status information. Status data is placed in a reporter dictionary that is shared with the scratch_translator class and protected by a thread lock. Continuously polling ensures a fast and efficient transfer of sensor data to Scratch upon request. During the continuous internal polling cycle, each sensor is polled on an individual basis to provide orderly control flow control. After each sensor poll, the command deque is checked for any pending actuator commands issued by Scratch. Upon detection they are sent immediately to the Arduino. The Scratch command deque is managed as a FIFO and it does not require lock protection.

4.4 scratch_translator

The scratch_translator is a simplified HTTP web server. It provides the connection services for Scratch connections over TCP/IP. For normal operation, Scratch utilizes the HTTP GET mechanism and the Scratch translator processes all GET requests. When a poll request is detected, the scratch_translator acquires the lock to the pre-populated reporter dictionary. It reads all sensor data and builds a Scratch HTTP reply containing a full set of sensor data. When an actuator command is detected, it is tested for validity and then placed in the command deque for processing by the arduino_translator.

5 An Example: Building an S2A Application for the Simon Memory Game

5.1 Application Design

A fully working version of this example is included with this package.

In building an application, the first step is to design your application. Decide what actuators and sensors are required and what Scratch blocks you will need to represent them.

For the Simon game, an existing game was remixed from an implementation originally written by the Scratch user jimfred. For the game, we are going to use four actuators and four sensors. The actuators are a red, yellow, green and blue LED, one for each color of the game. The sensors are four push button switches, one associated with each LED.

The inside cover of this document has a breadboard circuit showing the wiring diagram for one LED and one switch. Just replicate for the other 3.

The design specifies pinouts as follows:

- Red LED on pin 3
- Yellow LED on pin 4
- Green LED on pin 5
- Blue LED on pin 6

Place the switches physically near the LEDs you wish to associate with the switch. Here is switch the pinout:

- Red Switch on pin 10

- Yellow Switch on pin 9
- Green Switch on pin 8
- Blue Switch on pin 11

Notice that you can define pins in any order and use any mode that your design requires. There are no restrictions. The pins picked here are just an example of a working implementation.

5.2 Build and Test the Arduino Circuit

If you are breadboarding the Arduino circuit, you can use the Arduino JSON client to easily exercise your design. To test, you would send JSON commands directly to the Arduino through the Arduino IDE serial monitor. There is no need to create an Arduino sketch to test your design. First, download the S2AJson.ino client sketch into the Arduino. There are template commands listed near the beginning of this sketch. Cut, paste and modify these commands and execute them through the serial monitor. Since the commands are provided in valid JSON format, you do not need to learn JSON to use this feature. Once you are satisfied that your Arduino hardware design is functioning properly, go on to the next step of specifying the Scratch blocks to support your design.

NOTE: S2AJson.ino has a limitation in that it is not able to interpret analog pin numbers using the "A" pin prefix (A0, A1, etc). All pins, both analog and digital are specified using their actual pin number. To find the analog pin numbers, the "Definitive Arduino Pinout Diagram" is provided in the inside cover of this document. Using the diagram, it can be seen that A0 translates to pin 14, A1, to pin 15, etc. The pinout of analog pins for the Uno and Leonardo are the same, so this diagram can be used for either board. A future version of the client may rectify this deficiency.

5.3 Specify and Create the Scratch Processing Blocks

To create the monitor and actuator blocks that will be used in the Scratch program, a block descriptor file needs to be created. The Scratch code block descriptor files for all of the examples, are contained in the demo folders.

The .s2e code block descriptor file uses a JSON format as specified by the Scratch development team. After creating a .s2e descriptor file, it is a good idea to check if the file you created is in valid JSON format. To do so, use the on-line JSON validator, jsonlint.com. If you import an improperly formatted file to Scratch, Scratch will reject the file without comment, and no blocks will be created. Note also that JSON has no provision for comments (not my decision, nor that of the Scratch team).

Let's dissect s2a_simon.s2e line by line.

```
{      "extensionName": "S2A Simon Game",
       "extensionPort": 50211,
       "blockSpecs": [
           [" ", "Red_LED %n",       "red_led_pin_3",    0 ],
           [" ", "Yellow_LED %n",    "yellow_led_pin_4", 0 ],
           [" ", "Green_LED %n",     "green_led_pin_5",  0 ],
           [" ", "Blue_LED %n",      "blue_led_pin_6",   0 ],
           ["r", "Red_Switch",       "red_switch_pin_10"   ],
           ["r", "Yellow_Switch",    "yellow_switch_pin_9" ],
           ["r", "Green_Switch",     "green_switch_pin_8"  ],
           ["r", "Blue_Switch",      "blue_switch_pin_11"  ]
       ],
    "useHTTP": true
}
```

The file is bounded by an open and close brace.

The ***extensionName*** can be anything you wish and will appear in the More Blocks section of Scratch when the blocks are created.

The ***extensionPort*** is the TCP/IP port number of the S2A web server. It is used by Scratch to connect to the server. It must match exactly with the one used in the translation map configuration file (see section 5.4).

BlockSpecs is a JSON array of JSON arrays. One array needs to be specified for each Scratch block to be created.

There are 2 “types” of arrays. One will create a reporter block (sensor block). The arrays that begin with `["r",...` are reporter blocks.

The arrays that begin with `" ",...` create command blocks (actuator blocks).

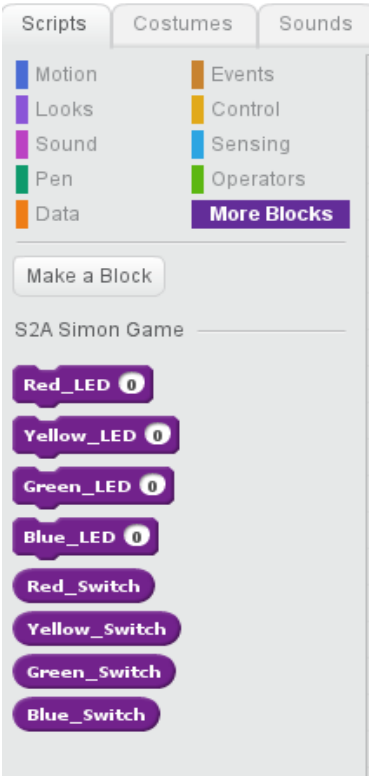
Each block is described by an array with the following fields:

* block type
* block format
* operation or remote variable name
* (optional) zero or more default parameter values

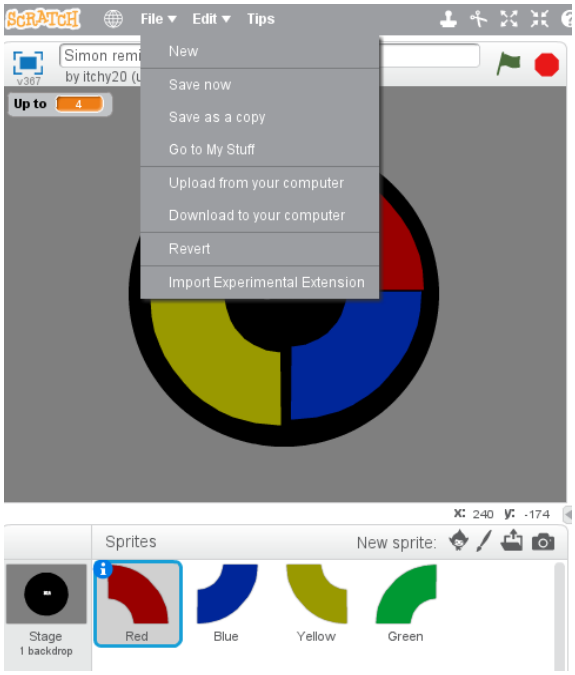
The block type is one of these three strings:
" " - command block
"r" - number reporter block (round ends)
"b" - boolean reporter block (pointy ends)

The block format is a string that describes the labels and parameter slots that appear on the block. Parameter slots are indicated by a word starting with "%" and can be one of:
%n - number parameter (round ends)
%s - string parameter (square ends)
%b - boolean parameter (pointy ends)

Here is a screenshot of the blocks created by the file above:



To load the .s2e file into Scratch, open the Scratch Project Editor and then hold the shift key while clicking on the File menu option. Select “Import Experimental Extension”. A file chooser dialog box will open. Select the desired file. Once all the other steps below are complete, we can then proceed to build a Scratch program that interacts with the Arduino.



5.4 Create the s2a Map Configuration File

The following is the Simon game configuration file used by S2A to perform command and reporter status mapping between Scratch and the Arduino. A discussion of each section is provided below in *italic* font. Note, that unlike the Scratch block descriptor file, comments may be freely added to this file. Comments begin with #.

```
[ProjectNameSection]

Project = S2A_Simon_Game
```

This section contains a user selected project name. It is not used by S2A, but helps to identify the file when editing.

```
[SerialPortSection]

# arduino port

ComPort = /dev/ttyACM0

# baudrate must match the value of the arduino sketch

BaudRate = 115200

TimeOut = 1
```

This section contains the data for the serial port. The only value that should be modified is the ComPort your Arduino uses. Different operating systems likely use different port designators. Windows may use something like COM3.

```
# this value must match that in the .s2e script

[HTTPServerSection]

PORT = 50211
```

This value must match the value in the .s2e file exactly so that Scratch can find the HTTP server.

```
# specify the initial pin direction for each pin used by the application

# format is PIN = DIRECTION

[ArduinoPinDirection]

# Red LED

3 = output

#Yellow LED

4 = output
```

```
# Green LED
5 = output
# Blue LED
6 = output
# Red Switch
10 = input
# Yellow Switch
9 = input
# Green Switch
8 = input
# Blue Switch
11 = input
```

This section is user application specific. It identifies each pin and its mode.

```
# for each arduino pin used as an output, specify
# the output type and its value
# format is PIN = TYPE, VALUE
[ArduinoInitialOutputPinValues]
# Red LED
3 = digital,0
#Yellow LED
4 = digital,0
# Green LED
5 = digital,0
# Blue LED
6 = digital,0
```

This section is user application specific. For each output pin, the type of output (digital or analog) and the initial output value is specified here. For this example, all LEDs are set to an off state.

```
# These values must be the same used in the Scratch .s2e file
# format is PIN = Scratch_ID
[ReporterMapSection]
10 = red_switch_pin_10
9 = yellow_switch_pin_9
8 = green_switch_pin_8
11 = blue_switch_pin_11
```

This section is user application specific. For each reporter pin, the pin number is specified and it must be set to the exact name used in the .s2e file so that proper mapping is performed.

```
# report pin type specifier
# format is PIN = TYPE
[ReporterPinToTypeMap]
# Red Switch
10 = digital
# Yellow Switch
9 = digital
# Green Switch
8 = digital
# Blue Switch
11 = digital
```

This section is user application specific. It maps the reporter pins to their type (digital or analog).

```
# This section contains Scratch commands as the key.
# These values must be the same as the ones specified in the Scratch .s2e file.
# Format is COMMAND = PIN, #_OF_PARAMETERS FOR THE COMMAND, TYPE
# A type value of None indicates that this is a "special" command and is not
# a simple pin manipulation
```

```
[CommandPinMapSection]
red_led_pin_3    = 3,1,digital
yellow_led_pin_4 = 4,1,digital
green_led_pin_5  = 5,1,digital
blue_led_pin_6   = 6,1,digital
```

This section is user application specific. It maps the names of command blocks specified in the .s2e file to pin numbers, specifies the number of parameters for each block and specifies its type.

```
# These are the set of JSON commands sent to Arduino
# The values in CAPS will be replaced with the actual values at run time
[JsonStringTemplateSection]
writeValueToPin = {"write":{"pin":PIN,"type":"TYPE","value":VALUE}}
setPinDirection = {"mode":{"pin":PIN,"mode":"MODE"}}
readPinValue = {"read":{"pin":PIN,"type":"TYPE"}}
readEncoder = {"read":{"encoder":100,"type":"analog"}}
writeServo = {"write":{"type":"servo","value":VALUE, "pin":5}}
writePiezo = {"write":{"type":"piezo","freq":FREQ,"time":TIME}}
```

These strings are used internally and should not be changed by the user.

```
# any special processing needed
# for CodeSheild, the Tone and Servo commands do not restore
# interrupts appropriately, so special LED processing is needed

[SpecialProcessing]
enable_special_LED_processing = False
```

We are not using Tone or Servo for this program, so we set this to false.

5.5 Starting s2a

All the pieces needed to run s2a are now in place. We do not need a completed Scratch program to continue, just one that has loaded the .s2e file and has successfully created the Scratch blocks.

To start the program, decide if you are going to run the python2 or python3 version. We will use python2 as an example.

Here are the steps:

1. Open a command console
2. CD to the directory containing s2a.py.
3. Type the following command:

        python s2a.py ../configFiles/s2a_simon.cfg

Make sure that the Arduino with the JSON client sketch loaded is plugged in to your computer and that you have the Scratch project open.

You should see the following appear on your console:

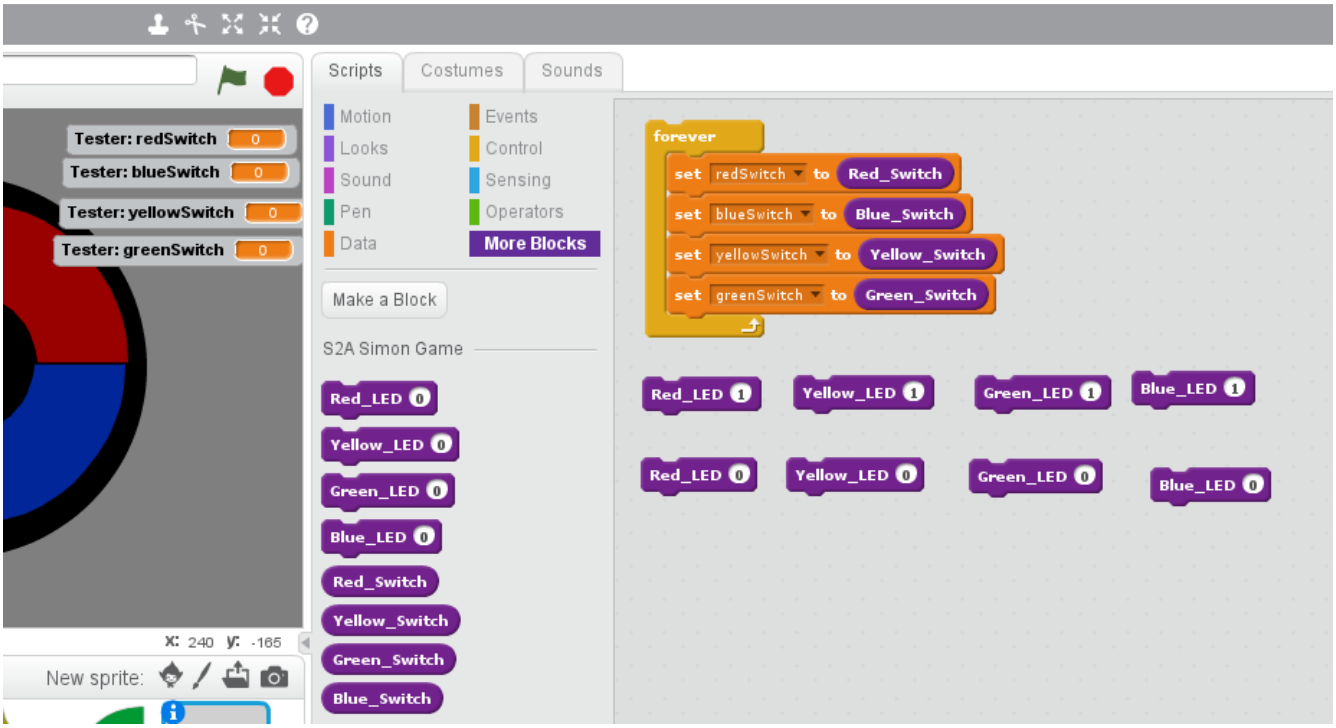
```
Using configuration file: ../configFiles/s2a_simon.cfg
If you wish to  use another configuration file,
specify the configuration file name on the command line.
Example:
python scratch_extension.py my_own_config_file

Opening Arduino Serial port /dev/ttyACM0
Arduino interface is up and running.

HTTP Serverport is initialized with port = 50211

Starting Scratch HTTP Server!
Use <Ctrl-C> to exit the extension

Waiting for Scratch handshake ....
Scratch is initialized and ready to Rock n Roll
```

5.6 Test End to End Communications

To test everything is working properly, create a tester sprite in Scratch. First create a variable for each switch on the Arduino circuit. Then in a forever loop, set the variable to the reporter value. Double click on the forever loop to have it execute, and press each switch. The values on the stage for the variable should change from 0 to 1 for each switch.

To test the LEDs, set an LED block parameter to a 1 and double click the block. To turn it off, set the parameter to a 0 and double click to extinguish it. Do this for all remaining LEDs.

It is assumed that you already checked your Arduino circuit earlier, so if things do not work as expected, carefully check that the proper values in.s2e file and the .cfg file match.



5.7 Create The Scratch Program

Once everything has been verified, create the Scratch program. You have now implemented your first Scratch/Adruino integration! A fully working version of the game has been provided as part of the demo.

6 The Demos

6.1 How to Use The Demos

Each demo folder contains its own Scratch program with the Scratch blocks already created for you.

To run the demo, load the .sb2 image into Scratch (press File at the upper left of the Scratch project editor and select “Upload from your computer” from the drop down menu), then start S2A using the .cfg file located in the demo folder or the configFiles directory. So for example, to run the CodeShield demo:

- Upload “S2A for CodeShield.sb2” into Scratch.
- Open a command console, cd to the S2A_python2 directory and type the following command line:
  - python s2a.py ../configFiles/CodeShield.cfg

6.2 CodeShield

If you have a CodeShield, full support is available by using the supplied files, CodeShield.cfg and CodeShield.s2e. A “tester” Scratch application is provided to view all the sensors and to have the ability to try each actuator. The cover of this document shows the Scratch demo for CodeShield. Sensor variable value are overlayed onto a photo of CodeShield.

6.3 Pin 13 LED Blinker

There is a demo to blink the LED that is provided on pin 13 of the Arduino board. Use the supplied files to run the demo.

6.4 Simon Memory Game

Refer to section to Section 5 for a full discussion of this demo.

7 Included Files and Directory Structure

```
S2A
├── arduinoSketch
│   └── S2AjsonClient
│       └── S2AjsonClient.ino
├── configFiles
│   ├── CodeShield.cfg
│   ├── HelloArduino.cfg
│   └── s2a_simon.cfg
├── S2A_python2
│   ├── arduino_serial.py
│   ├── arduino_serial.pyc
│   ├── arduino_translator.py
│   ├── s2a.py
│   ├── s2e.cfg
│   ├── scratch_translator.py
├── S2A_python3
│   ├── arduino_serial.py
│   ├── arduino_translator.py
│   ├── s2a.py
│   ├── s2e.cfg
│   └── scratch_translator.py
```

```
└── ScratchDemos
    ├── CodeShield
    │   ├── CodeShield.cfg
    │   ├── CodeShield.s2e
    │   └── S2A for CodeShield.sb2
    ├── HelloArduino
    │   ├── HelloArduino.cfg
    │   ├── HelloArduino.s2e
    │   └── Hello Arduino.sb2
    └── SimonRemix
        ├── s2a_simon.cfg
        ├── s2a_simon.s2e
        └── Simon remix.sb2
```

8 Acknowledgements

Many thanks go to Mitch Resnick and John Maloney both of MIT for providing me with an early copy of the Scratch 2.0 Hardware Extensions specification. Without their support this project would not have been possible.

The good folks at diyode.com for designing and marketing the CodeShield, a quality cost effective kit with a fascinating set of sensors and actuators.

The folks at Arduino who brought back the fun of homebrewing (now called hacking) electronic gadgets.

And last, but certainly not least, my wife, who has both encouraged and allowed me to fully embrace my inner geek.

Alan Yorinks

September 2013

9 Contact Information

If you have any comments, corrections, or concerns, please feel free to contact me at:

email: MisterYsLab@gmail.com