

Internet of Things Operating System (iotos)

Distributed Web Infrastructure & Device Control (based on Nodejs)

Getting started

pdykes@gmail.com (12/15/2016, v0.0.3)

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 programming 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

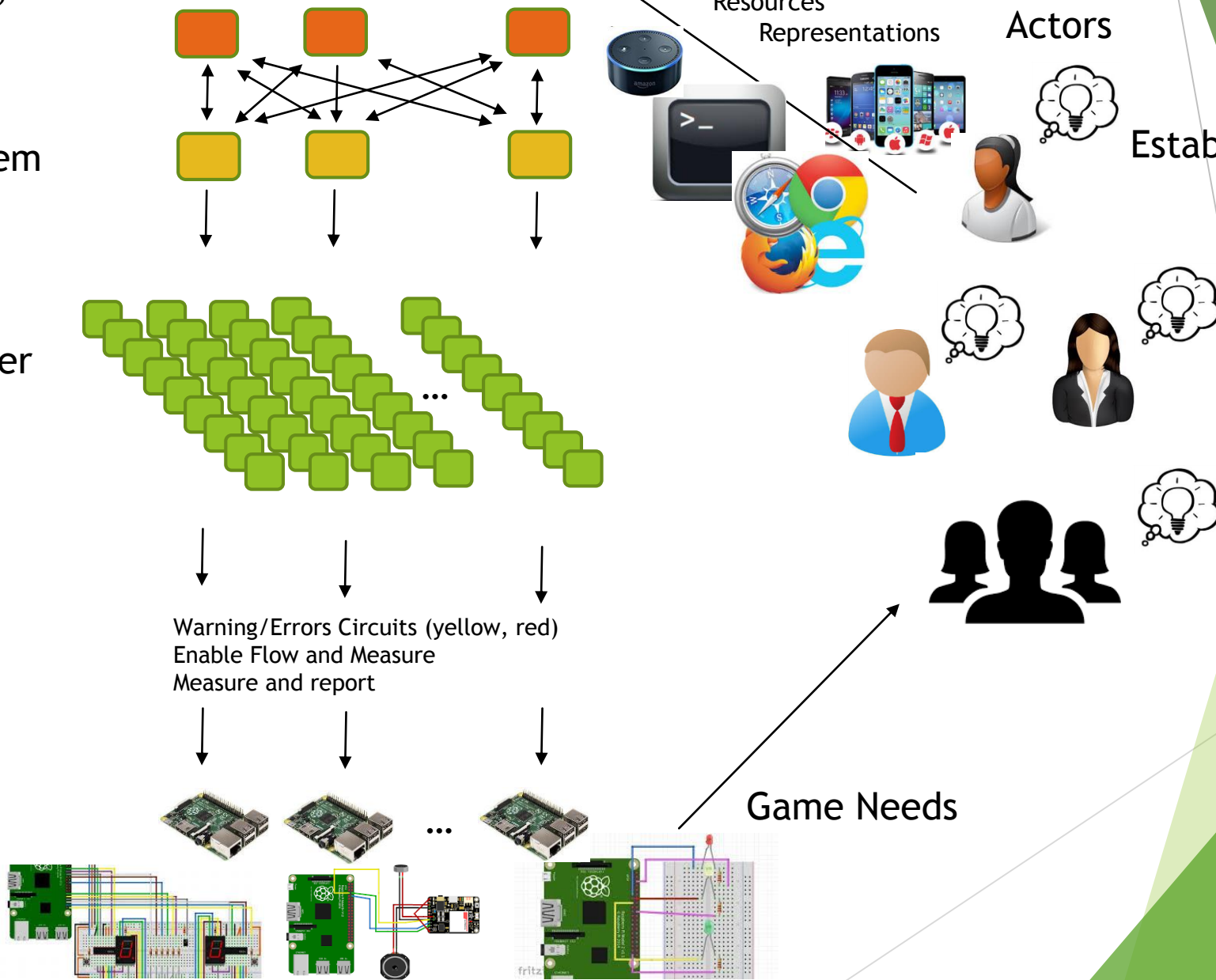
Iotos Operating System

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

Logical Game IoT Circuits

IoT Tier

Oil/Usage, Population, Pollution, Community Resources (sirens, detection, Lights), Monitoring, Alerting, etc...



Bottom Up Example

Iotos Voice, Command Line
and Web

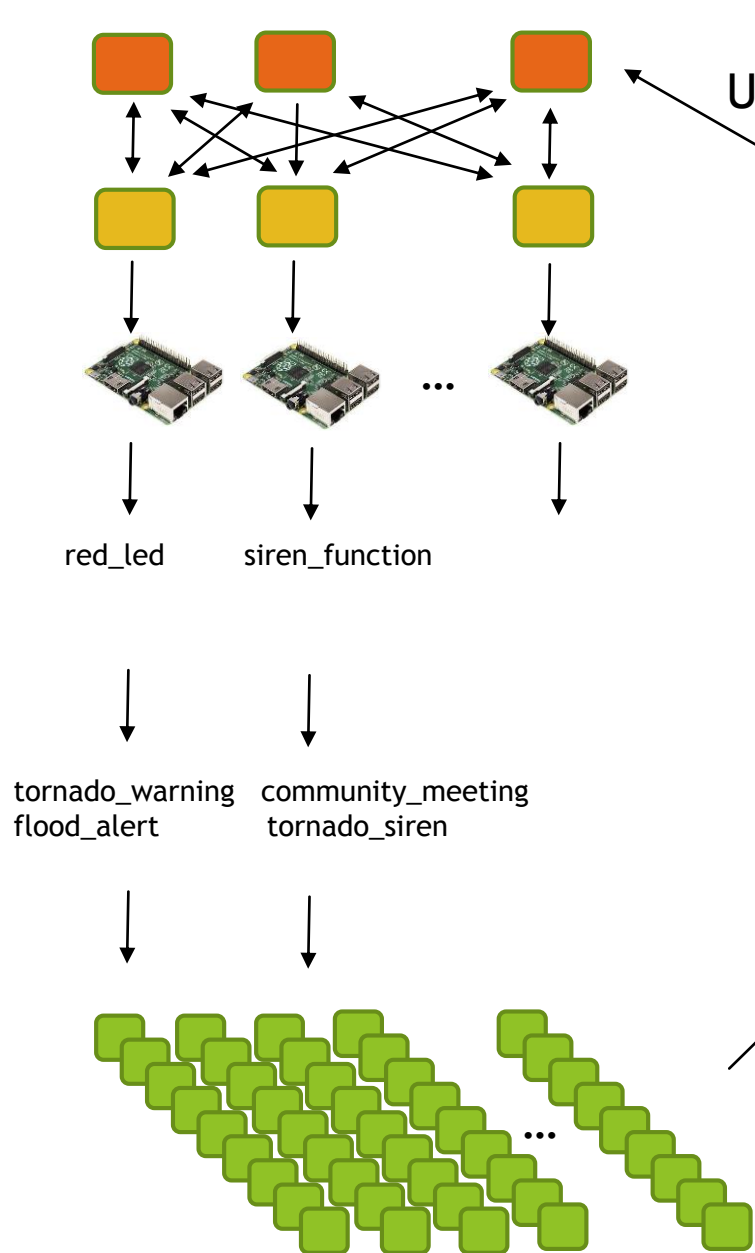
iotos
Operating System

IoT Tier

Specific Circuit “Basis”
(specific circuit)

Logical Game “Resource”
(1 circuit -> n logical
resources)

Logical Resources

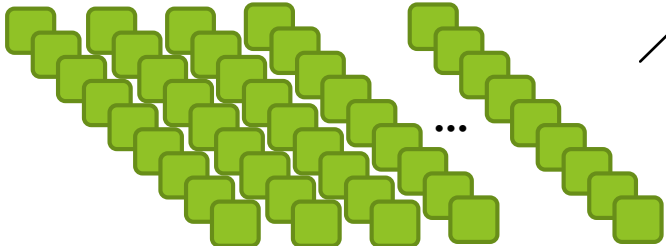


Use/Access
Technology

Actors



Game Needs



What does the local team using this solution do specifically?

Iotos Voice, Command Line
and Web

iotos
Operating System (GCOS)

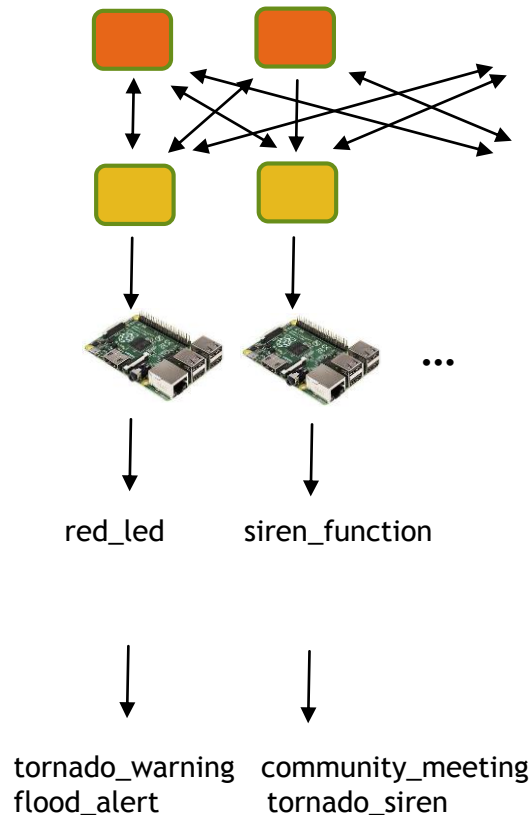
IoT Tier

Specific Circuit “Basis”

(specific circuit)

Logical Game “Resource”

(1 circuit -> n logical
resources)



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

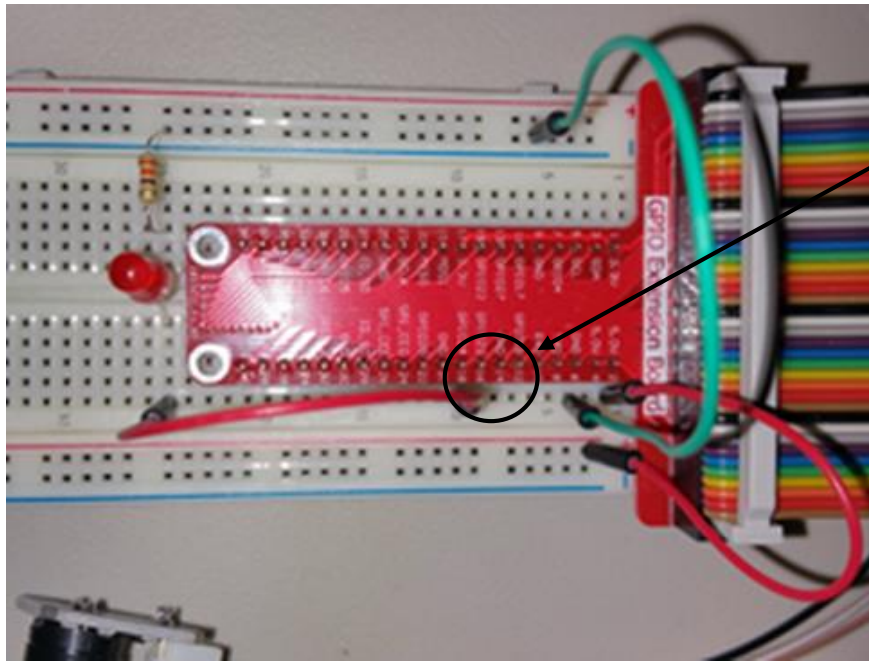
- Update global configuration file and distribute
 - Pis and all CLI agents (windows and other systems)
- Start iotos on all Rpi devices
- Initiate all resources
- Monitor the results and act accordingly

Creating a resource -> basis (basic circuit) & configuration example

```
1 {
2   "player_1_pi": {
3     "resources": {
4       "internal_IT" : {
5         "ip"       : "192.168.1.111",
6         "port"     : "8085",
7         "gcoss_root_uri" : "/api/v01"
8       },
9       "red_warning_light": {
10        "basis" : "led_light",
11        "instance" : "1",
12        "gpio_port": "24",
13        "mode": "out",
14        "on": "1",
15        "off": "0"
16      },
17      "yellow_warn_light": {
18        "basis" : "led_yellow",
19        "instance" : "1",
20        "gpio_port": "25",
21        "mode": "out",
22        "on": "1",
23        "off": "0"
24      },
25    }
26  }
27 }
```

Add a resource to global configuration

- Update gcoss\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

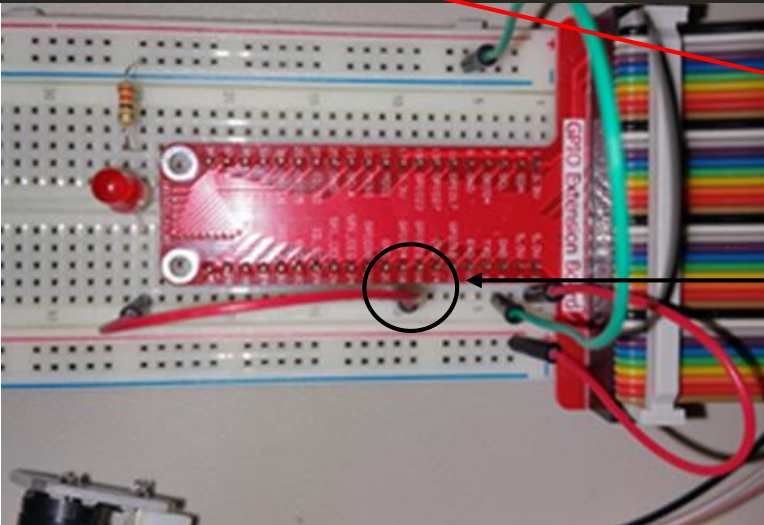


Basis (circuit) -> code “init” example to for “red_light”

```
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 Example

- ▶ Basic

- ▶ 1
- ▶ 2
- ▶ 3

- ▶ Advanced

- ▶ Chrome Apps
- ▶ Select Postman
- ▶ See the next screen (use after starting gcos via the Readme.MD)

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 gcoss via the Readme.MD)

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

The screenshot displays a REST client interface with the following details:

- URL:** `http://192.168.1.111:8080/api/v01/ctrl`
- Method:** `POST`
- Authorization:** No Environment
- Headers:** 1 header is present.
- Body:** `x-www-form-urlencoded` format.
- Parameters:** A table with 3 entries:

key	value
command	init
resource	red_warning_light
instance	1

The response status is `200 OK` with a time of `224 ms`. The response body is shown in the following JSON format:

```
{
  "/ctrl undefined state information: {
    "command": "init",
    "resource": "red_warning_light",
    "host": "192.168.1.111:8080",
    "unit": "player_2_pi",
    "port": "8080",
    "url": "http://192.168.1.111:8080/api/v01/ctrl",
    "response": {
      "module": "led_light",
      "method": "init",
      "status": "OK",
      "return_code": "200"
    }
  }
}
```

Start a resource: (light should turn on)

The screenshot displays a REST client interface with the following details:

- URL Bar:** `http://192.168.1.111:8` and `http://192.168.1.111:8080/api/v01/ctrl`.
- Method:** `POST`.
- Environment:** `No Environment`.
- Body Tab:** Selected, showing `x-www-form-urlencoded` format.
- Form Fields:**
 - `command`: `start` (highlighted with a red box)
 - `resource`: `red_warning_light`
 - `instance`: `1`
 - `key`: `value`
- Response Tab:** Shows `Status: 200 OK` and `Time: 224 ms`.
- Response Body:** A JSON object containing metadata and response details.

```
1 /ctrl undefined state information: {  
2   "command": "init",  
3   "resource": "red_warning_light",  
4   "host": "192.168.1.111:8080",  
5   "unit": "player_2_pi",  
6   "port": "8080",  
7   "url": "http://192.168.1.111:8080/api/v01/ctrl",  
8   "response": {  
9     "module": "led_light",  
10    "method": "init",  
11    "status": "OK",  
12    "return_code": "200"  
13  }  
14 }
```

Stop a resource: (light should turn off)

The screenshot shows a REST client interface with a POST request to `http://192.168.1.111:8080/api/v01/ctrl`. The request body is `x-www-form-urlencoded` and contains the following data:

key	value
command	stop
resource	red_warning_light
instance	1

A red arrow points to the `Send` button. The response status is `200 OK` and the time is `224 ms`. The response body is shown in the `Body` tab, displaying the following JSON:

```
{
  "ctrl": {
    "command": "init",
    "resource": "red_warning_light",
    "host": "192.168.1.111:8080",
    "unit": "player_2_pi",
    "port": "8080",
    "url": "http://192.168.1.111:8080/api/v01/ctrl",
    "response": {
      "module": "led_light",
      "method": "init",
      "status": "OK",
      "return_code": "200"
    }
  }
}
```

- 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