Grand Challenge Design OS

Getting started

pdykes@gmail.com (v0.0.2)

Grand Challenge Operating System

- Supports the “Grand Challenge” IoT game used in schools, allowing students to extend and play with IoT, yet avoid the enormity of network programming and managing 10s to 100s of Rpi units
- Support a logical set of “resources” and physical circuits “basis” that can be reused to create the concept of a full IoT environment supporting a gaming or modeling environment
- No database needed, driven from a configuration file and command-line
- Nodejs web os running on Rpi units
- Data can be provided via direct query or via emitters
- Supports a framework to run on N Rpi units to ease the distributed programming
- Extensible by dev team to add any sort of IoT extensions
- Extensions loaded in real-time as needed and offer a rich state of standard framework APIs
- Command line and Restful api support provided to control all instances via many methods

- Github location: https://github.com/pdykes/gcos

- Todo: MPV2
  - Command line tool
  - Enable basis circuits
  - Emit results to target URLs, websocket endpoints

Diagram

Grand Challenge
Command Line and Web

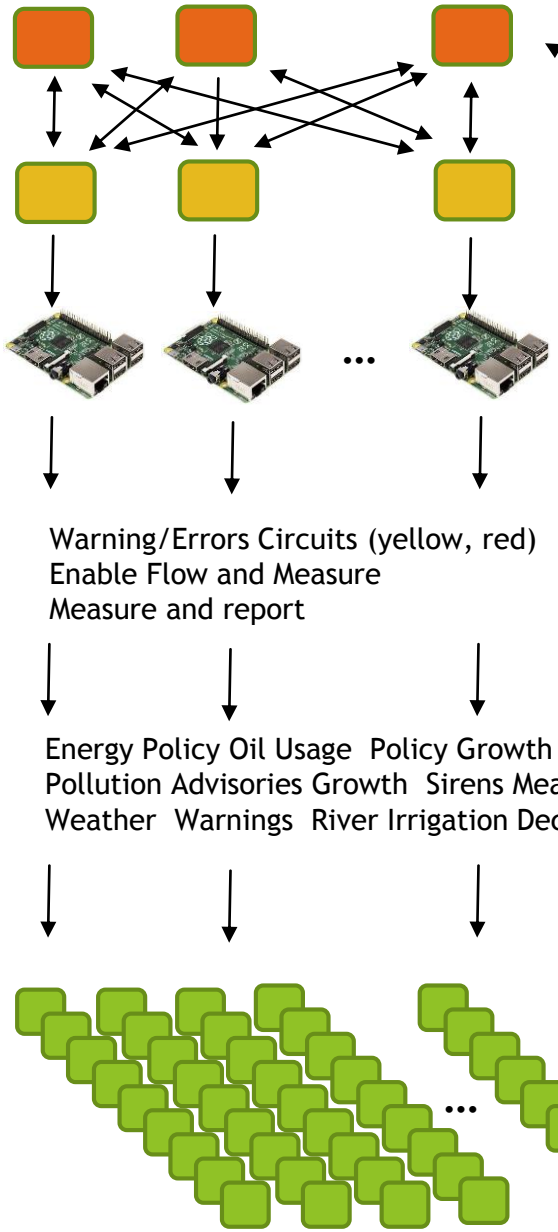
Grand Challenge
Operating System

IoT Tier

Logical Game
IoT Circuits

Logical Game
Resources/Concepts

Game Cells



Players

Apply
Technology

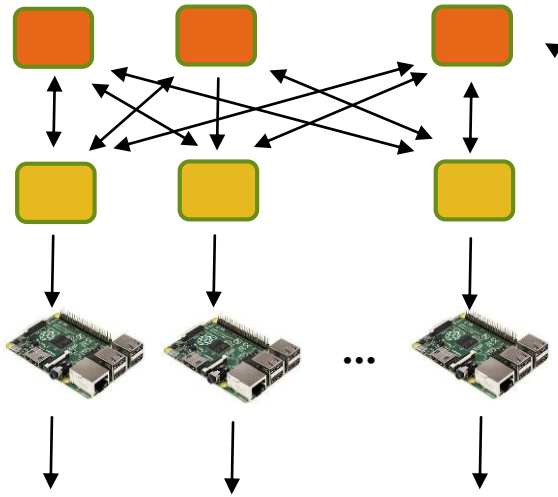
Game Needs

Example

Grand Challenge
Command Line and Web

Grand Challenge
Operating System

IoT Tier



red_led siren_function

Specific Circuit “Basis”

(specific circuit)

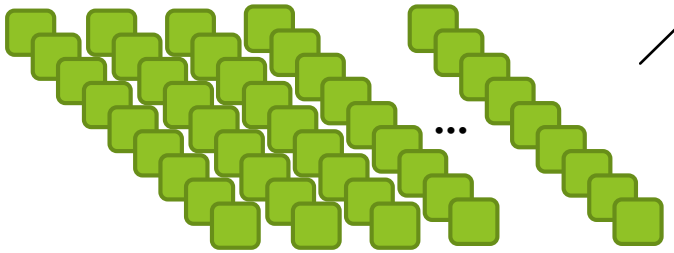
Logical Game “Resource”

(1 circuit -> n logical resources)

tornado_warning
flood_alert

community_meeting
tornado_siren

Game Cells



Apply Technology

Players

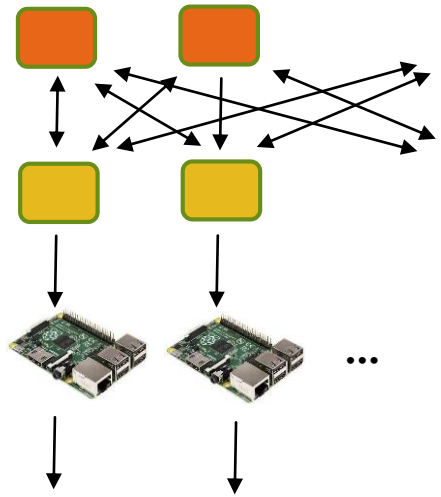
Game Needs

What does the local Rpi/GCOS team do specifically?

Grand Challenge
Command Line and Web

Grand Challenge
Operating System (GCOS)

IoT Tier



Specific Circuit “Basis”

(specific circuit)

Logical Game “Resource”

(1 circuit -> n logical resources)

tornado_warning
flood_alert

community_meeting
tornado_siren

a) Work with Player/Game Designer
   - Create logical resource required
   - Create circuit designs to support logical resources
   - Hopefully one physical circuit support many logical “resource(s)”

b) Design the circuit on the Rpi (base circuit – “basis”)
   - GPIO 24 -> Led -> Resistor -> Ground
   - Test circuit
   - Add to reference set of circuits available & ensure no conflicts

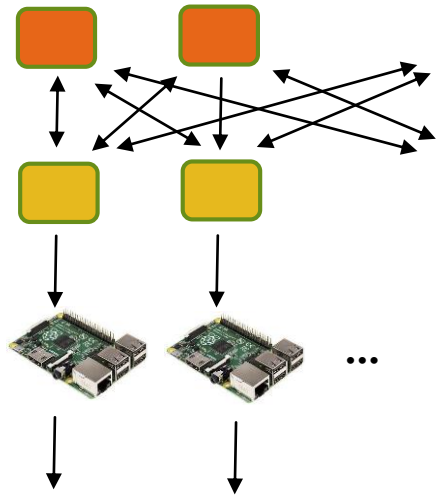
c) Extend GCOS to support PI/Circuit with code
   - Design software to initialize, control and unload to implement the “basis” of each resource
   - Test, deploy and train players to use the solution
   - Integrate basis/resource to report data to Game web application

What does the local Rpi/GCOS team do specifically? (cont)

Grand Challenge
Command Line and Web

Grand Challenge
Operating System (GCOS)

IoT Tier



Specific Circuit “Basis”

(specific circuit)

red_led siren_function

Logical Game “Resource”

(1 circuit -> n logical resources)

tornado_warning
flood_alert

community_meeting
tornado_siren

a) Create resource in global configuration
- Update gcos\configuration.json
- For each node/player, add logical resource
- Reference the basis and instance of basis used (can have more than once instance)
- Manage the central file and ensure all Rpi units acquire new standard file

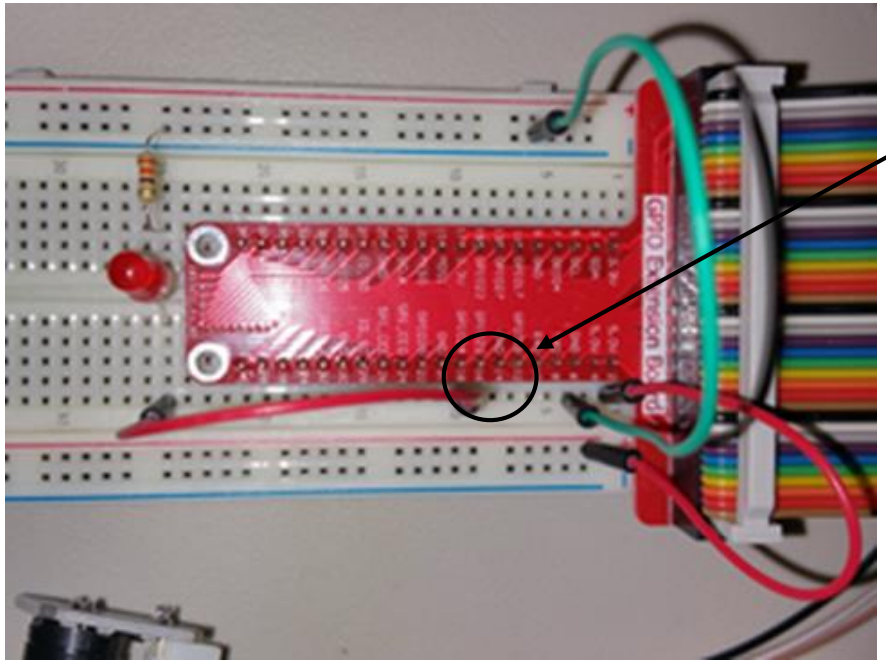
b) Design the circuit on the Rpi (base circuit – “basis”)
- GPIO 24 -> Led -> Resistor -> Ground
- Test circuit
- Add to reference set of circuits available & ensure no conflicts

c) Extend GCOS to support PI/Circuit with code
- Create an extension program that talks to the “basis” circuit
- Test the functions via GCOS to interact with the code that drives the circuit
- Code extension for each basis will be located in gcos\ext
- Code dynamically pulled into GCOS at gcos startup on Rpi

GCOS resource -> basis configuration example

configuration.json

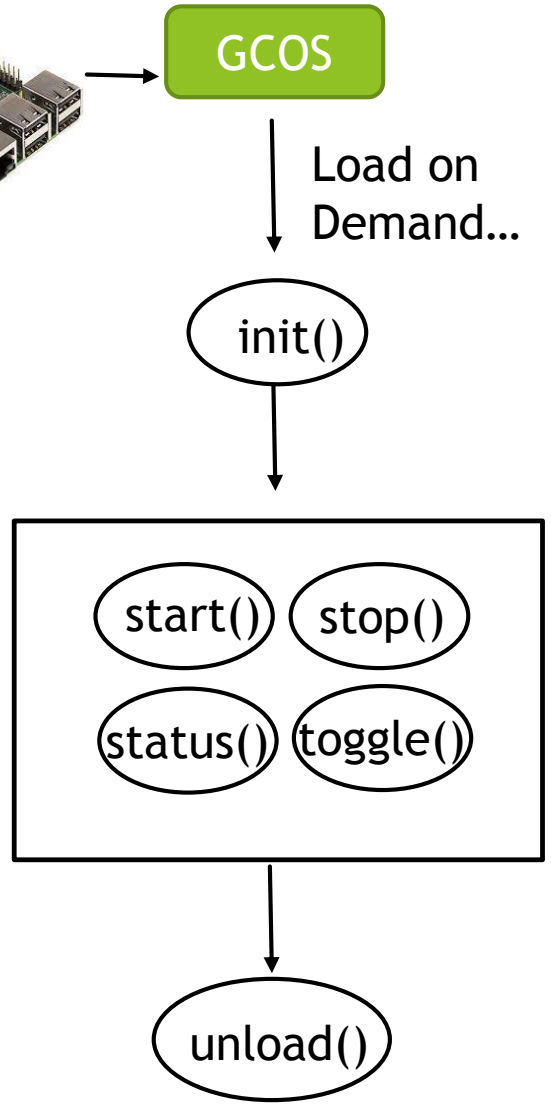
```
{
    "player_1_pi": {
        "resources": {
            "internal_IT" : {
                "ip"           : "192.168.1.111",
                "port"         : "8085",
                "gcos_root_uri" : "/api/v01"
            },
            "red_warning_light": {
                "basis" : "led_light",
                "instance" : "1",
                "gpio_port": "24",
                "mode": "out",
                "on": "1",
                "off": "0"
            },
            "yellow_warn_light": {
                "basis" : "led_yellow",
                "instance" : "1",
                "gpio_port": "25",
                "mode": "out",
                "on": "1",
                "off": "0"
            },
```

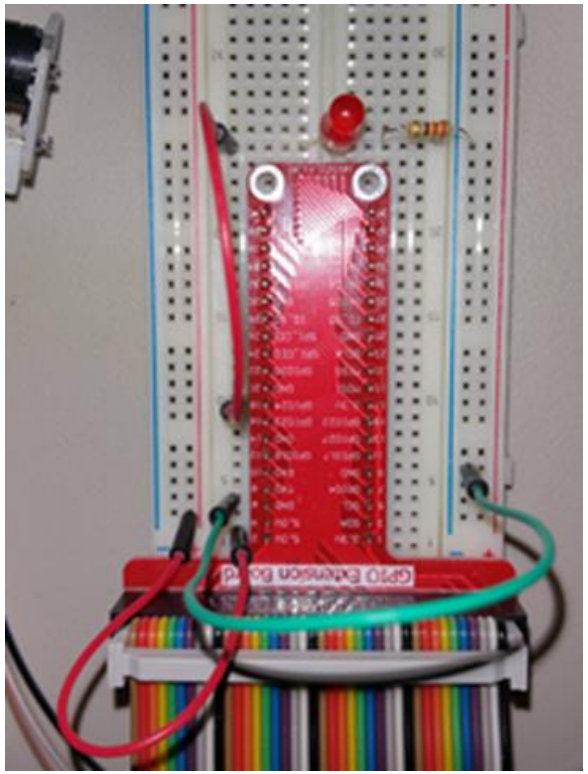


Add a resource to global configuration

- Update gcos\configuration.json (to left)
- Each Rpi is related to player currently “player_1_pi”
- Each Rpi offers a set of resources and underlying basis circuits for all resources
- “red_warning_light” is an example resource
- There could be many uses of a red warning light, each having their own name and identity to the gamers and tied to grand challenge
- The “basis” describes the code and Rpi circuit supporting the logical resource
- There can be more than once instance of this circuit (instance = 1) in this case, the resource will be tied to a basis instance
- Eventually the basis could be shared or exclusively held by players
- gpio_port (24, circled on left) and mode (output) are used to configure and use circuit during runtime and required by Rpi
- The other attribute are for future
- The current version only supports one gpio per circuit, a limitation that will be corrected soon
- Manage the central file and ensure all Rpi units acquire new standard file

Extending GCOS & creating a basis-> life cycle for “led_light” example





Life Cycle of a GCOS Rpi Basis Circuit

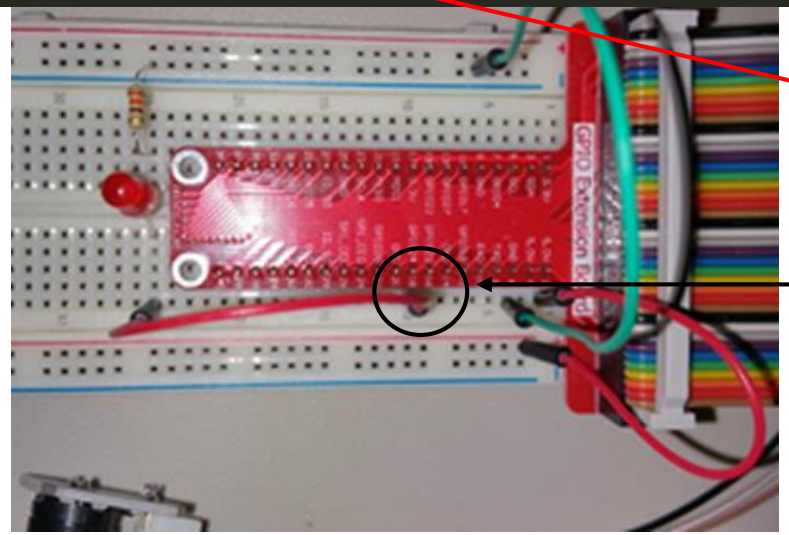
- 1) Phase 1
  - Load basis module and initialize
  - => init() method – initialize the GPIO pin, set mode
- 2) start – start the circuit, or
- 2) stop – stop the circuit, or
- 2) status – query basis for status data, or
- 2) toggle – cycle state swap between on/off, or
- 2)/3) unload – removes reference & cleanup

- Status Considerations
  - init() call should be done first and will pass in GPIO pin and output direction data direction (“in”, “out”)
  - After init, all other methods can be called in the life cycle diagram (even unload(), but why?)
  - Typically, status() maybe called the most often, start(), stop() and toggle() to control state in normal operation
- Programmer will create a “led_light.js” file from standard template, implement each method
- GCOS can support as many circuits as you can create

Extending GCOS basis-> code “init” example to for “led_light”

led_light.js

```
36  // init function - used to establish the gpio such that can start, stop and toggle the device
37  //
38  //                  The core GPIO command is provided, and return data. The data can be augmented
39  //                  as needed.
40  //
41  function init(gpio_mode, gpio_pin) {    // TODO PERRY  Need to handle > 1 GPIO pin, e.g could enable several
42
43   debug("Module [" + module_name + "] init [Mode: " + gpio_mode + " Pin: " + gpio_pin + "]");
44
45   try {
46    // Gpio = require("onoff").Gpio, led = new Gpio(gpio_pin, gpio_mode),  iv;
47    Gpio = require('onoff').Gpio, // Constructor function for Gpio objects.
48    led = new Gpio(gpio_pin, gpio_mode);         // Export GPIO #14 as an output.
49   } catch (err) {
50    console.log("Error: " + err);
51    return (err);
52   }
53
54   debug("init complete");
55
56    var data_response = {
57    "module"      : module_name,
58    "method"      : "init",
59    "status"      : "OK",
60    "return_code" : "200"
61   };
62
63   return(data_response);
64  }
65
```



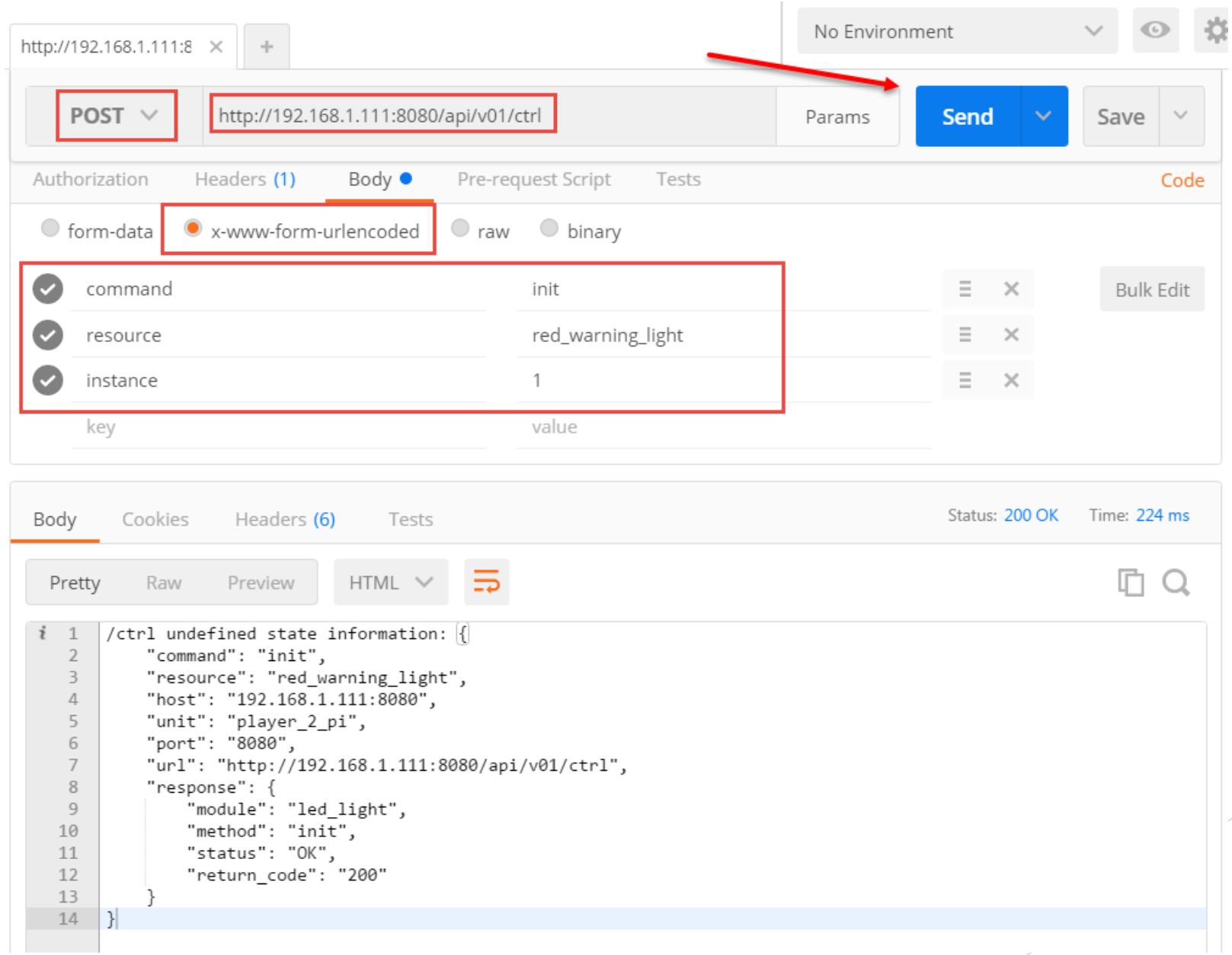
Extension File for each “basis”

- Team members create a nodejs module
- The following methods need to be implemented by the user
  - init – initialize the GPIO pin, set mode
  - start – start the circuit
  - stop – stop the circuit
  - toggle – cycle state swap between on/off
  - unload – removes reference, cleanup
- There is an example to help, the init function is shown on the left
  - init() call passing in GPIO pin and output direction (“in”, “out”)
  - Debug statements should be added for testing and can be dynamically turned on if problems occur during operation
  - Logic that needs to be ran, in the init case the npm onoff package is used and communicates with the Rpi gpio circuit
  - Data should be created reflecting what happened and is added to what the GCOS applications that use the data when function returns
- Programmer will create a “led_light.js” file from standard and adjust methods

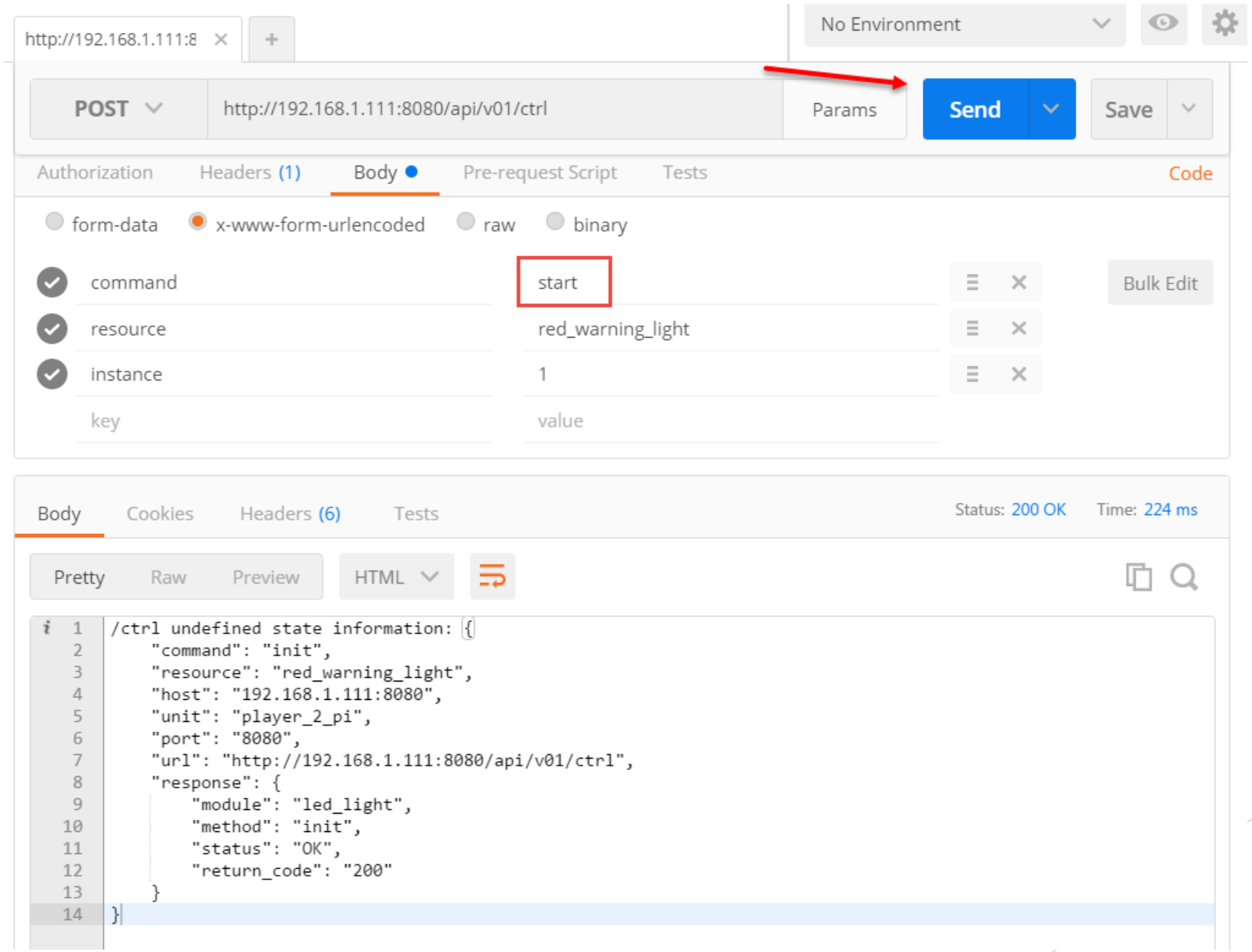
Debugging Tools – Postman (cli coming soon)

- Install Postman
  - Open Chrome
  - Chrome Apps
  - Search and install Postman
- Run PostMan
  - Chrome Apps
  - Select Postman
  - See the next screen (use after starting gcos via the Readme.MD)

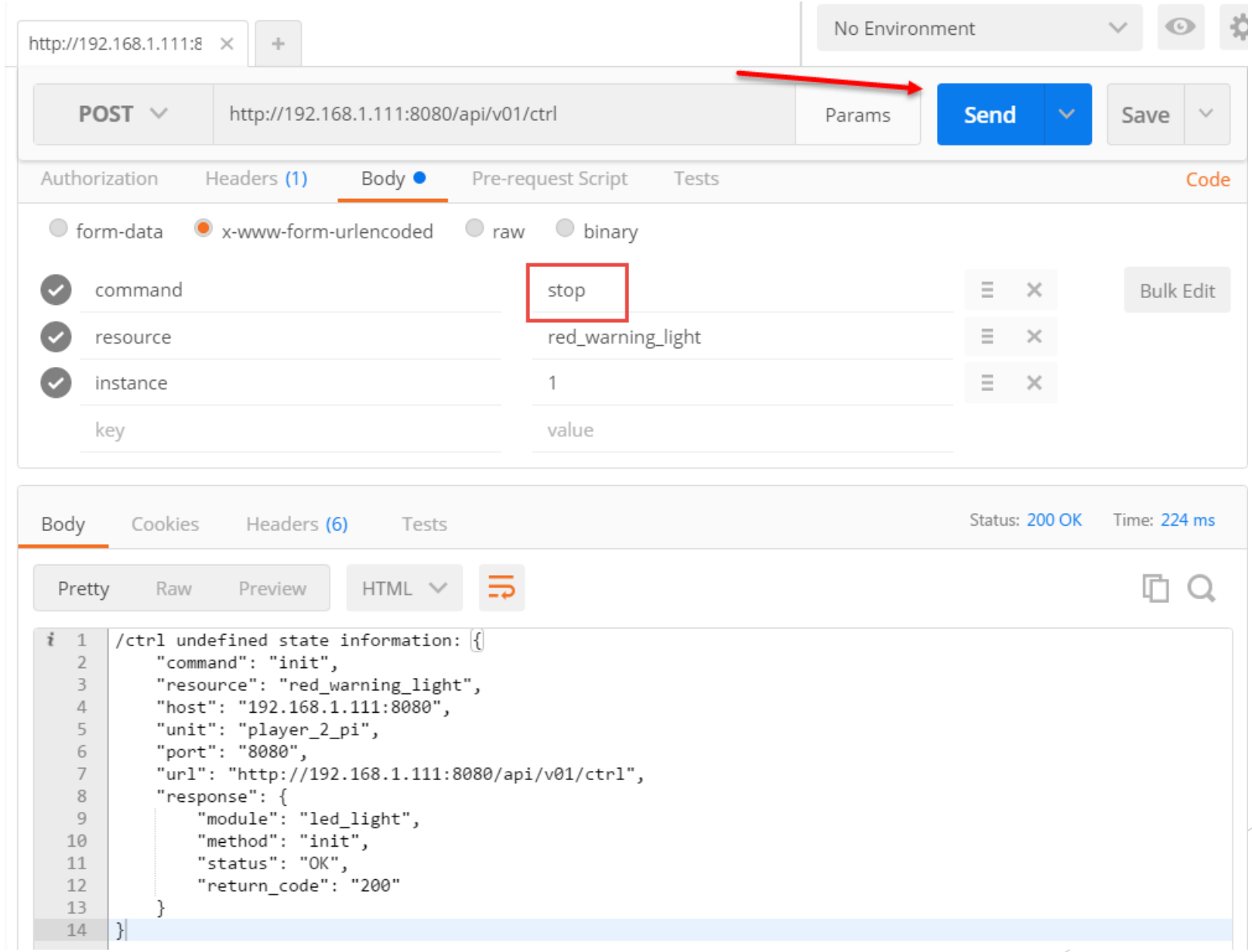
Initialize a resource (sample included, make a led circuit attached to gpio 24)



Start a resource: (light should turn on)



Stop a resource: (light should turn off)



- More Commands
  - manage – command line tool to target IoT device resources
  - gcos – command to start the server on a given instance
- Operations the rest api and command line support
  - init, start, stop provided as diretive
  - toggle, unload are used as well
    - Toggle switches the value from on -> off, off -> on
    - Unload disables the circuit and user must run init again
  - Init and Unload are done once, then use for many times and when wrapping up unload for example