Firefly Documentation

Release a0.1

Zachary Priddy

Jul 16, 2017

Contents

1 Events **3**
1.1 Standard Event Format . . . 3
1.2 Sending an Event . . . 3

2 Plugins **5**
2.1 Device Plugin . . . 5
2.2 App Plugin . . . 7

3 Location **9**
3.1 Location Scheduler . . . 9
3.2 Mode Handling . . . 9
3.3 Time . . . 9
3.4 Sun Status . . . 9

4 Scheduler **11**
4.1 Global Scheduler . . . 11
4.2 Instance Scheduler . . . 11
4.3 Scheduler Syntax . . . 11

5 Config **13**
5.1 Pushover Config . . . 13
5.2 SmartThings Config . . . 13

Contents:

CHAPTER 1

Events

Standard Event Format

This is how event messages should be formatted:

```
{'dimmer':{'state':'on', 'level':'100'}}
```

This should follow the format of:

```
{'EVENT DEVICE TYPE': {'EVENT VAR': 'EVENT VAL'}}
```

Sending an Event

NOTE: In next release send_direct_event will be replaced with send_event

All files that require sending an event should follow the format:

```
from core.untils import send_direct_event

...

send_direct_event(<<DEVICE>>, <<EVENT IN STANDARD EVENT FORMAT>>)
```

Sample Event:

```
from core.utils import send_direct_event

...

send_direct_event('Kitchen Motion', {'motion':{'state':'active'}})
```

CHAPTER 2

Plugins

There are two different types of plugins:

- Device Plugin
- App Plugin

Plugins have a small set of required functions and libraries in order to function correcly. Besides these required functions there are no limits on what can be used with them. Below there are sample templates for the different plgin types.

NOTE: We do suggest using treq instead of requests to keep things running faster.

Difference between device plugin and app plugin:

Device Plugin:

- Can reviece commands
- Can send events to the Firefly controller

App Plugin:

- Can receive events. (ie triggers)
- Can subscribe to events from devices

Device Plugin

There are two parts to the device plugins:

1. The device its self
2. The device installer script

Device installer scripts can be used to install more than just one device. i.e: SmartThings installer script installs various device types using a common smartthings libary. By doing this the installer script sets up a refresh loop to refresh all devices that it installed.

Device Template

NOTE: It is suggested to keep a 'getValue' function as shown in the sample device blow, But it is not required.

```
from core.models.device import Device
import logging

class Device(Device):
  def __init__(self, deviceID, args={}):
    self.METADATA = {
      'title' : 'Plugin Title',
      'type' : 'Plugin Type',
      'package' : 'FOLDER NAME OF PLUGIN',
      'module' : 'FILE NAME OF PLUGIN'
    }

    self.COMMANDS = {
      'command' : self.function
    }

    self.REQUESTS = {
      'request' : self.function
    }
    args = args.get('args')
    name = args.get('name')
    super(Device,self).__init__(deviceID, name)

  # Your Code Goes Here
```

Sample Device

```
from core.models.device import Device
import logging

class Device(Device):

def __init__(self, deviceID, args={}):
  self.METADATA = {
    'title' : 'Firefly Presence Device',
    'type' : 'presence',
    'package' : 'ffPresence',
    'module' : 'ffPresence'
  }

  self.COMMANDS = {
    'presence' : self.setPresence
  }

  self.REQUESTS = {
    'presence' : self.getPresence
  }
  args = args.get('args')
  name = args.get('name')
  super(Device,self).__init__(deviceID, name)

  self._notify_present = args.get('notify_present')
  self.notify_not_present = args.get('notify_not_present')
  self.notify_device = args.get('notify_device')
  self._presence = True

def setPresence(self, value):
  from core.firefly_api import event_message
  if value is not self.presence:
    self.presence = value
    event_message(self._name, "Setting Presence To " + str(value))
    logging.debug("Setting Presence To " + str(value))


def getPresence(self):
  return self.presence

@property
def presence(self):
    return self._presence

@presence.setter
def presence(self, value):
  self._presence = value
```

Device With Children

Part 1: Installer

Part 2: Background High Speed Refresh

Most devices will not need something like this. This was an example of a high speed refresher polling form an external API every two seconds for changes.

```
```

App Plugin

CHAPTER 3

Location

The location module is used for:

- Location Based Event Scheduling
  - Dawn Scheduling
  - Sunrise Scheduling
  - Solar Noon Scheduling
  - Sunset Scheduling
  - Dusk Scheduling
- Mode Handling
- Current Time
- Current Sun Status
  - isDark
  - isLight

Location Scheduler

Mode Handling

Time

Sun Status

CHAPTER 4

Scheduler

There are two ways to use the scheduler:

1. The Global Scheduler: Accesable from all modules - This is useful if you want one module to schedule and event and another module to unschdule the task
2. Per Instance Scheduler: A scheduler object just used in one module.

Global Scheduler

Instance Scheduler

Scheduler Syntax

CHAPTER 5

Config

Pushover Config

This goes into a file named pushover.json

```
{
  "Zach Pushover": {
    "api_key":"<<API KEY>>",
    "user_key":"<<USER KEY>>",
    "default_message":"This is a test message from firefly",
    "default_priority":"high",
    "presence_device":"Zach Presence"
  },
    "Ariel Pushover": {
    "api_key":"<<API KEY>>",
    "user_key":"<<USER KEY>>",
    "default_message":"This is a test message from firefly",
    "default_priority":"high",
    "presence_device":"Ariel Presence"
  }
}
```

SmartThings Config

This goes into a file name ZPSmartThings_settings.py

```
settings = {'access_token':'<TOKEN>', 'url':'<URL>'}
```

Make a SmartApp in the SmartThings IDE

```
//
// Definition
```

```
//
definition(
    name: "Firefly Access",
    namespace: "zpriddy",
    author: "zpriddy",
    description: "API access for Alexa.",
    category: "Convenience",
    iconUrl: "https://s3.amazonaws.com/smartapp-icons/Convenience/Cat-Convenience.png
↪",
    iconX2Url: "https://s3.amazonaws.com/smartapp-icons/Convenience/Cat-
↪Convenience@2x.png",
    oauth: true) {
}


//
// Preferences
//
preferences {
    section("Allow access to the following things...") {
        input "contacts", "capability.contactSensor", title: "Which contact sensors?",
↪ multiple: true, required: false
        input "locks", "capability.lock", title: "Which Locks?", multiple: true,␣
↪required: false
        input "meters", "capability.powerMeter", title: "Which meters?", multiple:␣
↪true, required: false
        input "motions", "capability.motionSensor", title: "Which motion sensors?",␣
↪multiple: true, required: false
        input "presences", "capability.presenceSensor", title: "Which presence␣
↪sensors?", multiple: true, required: false
        input "dimmers", "capability.switchLevel", title: "Which dimmers?", multiple:␣
↪true, required: false
        input "switches", "capability.switch", title: "Which switches?", multiple:␣
↪true, required: false
        input "temperatures", "capability.temperatureMeasurement", title: "Which␣
↪temperature sensors?", multiple: true, required: false
        input "humidities", "capability.relativeHumidityMeasurement", title: "Which␣
↪humidity sensors?", multiple: true, required: false
        input "colors", "capability.colorControl", title: "Which Color Lights?",␣
↪multiple: true, required: false


    }
}


//
// Mappings
//
mappings {
    path("/config") {
        action: [
            GET: "getConfig",
            POST: "postConfig"
        ]
    }
    path("/contact") {
        action: [
            GET: "getContact"
        ]
    }
    path("/color") {
        action: [
            GET: "getColor",
            POST: "postColor"
        ]
    }
    path("/devices") {
        action: [
            GET: "getDevices"
        ]
    }
    path("/lock") {
        action: [
            GET: "getLock",
            POST: "postLock"
        ]
    }
    path("/mode") {
        action: [
            GET: "getMode",
            POST: "postMode"
        ]
    }
    path("/motion") {
        action: [
            GET: "getMotion"
        ]
    }
    path("/phrase") {
        action: [
            GET: "getPhrase",
            POST: "postPhrase"
        ]
    }
    path("/power") {
        action: [
            GET: "getPower"
        ]
    }
    path("/presence") {
        action: [
            GET: "getPresence"
        ]
    }
    path("/dimmer") {
        action: [
            GET: "getDimmer",
            POST: "postDimmer"
        ]
    }
    path("/dimmerLevel") {
        action: [
            POST: "dimmerLevel"
        ]
    }
    path("/switch") {
        action: [
            GET: "getSwitch",
            POST: "postSwitch"
        ]
    }
    path("/temperature") {
        action: [
            GET: "getTemperature"
        ]
    }
    path("/humidity") {
        action: [
            GET: "getHumidity"
        ]
    }
    path("/weather") {
        action: [
            GET: "getWeather"
        ]
    }
}


//
// Installation
//
def installed() {
    initialize()
}

def updated() {
    unsubscribe()
    initialize()
}

def initialize() {
    state.dashingURI = ""
    state.dashingAuthToken = ""
    state.widgets = [
        "contact": [:],
        "lock": [:],
        "mode": [:],
        "motion": [:],
        "power": [:],
        "presence": [:],
        "dimmer": [:],
        "switch": [:],
        "temperature": [:],
        "humidity": [:],
        "color" : [:],

        ]

    subscribe(contacts, "contact", contactHandler)
    subscribe(location, "mode", locationHandler)
    subscribe(locks, "lock", lockHandler)
    subscribe(motions, "motion", motionHandler)
    subscribe(meters, "power", meterHandler)
    subscribe(presences, "presence", presenceHandler)
    subscribe(dimmers, "switch", dimmerSwitch)
    subscribe(dimmers, "level", dimmerHandler)
    subscribe(switches, "switch", switchHandler)
    subscribe(temperatures, "temperature", temperatureHandler)
    subscribe(humidities, "humidity", humidityHandler)
    subscribe(colors, "color", colorHandler)

}


//
// Config
//
def getConfig() {
    ["dashingURI": state.dashingURI, "dashingAuthToken": state.dashingAuthToken]
}

def postConfig() {
    state.dashingURI = request.JSON?.dashingURI
    state.dashingAuthToken = request.JSON?.dashingAuthToken
    respondWithSuccess()
}

//
// Contacts
//
def getContact() {
    def deviceId = request.JSON?.deviceId
    log.debug "getContact ${deviceId}"

    if (deviceId) {
        registerWidget("contact", deviceId, request.JSON?.widgetId)

        def whichContact = contacts.find { it.displayName == deviceId }
        if (!whichContact) {
            return respondWithStatus(404, "Device '${deviceId}' not found.")
        } else {
            return [
                "deviceId": deviceId,
                "deviceType":whichContact.name,
                "state": whichContact.currentContact]
        }
    }

    def result = [:]
    contacts.each {
        result[it.displayName] = [
            "state": it.currentContact,
            "deviceType":it.name,
            "widgetId": state.widgets.contact[it.displayName]]}

    return result
}

def contactHandler(evt) {
    def widgetId = state.widgets.contact[evt.displayName]
    notifyWidget(widgetId, ["state": evt.value])
}

//
// Locks
//
def getLock() {
    def deviceId = request.JSON?.deviceId
    log.debug "getLock ${deviceId}"

    if (deviceId) {
        registerWidget("lock", deviceId, request.JSON?.widgetId)

        def whichLock = locks.find { it.displayName == deviceId }
        if (!whichLock) {
            return respondWithStatus(404, "Device '${deviceId}' not found.")
        } else {
            return [
                "deviceId": deviceId,
                "deviceType":whichLock.name,
                "state": whichLock.currentLock]
        }
    }

    def result = [:]
    locks.each {
        result[it.displayName] = [
            "state": it.currentLock,
            "deviceType":it.name,
            "widgetId": state.widgets.lock[it.displayName]]}

    return result
}

def postLock() {
    def command = request.JSON?.command
    def deviceId = request.JSON?.deviceId
    log.debug "postLock ${deviceId}, ${command}"

    if (command && deviceId) {
        def whichLock = locks.find { it.displayName == deviceId }
        if (!whichLock) {
            return respondWithStatus(404, "Device '${deviceId}' not found.")
        } else {
            whichLock."$command"()
        }
    }
    return respondWithSuccess()
}

def lockHandler(evt) {
    def widgetId = state.widgets.lock[evt.displayName]
    notifyWidget(widgetId, ["state": evt.value])
}

//
// Meters
//
```

```
def getPower() {
    def deviceId = request.JSON?.deviceId
    log.debug "getPower ${deviceId}"

    if (deviceId) {
        registerWidget("power", deviceId, request.JSON?.widgetId)

        def whichMeter = meters.find { it.displayName == deviceId }
        if (!whichMeter) {
            return respondWithStatus(404, "Device '${deviceId}' not found.")
        } else {
            return [
                "deviceId": deviceId,
                "value": whichMeter.currentValue("power"),
                "deviceType":whichMeter.name,
                "energy":whichMeter.currentValue("energy")]
        }
    }

    def result = [:]
    meters.each {
        it.poll()
        result[it.displayName] = [
            "value": it.currentValue("power"),
            "energy": it.currentValue("energy"),
            "deviceType":it.name,
            "widgetId": state.widgets.power[it.displayName]]}

    return result
}

def meterHandler(evt) {
    def widgetId = state.widgets.power[evt.displayName]
    notifyWidget(widgetId, ["value": evt.value])
}

//
// Modes
//
def getMode() {
    def result = [:]
    def modeId = request.JSON?.modeId
    def widgetId = request.JSON?.widgetId
    if (widgetId) {
        if (!state['widgets']['mode'].contains(widgetId)) {
            state['widgets']['mode'].add(widgetId)
            log.debug "registerWidget for mode: ${widgetId}"
        }
    }

    if(modeId)
    {
        def whichMode = modes.find { it.displayName == modeId }
        if (!whichMode) {
            return respondWithStatus(404, "Device '${modeId}' not found.")
        } else {
            return [
                "modeId": modeId ]
        }
    }


    location.modes.each {
        result[it.name] = [
            "name": it.name]
            }



    return result

}

def postMode() {
    def mode = request.JSON?.mode
    log.debug "postMode ${mode}"

    if (mode) {
        setLocationMode(mode)
    }

    if (location.mode != mode) {
        return respondWithStatus(404, "Mode not found.")
    }
    return respondWithSuccess()
}

def locationHandler(evt) {
    for (i in state['widgets']['mode']) {
        notifyWidget(i, ["mode": evt.value])
    }
}

//
// Motions
//
def getMotion() {
    def deviceId = request.JSON?.deviceId
    log.debug "getMotion ${deviceId}"

    if (deviceId) {
        registerWidget("motion", deviceId, request.JSON?.widgetId)

        def whichMotion = motions.find { it.displayName == deviceId }
        if (!whichMotion) {
            return respondWithStatus(404, "Device '${deviceId}' not found.")
        } else {
            return [
                "deviceId": deviceId,
                "deviceType":whichMotion.name,
                "state": whichMotion.currentValue("motion")]
        }
    }

    def result = [:]
    motions.each {
        result[it.displayName] = [
            "state": it.currentValue("motion"),
            "deviceType":it.name,
            "widgetId": state.widgets.motion[it.displayName]]}



    return result
}

def motionHandler(evt) {
    def widgetId = state.widgets.motion[evt.displayName]
    notifyWidget(widgetId, ["state": evt.value])
}

//
// Phrases
//
def getPhrase() {
    def result = [:]
//    def phraseId = request.JSON?.phraseId

/*
    location.helloHome.each {
        result[it.name] = [
            "name": it.name]
            }
*/
    result = location.helloHome?.getPhrases()*.label
    log.debug result

    log.debug "GETING PHRSES"
    return result

}

def postPhrase() {
    def phrase = request.JSON?.phrase
    log.debug "postPhrase ${phrase}"

    if (!phrase) {
        respondWithStatus(404, "Phrase not specified.")
    }

    location.helloHome.execute(phrase)

    return respondWithSuccess()

}

//
// Presences
//
def getPresence() {
    def deviceId = request.JSON?.deviceId
    log.debug "getPresence ${deviceId}"

    if (deviceId) {
        registerWidget("presence", deviceId, request.JSON?.widgetId)

        def whichPresence = presences.find { it.displayName == deviceId }
        if (!whichPresence) {
            return respondWithStatus(404, "Device '${deviceId}' not found.")
        } else {
            return [
                "deviceId": deviceId,
                "deviceType":whichPresence.name,
                "state": whichPresence.currentPresence]
        }
    }

    def result = [:]
    presences.each {
        result[it.displayName] = [
            "state": it.currentPresence,
            "deviceType":it.name,
            "widgetId": state.widgets.presence[it.displayName]]}

    return result
}

def presenceHandler(evt) {
    def widgetId = state.widgets.presence[evt.displayName]
    notifyWidget(widgetId, ["state": evt.value])
}

//
// Dimmers
//
def getDimmer() {
    def deviceId = request.JSON?.deviceId
    log.debug "getDimmer ${deviceId}"

    if (deviceId) {
        registerWidget("dimmer", deviceId, request.JSON?.widgetId)

        def whichDimmer = dimmers.find { it.displayName == deviceId }
        if (!whichDimmer) {
            return respondWithStatus(404, "Device '${deviceId}' not found.")
        } else {
            return [
                "deviceId": deviceId,
                "level": whichDimmer.currentValue("level"),
                "deviceType":whichDimmer.name,
                "state": whichDimmer.currentValue("switch")
            ]
        }
    }

    def result = [:]
    dimmers.each {
        result[it.displayName] = [
            "state": it.currentValue("switch"),
            "level": it.currentValue("level"),
            "deviceType":it.name,
            "widgetId": state.widgets.dimmer[it.displayName]]}
    return result
}

def postDimmer() {
    def command = request.JSON?.command
    def deviceId = request.JSON?.deviceId
    log.debug "postDimmer ${deviceId}, ${command}"

    if (command && deviceId) {
        def whichDimmer = dimmers.find { it.displayName == deviceId }
        if (!whichDimmer) {
            return respondWithStatus(404, "Device '${deviceId}' not found.")
        } else {
            whichDimmer."$command"()
        }
    }
    return respondWithSuccess()
}

def dimmerLevel() {
    def command = request.JSON?.command
    def deviceId = request.JSON?.deviceId
    log.debug "dimmerLevel ${deviceId}, ${command}"
    command = command.toInteger()
    if (command && deviceId) {
        def whichDimmer = dimmers.find { it.displayName == deviceId }
        if (!whichDimmer) {
            return respondWithStatus(404, "Device '${deviceId}' not found.")
        } else {
            whichDimmer.setLevel(command)
        }
    }
    return respondWithSuccess()
}

def dimmerHandler(evt) {
    def widgetId = state.widgets.dimmer[evt.displayName]
    pause(1000)
    notifyWidget(widgetId, ["level": evt.value])
}

def dimmerSwitch(evt) {
    def whichDimmer = dimmers.find { it.displayName == evt.displayName }
    def widgetId = state.widgets.dimmer[evt.displayName]
    notifyWidget(widgetId, ["state": evt.value])
}

//
// Switches
//
def getSwitch() {
    def deviceId = request.JSON?.deviceId
    log.debug requestJSON
    log.debug "getSwitch ${deviceId}"

    if (deviceId) {
        registerWidget("switch", deviceId, request.JSON?.widgetId)
        def whichSwitch = switches.find { it.displayName == deviceId }
        if (!whichSwitch) {
            return respondWithStatus(404, "Device '${deviceId}' not found.")
        } else {
            return [
                "deviceId": deviceId,
                "deviceType":whichSwitch.name,
                "switch": whichSwitch.currentSwitch]
        }
    }

    def result = [:]
    switches.each {
        result[it.displayName] = [
            "state": it.currentSwitch,
            "deviceType":it.name,
            "widgetId": state.widgets.switch[it.displayName]]}

    return result
}

def postSwitch() {
    def command = request.JSON?.command
    def deviceId = request.JSON?.deviceId
    log.debug "postSwitch ${deviceId}, ${command}"

    if (command && deviceId) {
        def whichSwitch = switches.find { it.displayName == deviceId }
        if (!whichSwitch) {
            return respondWithStatus(404, "Device '${deviceId}' not found.")
        } else {
            whichSwitch."$command"()
        }
    }
    return respondWithSuccess()
}

def switchHandler(evt) {
    def widgetId = state.widgets.switch[evt.displayName]
    notifyWidget(widgetId, ["state": evt.value])
}

//
// Colors
//
def getColor() {
    def deviceId = request.JSON?.deviceId
    log.debug requestJSON
    log.debug "getColor ${deviceId}"

    if (deviceId) {
        registerWidget("color", deviceId, request.JSON?.widgetId)

        def whichColor = colors.find { it.displayName == deviceId }
        if (!whichColor) {
            return respondWithStatus(404, "Device '${deviceId}' not found.")
        } else {
            return [
                "deviceId": deviceId,
                "deviceType":whichSwitch.name,
                "color": whichColor.currentColor]
        }
    }

    def result = [:]
    colors.each {
        result[it.displayName] = [
            "hue": it.currentValue("hue"),
            "sat": it.currentValue("saturation"),
            "deviceType":it.name,
            "widgetId": state.widgets.color[it.displayName]]}

    return result
}

def postColor() {
    def hex = request.JSON?.hex
    def hue = request.JSON?.hue
    def sat = request.JSON?.sat
    def level = request.JSON?.level
    def deviceId = request.JSON?.deviceId
    log.debug "postColor ${deviceId}, ${hex}, ${hue}, ${sat}"
    def value = [:]
    value.hex = hex
    value.saturation = (sat as Integer)/255*100
    value.hue = (hue as Integer) /360*100
    value.level = null

    log.debug value

    if (command && deviceId) {
        def whichColor = colors.find { it.displayName == deviceId }
        if (!whichColor) {
            return respondWithStatus(404, "Device '${deviceId}' not found.")
        } else {
            whichColor.setColor(value)
        }
    }
    return respondWithSuccess()
}

def colorHandler(evt) {
    def widgetId = state.widgets.color[evt.displayName]
    notifyWidget(widgetId, ["color": evt.value])
}

//
// Temperatures
//
def getTemperature() {
    def deviceId = request.JSON?.deviceId
    log.debug "getTemperature ${deviceId}"

    if (deviceId) {
        registerWidget("temperature", deviceId, request.JSON?.widgetId)
        def whichTemperature = temperatures.find { it.displayName == deviceId }
        if (!whichTemperature) {
            return respondWithStatus(404, "Device '${deviceId}' not found.")
        } else {
            return [
                "deviceId": deviceId,
                "deviceType":whichTemperature.name,
                "value": whichTemperature.currentTemperature]
        }
    }

    def result = [:]
    temperatures.each {
        result[it.displayName] = [
            "value": it.currentTemperature,
            "deviceType":it.name,
            "widgetId": state.widgets.temperature[it.displayName]]}

    return result
}

def temperatureHandler(evt) {
    def widgetId = state.widgets.temperature[evt.displayName]
    notifyWidget(widgetId, ["value": evt.value])
}

//
// Humidities
//
def getHumidity() {
    def deviceId = request.JSON?.deviceId
    log.debug "getHumidity ${deviceId}"

    if (deviceId) {
        registerWidget("humidity", deviceId, request.JSON?.widgetId)

        def whichHumidity = humidities.find { it.displayName == deviceId }
        if (!whichHumidity) {
            return respondWithStatus(404, "Device '${deviceId}' not found.")
        } else {
            return [
                "deviceId": deviceId,
                "deviceType":ehichHumidity.name,
                "value": whichHumidity.currentHumidity]
        }
    }

    def result = [:]
    humidities.each {
        result[it.displayName] = [
            "value": it.currentHumidity,
            "deviceType":it.name,
            "widgetId": state.widgets.humidity[it.displayName]]}

    return result
}
```

```
def humidityHandler(evt) {
    def widgetId = state.widgets.humidity[evt.displayName]
    notifyWidget(widgetId, ["value": evt.value])
}

//
// Weather
//
def getWeather() {
    def feature = request.JSON?.feature
    if (!feature) {
        feature = "conditions"
    }
    return getWeatherFeature(feature)
}

//
// Get All Devices
//
def getDevices() {
    def rDimmers = [:]
    dimmers.each {
        rDimmers[it.displayName] = [
            "state": it.currentValue("switch"),
            "level": it.currentValue("level"),
            "deviceType":it.name]}

  def rSwitches = [:]
    switches.each {
        rSwitches[it.displayName] = [
            "state": it.currentSwitch,
            "deviceType":it.name]}

  def rMotion = [:]
    motions.each {
        rMotion[it.displayName] = [
            "state": it.currentValue("motion"),
            "deviceType":it.name]}

  def rContacts = [:]
    contacts.each {
        rContacts[it.displayName] = [
            "state": it.currentContact,
            "deviceType":it.name]}

    def result = [:]
    result['contacts'] = rContacts
    result['motion'] = rMotion
    result['dimmers'] = rDimmers
    result['switches'] = rSwitches

  return result
}


//
// Widget Helpers
//
```

```
private registerWidget(deviceType, deviceId, widgetId) {
    if (deviceType && deviceId && widgetId) {
        state['widgets'][deviceType][deviceId] = widgetId
        log.debug "registerWidget ${deviceType}:${deviceId}@${widgetId}"
    }
}

private notifyWidget(widgetId, data) {
    if (widgetId && state.dashingAuthToken) {
        def uri = getWidgetURI(widgetId)
        data["auth_token"] = state.dashingAuthToken
        log.debug "notifyWidget ${uri} ${data}"
        httpPostJson(uri, data)
    }
}

private getWidgetURI(widgetId) {
    state.dashingURI + "/widgets/${widgetId}"
}


//
// Response Helpers
//
private respondWithStatus(status, details = null) {
    def response = ["error": status as Integer]
    if (details) {
        response["details"] = details as String
    }
    return response
}

private respondWithSuccess() {
    return respondWithStatus(0)
}
```