electric **imp**

Matt Haines

matt@electricimp.com

Slides, Code, etc

- https://github.com/beardedinventor/imp201

- *BlinkUp Credentials*
  - SSID: impdemo
  - PW: electric

What are we looking at today?

- Sleeping
  - The sleep methods
  - Lazy connections
  - The *nv* table
- Building APIs
  - The request object
  - Simple Security
  - Rocky
- External Web Services
  - http.request API
  - Classes, callbacks, and scope

Pt. 1 - Sleeping

Agent Code

We’re going to use the following agent code in all of our sleep examples:

/sleeping/sleep.agent.nut

Sleeping

- **imp.sleep**(timeout)
  - Shallow sleeps for *timeout* seconds
  - Code execution continues after the sleep method completes
- **imp.wakeup**(timeout, callback)
  - Tells imp to execute *callback* in *timeout* seconds.
- **server.sleepfor**(sleepTime)
  - Deep sleep (imp disconnects and goes to sleep)
  - Imp informs server of deep sleep
- **imp.deepsleepfor**(sleepTime)
  - Deep sleep (imp disconnects and goes to sleep)
  - Imp doesn't inform server of deep sleep

imp.sleep

/sleeping/1_impsleep.device.nut

Why does this stop working?

- The imp is single threaded
- OS tasks + developer code share the thread

- **imp.sleep** blocks, so the imp can't process messages while we're sleeping
- We call **readAndSend()** inline, so we never actually yield the thread

imp.wakeup

/sleeping/2_impwakeup.device.nut

Why is this better?

- **imp.wakeup** is non-blocking
  - It tells the imp to schedule some work (the *callback*) to execute at some time in the future (after the *timeout*)
- **imp.wakeup** yields the thread (which means OS level tasks, and other messages can be processed)

server.sleepfor

/sleeping/3_serversleepfor.device.nut

server.sleepfor

- **server.sleepfor** does two things:
  - Puts the imp into a deepsleep for the specified period of time
  - Sends a message to the server indicating how long it will be asleep for

```
[Status] sleeping until 1433784145000
```

imp.deepsleepfor

/sleeping/4_impdeepsleepfor.device.nut

imp.deepsleepfor

- imp.deepsleepfor works identically to server.sleepfor, but **it doesn't inform the server we're going to sleep**

- **Why do we care?**
  - Sending messages takes extra power
  - Sending messages means we need to connect

Lazy Connects

- On a warm boot*, the imp won't connect until we use a command that requires a connection:
  - **server.log**
  - **agent.send**
  - **server.sleepfor**
  - …

* Warm boots are after deep sleeps

* Cold boots are after power cycles

Lazy Connects

/sleeping/5_lazyconnect.device.nut

Lazy Connects

- After you build and run, it's not going to look like your imp is waking up anymore
- LEDs don't blink because it's not connecting

- We'll need to push-push (power cycle) in order to force the imp to connect and download new code

The *nv* Table

- When the imp goes into deepsleep, it looses all state, and begins executing code from the top on wake

- The nv table let's us store a small amount of state information/data

The *nv* Table

/sleeping/6_nvtable.device.nut

Pt. 2 – Agent Driven APIs

Device Code

We’re going to use the following agent code in all of our sleep examples:

/apis/apis.device.nut

Basic APIs

- Getting Started Guide goes through a really simple API example
  - Ignores path
  - Ignore verb
  - No security
  - Uses query parameters

Basic API

/api/1_basicapi.agent.nut

The *request* Object

/api/2_request.agent.nut

Using Paths

- Create two paths:
  - / returns current state
  - /set lets you set state with ?state

API with Paths

/api/3_paths.agent.nut

Using Headers for Security

- Let's add some security
- Requests will need a "X-API-KEY" header
- We're going to hardcode the required API-Key

Using Headers for Security

/api/4_simplesecurity.agent.nut

Endpoint Specific Security

- In the real world, we often need different authentication methods for different end points.

- We're going to modify the API so that it only checks for the X-API-KEY when we're trying to set the light

Endpoint Specific Security

/api/5_endpointsecurity.agent.nut

This is not maintainable!

- We've setup two endpoints, and the code is already a little tricky to follow

- A real product might have 5-10+ endpoints with more complicated user access control

Rocky

- https://github.com/electricimp/rocky
- #require "Rocky.class.nut:1.1.1"

- Allows you to specify behavior by route
- Easily add authentication methods to routes
  – Easily handle what happens on unauthorized requests
- Manage error handling, etc.

Rocky

/api/6_rocky.agent.nut

Pt. 3 - Working with web services

We'll be using IFTTT's Maker Channel
(shhhhhh…)

https://ifttt.com/maker

HTTP Requests

- Agents have six APIs to help build requests:
  - http.get(*url, [headers]*)
  - http.put(*url, [headers, body]*)
  - http.post(*url, [headers, body]*)
  - http.httpdelete(*url, [headers]*)
  - http.request(*verb, url, [headers, body]*);

HTTP Requests

/webservices/1_webserviceFunction.agent.nut

Web Service Classes

- Instantiate object with required credentials
- All requests should be made asynchronously
- Callbacks for requests must take at least two parameters:
  - **err:** *null* on success, or a string describing the error
  - **response:** The HTTP Response Object
  - It's recommended to also require a third parameter:
    - **data:** the decoded data from the request

Web Service Classes

/webservices/2_webserviceClass.agent.nut

How to communicate with X Service?

- HTTPS
  - Inbound data (http.onrequest)
    - Service -> Agent -> Device
  - Outbound data (http.request)
    - Device -> Agent -> Service

Resources

• imp.wakeup	https://electricimp.com/docs/api/imp/wakeup
• imp.sleep	https://electricimp.com/docs/api/imp/sleep
• server.sleepfor	https://electricimp.com/docs/api/server/sleepfor
• imp.deepsleepfor	https://electricimp.com/docs/api/imp/deepsleepfor
• http.onrequest	https://electricimp.com/docs/api/http/onrequest
• Rocky	https://github.com/electricimp/rocky
• http.request	https://electricimp.com/docs/api/http/request
• Libraries	https://electricimp.com/docs/api/examples/libraries

Questions?