Internet of Things Operating System (iotos)

Distributed Web Infrastructure & Device Control (based on Nodejs)

Getting started

pdykes@gmail.com (12/15/2016, v0.0.4)

Youtube reference coming soon.

Basic Overview - iotos

- Objectives
  - Create a straightforward (potentially simple) extensible, distributed solution to utilize Raspberry Pis and Attached Circuits more effectively
  - Assist teams that wish to focus on basic function, yet enable features that are required, yet challenging and often hamper effective use at scale
  - Do not limit the programming model for attached circuits, e.g. supports any sort of GPIO and PWM programing library (in fact, support multiple concurrently), often users can code their solution in javascript with simple reorganization reuse in iotos.
  - All users to create various circuits across potentially 100+ Pis and virtually control, obtain status and route status to target endpoints based on results
  - Enable dashboards that monitor the results and other resources to describe the findings with basic JSON and restful apis without overburdening the “builders”
- Critical Information
  - GitHub Location: https://github.com/pdykes/iotos
  - User creates circuits, creates javascript functions to interact with the circuit, and correlates circuit and code to a logical resources referenceable via http and cli.
  - Output can be consumed by any solution utilizing JSON.

Overview

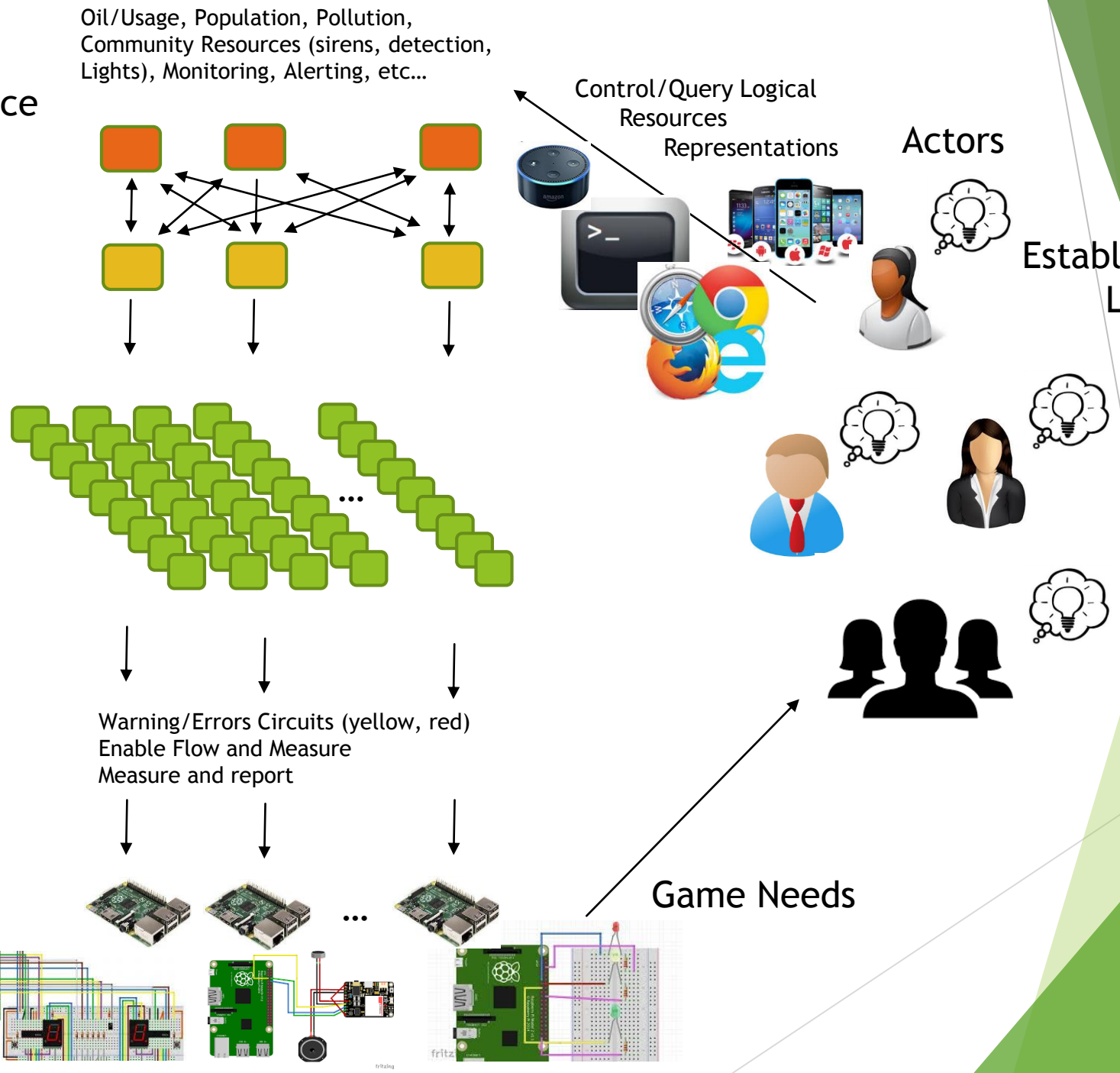
Iotos Command Line, Voice and Web resources

Iotos Operating System

Support a fabric of resources (circuits) which implement user capabilities

Logical Game IoT Circuits

IoT Tier

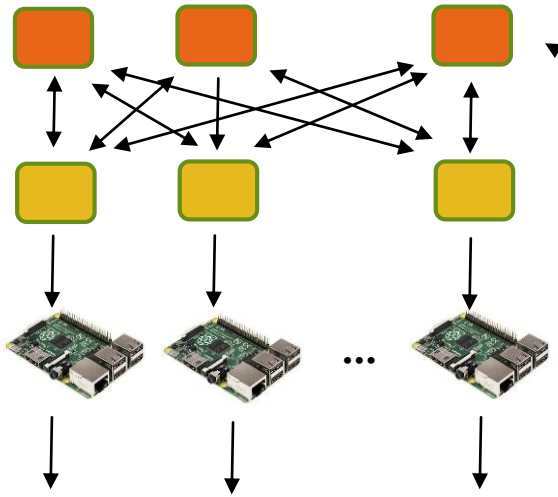


Bottom Up Example

Iotos Voice, Command Line and Web

iotos
Operating System

IoT Tier



red_led siren_function

Specific Circuit “Basis”

(specific circuit)

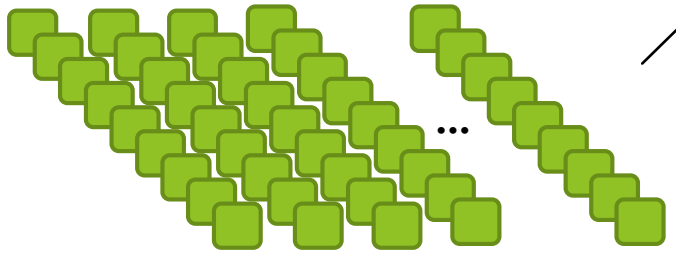
Logical Game “Resource”

(1 circuit -> n logical resources)

tornado_warning
flood_alert

community_meeting
tornado_siren

Logical Resources



Actors

Use/Access Technology

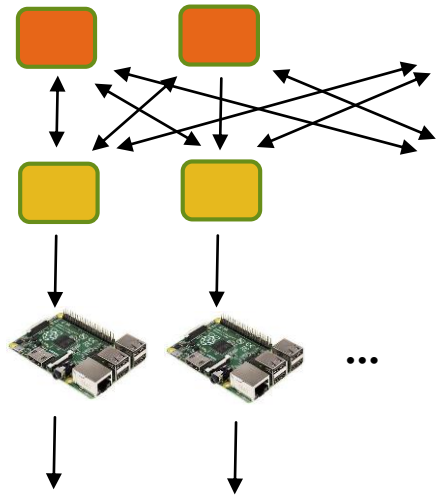
Game Needs

What does the local team using this solution do specifically?

Iotos Voice, Command Line and Web

iotos
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) Design the high level resources required
- Create logical resource required
- Create circuit designs to support logical resources
- One physical circuit support many logical “resource(s)”

b) Design the physical circuits on Raspberry Pi
- Using circuit, gpio board, breadboard, create any circuit
- Example: GPIO 24 -> Led -> Resistor -> Ground
- Test circuit, write Node script to initialize, start, stop (based on samples)
- Determine which Pi will host which circuits (up to 100+ fine)

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

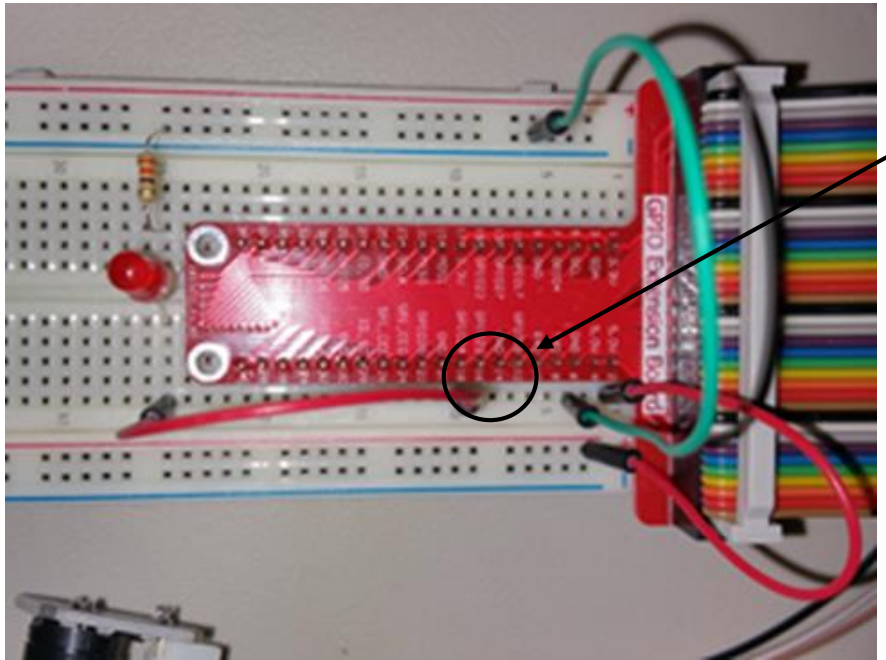
d) Configure resource to be access via network
- Create logical resource reference via common methods
- Correlate logical resource to circuit basis
- Direct if/where resource should send status in real-time
- Control and Monitor resources in real-time

e) Operate resources via network & utilize generated data
- Update global configuration file and distribute
  - Pis and all CLI agents (windows and other systems)
- Start iotos on all Rpi devices
- Initiate all resources
- Monitor the results and act accordingly

Creating a resource -> basis (basic circuit) & 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"
            },
```



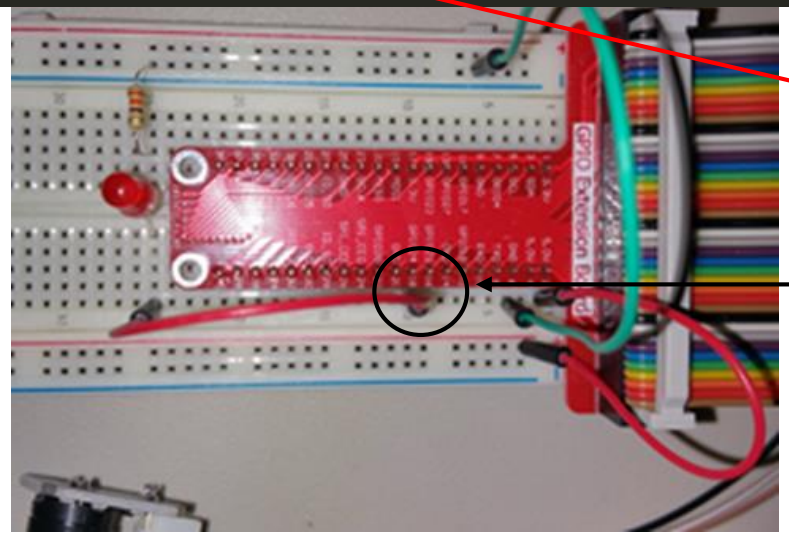
Add a resource to global configuration

- Update gcos\configuration.json (to left)
- Each Rpi is related to actor currently “actor_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 actors
- 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

Basis (circuit) -> code “init” example to for “red_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

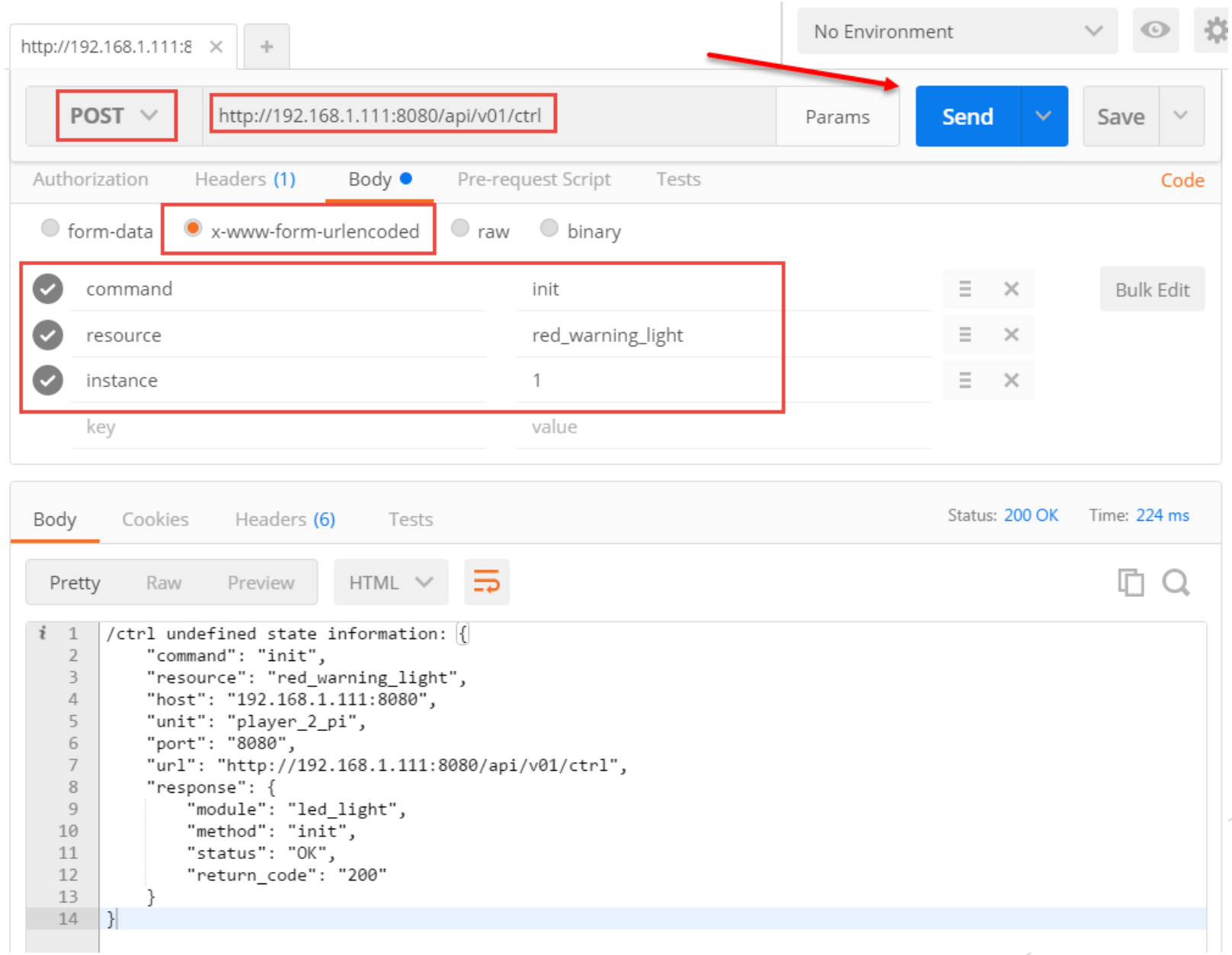
Command Line Setup and Examples

- Setup
  - See Readme.MD (or default page on github web site (shown earlier) that formats it nicely)
    - Common question: on windows, remove onoff package.json before perming “onoff” package.
  - Assume for below:
    - Have an LED circuit on GPIO port 24, that is a basic LED -> Resistor -> Ground. Also, have window open and in the project directory (assume projects\iotos), nodejs 6.9+ higher installed, and “npm install” performed in project directory
    - Project directory iotos.ini configured on each system, include name of actor/target. Also, update the configuration.json file for each target w/IP address (use text editor).
- Basic
  - Open window, Start Server: nodejs iotos.js (or node iotos.js on some systems)
  - Open 2nd window, Command line (note configuration.json and iotos.ini should be configured at this point:
    - Initialize resource red_warning_light on player_1_pi
      - nodejs manage.js -t player_1_pi -r red_warning_light -d init
    - Start resource red_warning_light
      - nodejs manage.js -t player_1_pi -r red_warning_light -d start
    - More commands:
      - nodejs manage.js -t player_1_pi -r red_warning_light -d toggle
      - nodejs manage.js -t player_1_pi -r red_warning_light -d stop
      - nodejs manage.js -t player_1_pi -r red_warning_light -d unload

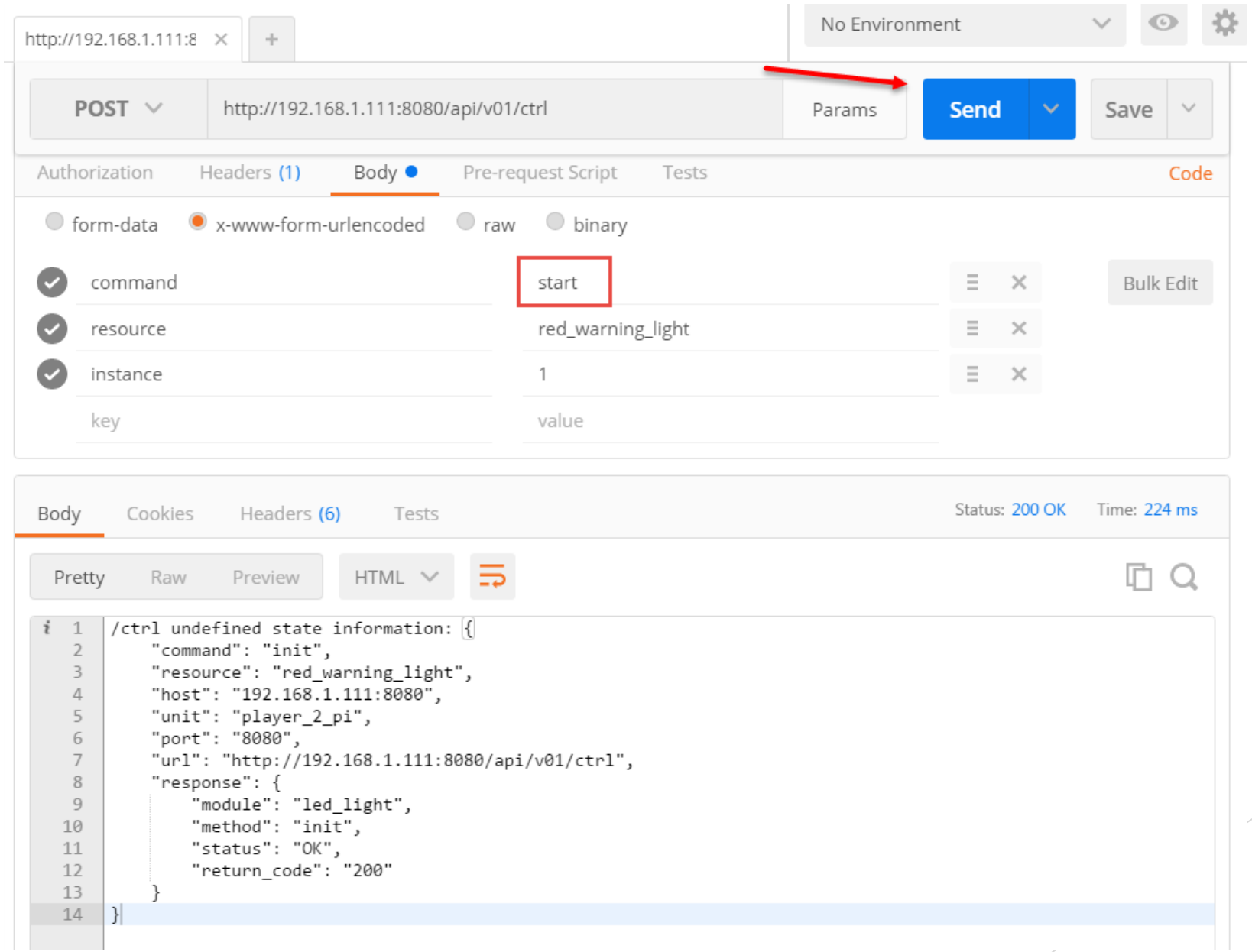
Basic Web Interaction

- 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)

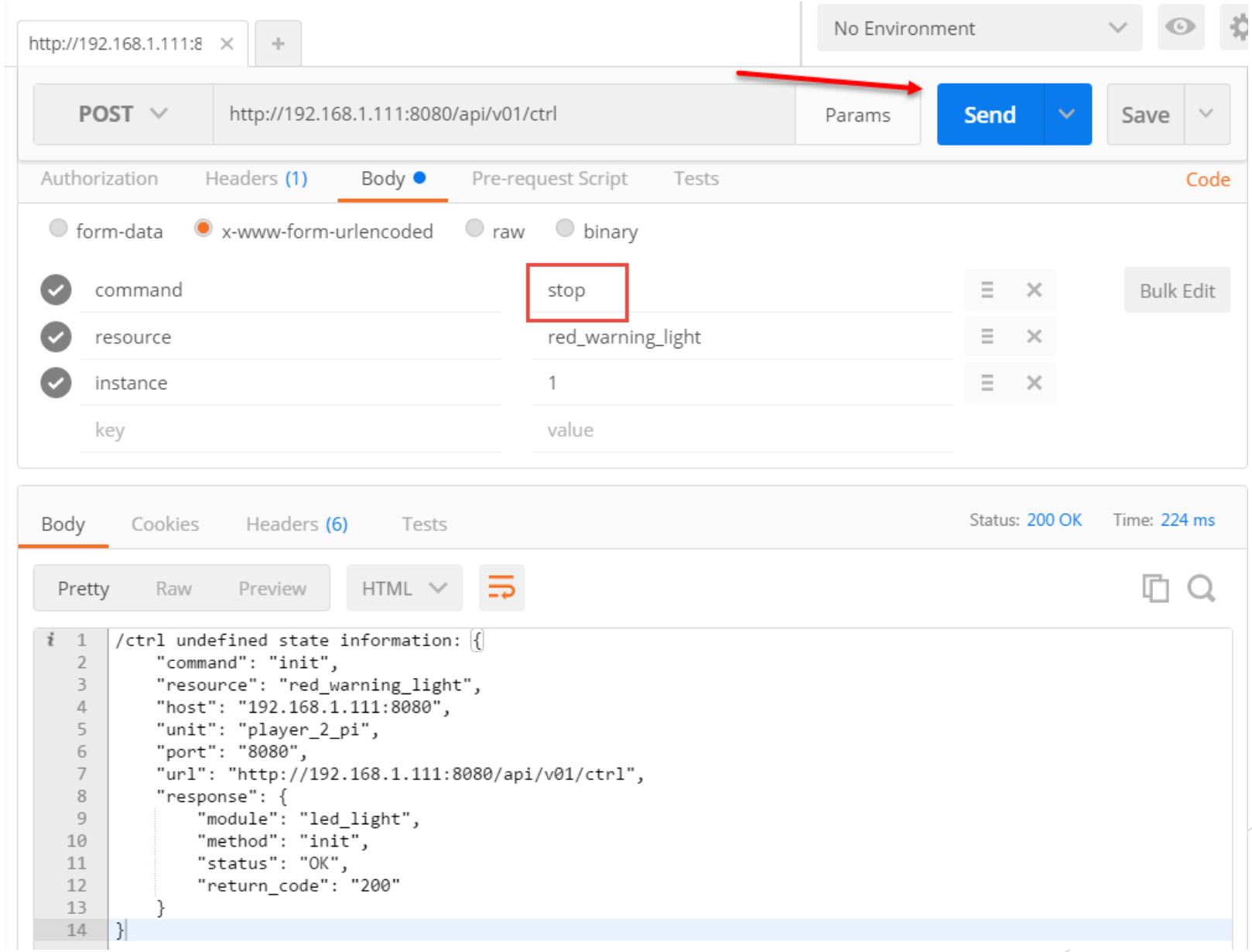
Initialize a resource (sample included, make a led circuit attached to gpio 24)



Start a resource: (light should turn on)



Stop a resource: (light should turn off)



- More Commands
  - init, start, stop provided above
  - 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