HomeSeer Plug-in: Hubitat Elevation

HS3: Hubitat

HS4: mcsHubitat

HS3-3.4.4.0 / HS4-4.4.4.0
June 3, 2021

Table of Contents

Table of Figures

1 Introduction

This plug-in mirrors your Hubitat Elevation devices to your HomeSeer home automation server. This capability allows you to include any of your Hubitat devices into HomeSeer events for status or control operations. Bring new ZigBee, Z-Wave or other devices from Hubitat that HomeSeer does not support.

Hubitat Documentation

1.1 Features

- Z-Wave Devices
- ZigBee Devices
- Lutron Radio RA2 / Caseta Pro
- Philips Hue bridge
- LiFX
- TP-Link SmartPlugs (Limited support)
- More devices and app support coming…
  - Hubitat Community on GitHub

1.2 Installation

Install from the Plug-in Manager:

Plug-Ins -> Manage -> Additional Interfaces -> Lighting & Primary Technology

The HS4 plug-in is called mcsHubitat. The HS3 plug-in is called Hubitat. Each plug-in maintains its information in different locations, but both point to the same set to devices. For HS3 users, only Hubitat is available. The HS4 user has a choice. Hubitat complies with HS3 API that will run under HS3 or HS4. mcsHubitat complies with HS4 API and will only work with HS4.

If a HS3 user of Hubitat plug-in desires to migrate their devices and events setup in HS3 to HS4 using mcsHubitat then they need to physically copy two files. The HS3 file \Config\hubitat.ini to \Config\mcsHubitat.ini. This file contains the URL to connect to Hubitat Elevation and the other settings that are contained in the setup page of Hubitat plug-in. The relationship between Hubitat device IDs and HomeSeer Devices is in the HS3 file\Config\ habitat-devicemap.json for HS3. For HS4 the file is in the same location with name mcsHabitat-devicemap.json. If the intent is keep the HS3 setup and create new devices with HS4 plug-in then no action should be taken with these files. If the intent is to preserve the device as events based upon these devices from

HS3 that were imported into HS4 then habitat-devicemap.json should be copied to mcsHabitat-devicemap.json. Once mcsHubitat starts and data exists in mcsHubitat-devicemap.json then it will be a one-way migration to mcsHubitat. Recovery from a change-of-mind will only be by restoring from a backup the HS4 configuration prior to running mcsHubitat.

1.3 Configuration

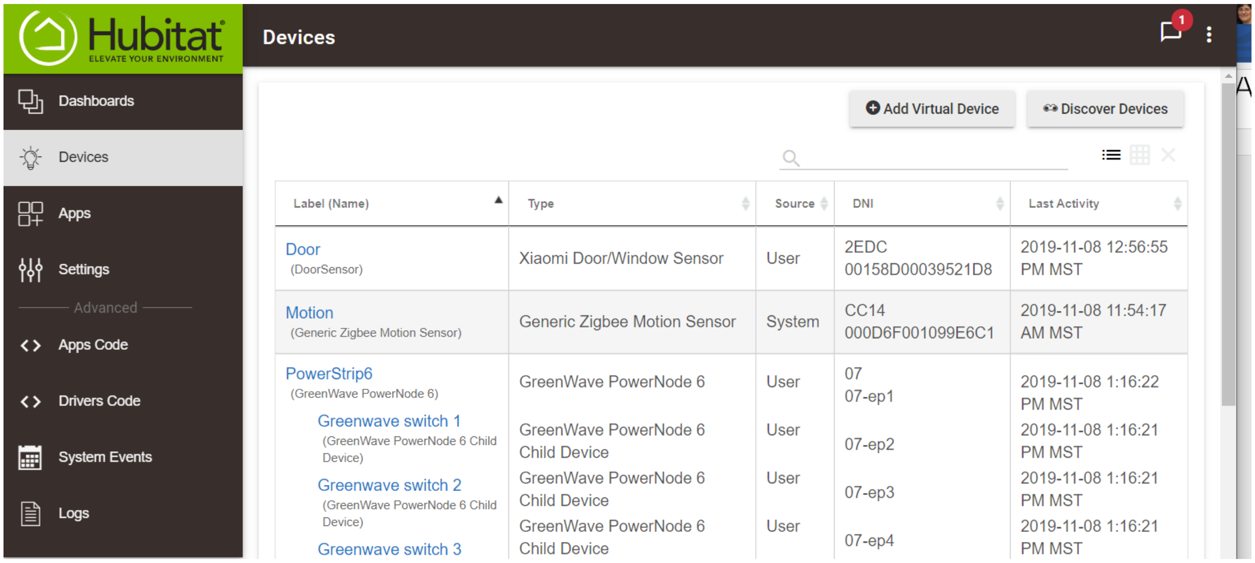
1.3.1 Hubitat

1.3.1.1 Initial Setup

- Pre-Requisites: Hubitat Hub setup with devices Hubitat Maker API app installed
- Go to Apps in Hubitat
- Add Built-In App
- Click Maker API in the list
  - Security - Local IP Address must be on. Cloud access is up to you
  - Select Devices (Click to Set)
  - Select the device(s) you want to be available through Maker API
  - Endpoints - Local URLS
  - Click on "**Get All Devices with Full Details**" link. Then on the new web page that Maker API opens, copy the URL of that web page. Will be something like: http://192.168.0.100/apps/api/2/devices/all?access_token=6f8xxx03-2bfe-4cbb-9611-ef303xxxf34a. Save this to enter into the plug-in configuration page.
  - You can come back to the Maker API app to get this link if you miss this step
  - Click Done

1.3.1.2 Hubitat New Device Addition

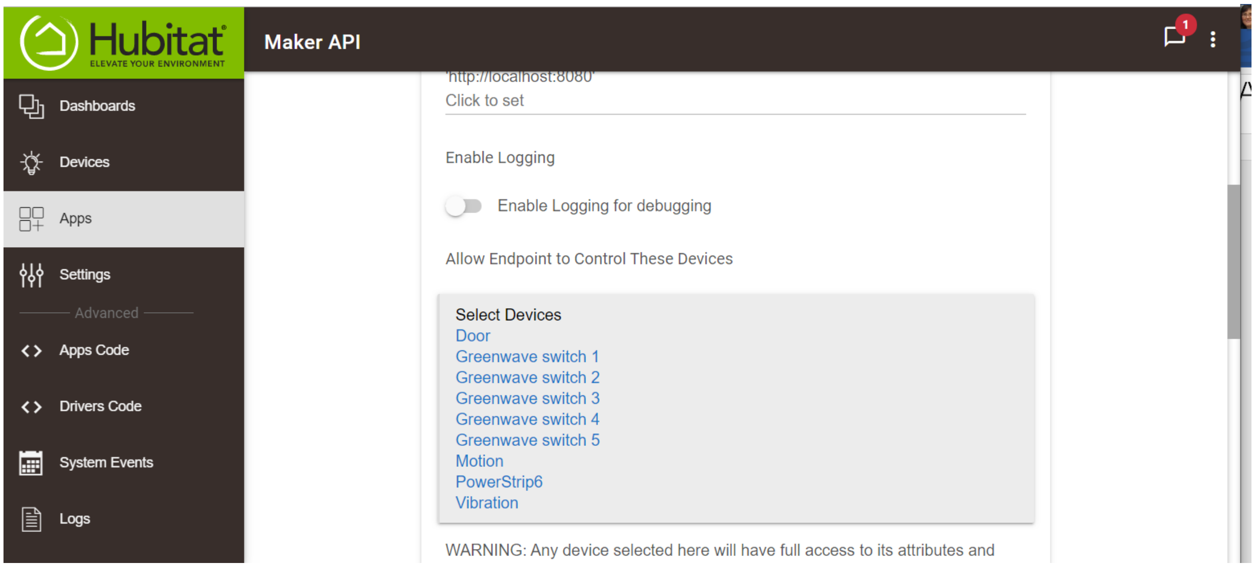
When new devices are added to Hubitat which is normally does through the Hubitat Devices page using the "Discover Devices" button in the top right. Additional prompts will then be provided to do a discovery or inclusion process. Newly discovered devices will appear on the same page. The new device should be given a name that will be the same name used in HS later.



Label (Name)	Type	Source	DNI	Last Activity
Door (DoorSensor)	Xiaomi Door/Window Sensor	User	2EDC 00158D00039521D8	2019-11-08 12:56:55 PM MST
Motion (Generic Zigbee Motion Sensor)	Generic Zigbee Motion Sensor	System	CC14 000D6F001099E6C1	2019-11-08 11:54:17 AM MST
PowerStrip6 (GreenWave PowerNode 6)	GreenWave PowerNode 6	User	07 07-ep1	2019-11-08 1:16:22 PM MST
Greenwave switch 1 (GreenWave PowerNode 6 Child Device)	GreenWave PowerNode 6 Child Device	User	07-ep2	2019-11-08 1:16:21 PM MST
Greenwave switch 2 (GreenWave PowerNode 6 Child Device)	GreenWave PowerNode 6 Child Device	User	07-ep3	2019-11-08 1:16:21 PM MST
Greenwave switch 3	GreenWave PowerNode 6 Child Device	User	07-ep4	2019-11-08 1:16:21 PM MST

Figure 1 Hubitat Devices Page

MakerAPI needs to be made aware of the new device. Go to the Apps -> MakerAPI page and scroll down to “Select Devices”. The newly discovered device will not be in this list. Click the “Select Devices” panel and a checkbox will be visible for the new device and it is used to make it visible to plug-in.



"http://localhost:8080"
Click to set

Enable Logging

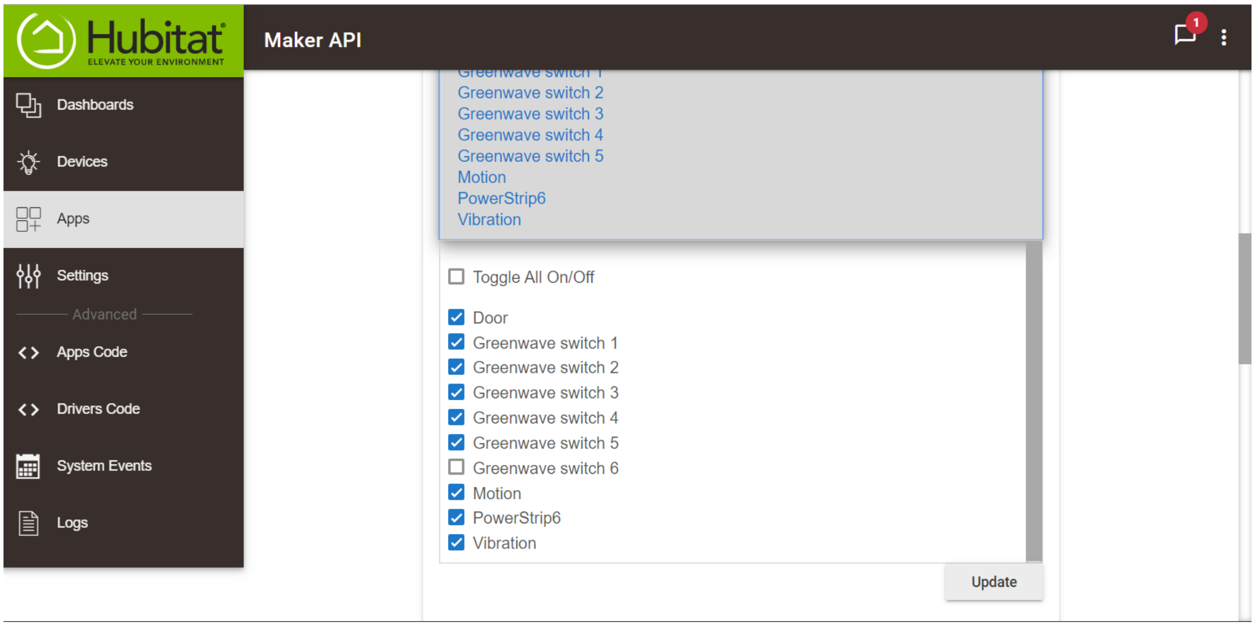
Enable Logging for debugging

Allow Endpoint to Control These Devices

Select Devices
Door
Greenwave switch 1
Greenwave switch 2
Greenwave switch 3
Greenwave switch 4
Greenwave switch 5
Motion
PowerStrip6
Vibration

WARNING: Any device selected here will have full access to its attributes and

Figure 2 Hubitat Maker API Page



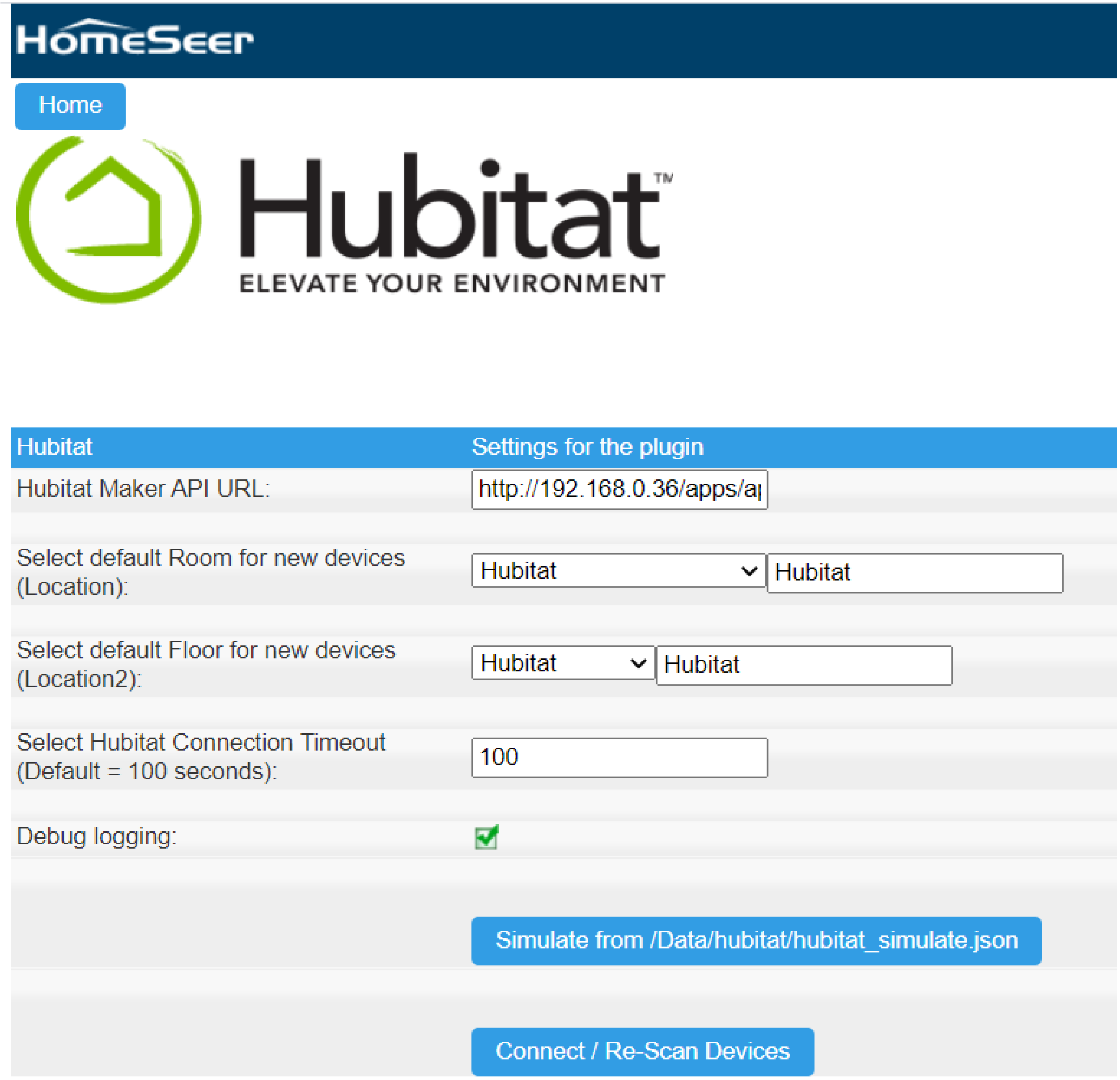
Greenwave switch 1
Greenwave switch 2
Greenwave switch 3
Greenwave switch 4
Greenwave switch 5
Motion
PowerStrip6
Vibration

- □ Toggle All On/Off
- ☑ Door
- ☑ Greenwave switch 1
- ☑ Greenwave switch 2
- ☑ Greenwave switch 3
- ☑ Greenwave switch 4
- ☑ Greenwave switch 5
- □ Greenwave switch 6
- ☑ Motion
- ☑ PowerStrip6
- ☑ Vibration

Figure 3 Hubitat Maker API Device Selection

1.3.2 Plug-in Setup

Navigate to the Hubitat Config page from HS and something similar to view below should be visible.

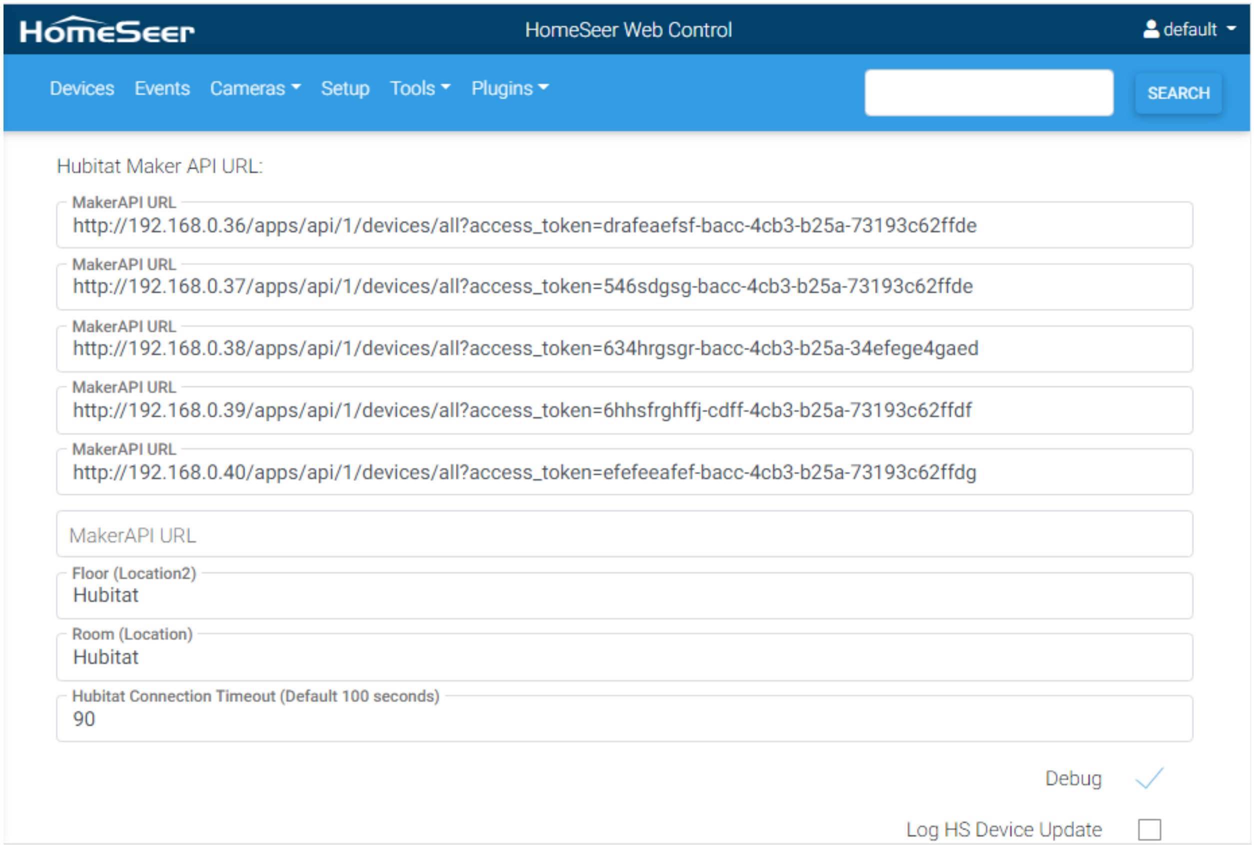


HomeSeer
Home
Hubitat™
ELEVATE YOUR ENVIRONMENT

Hubitat	Settings for the plugin
Hubitat Maker API URL:	http://192.168.0.36/apps/a
Select default Room for new devices (Location):	Hubitat ▾ Hubitat
Select default Floor for new devices (Location2):	Hubitat ▾ Hubitat
Select Hubitat Connection Timeout (Default = 100 seconds):	100
Debug logging:	☑

Simulate from /Data/hubitat/hubitat_simulate.json
Connect / Re-Scan Devices

Figure 4 HS3 Hubitat Plug-in Settings Page



Hubitat Maker API URL:

MakerAPI URL
http://192.168.0.36/apps/api/1/devices/all?access_token=drafeaefsf-bacc-4cb3-b25a-73193c62ffde

MakerAPI URL
http://192.168.0.37/apps/api/1/devices/all?access_token=546sdgsg-bacc-4cb3-b25a-73193c62ffde

MakerAPI URL
http://192.168.0.38/apps/api/1/devices/all?access_token=634hrgsgr-bacc-4cb3-b25a-34efege4gaed

MakerAPI URL
http://192.168.0.39/apps/api/1/devices/all?access_token=6hhsfrghffj-cdff-4cb3-b25a-73193c62ffdf

MakerAPI URL
http://192.168.0.40/apps/api/1/devices/all?access_token=efefeeafef-bacc-4cb3-b25a-73193c62ffdg

MakerAPI URL

Floor (Location2)
Hubitat

Room (Location)
Hubitat

Hubitat Connection Timeout (Default 100 seconds)
90

Debug ✓

Log HS Device Update □

Figure 5 HS4 Hubitat Plug-in Settings Page

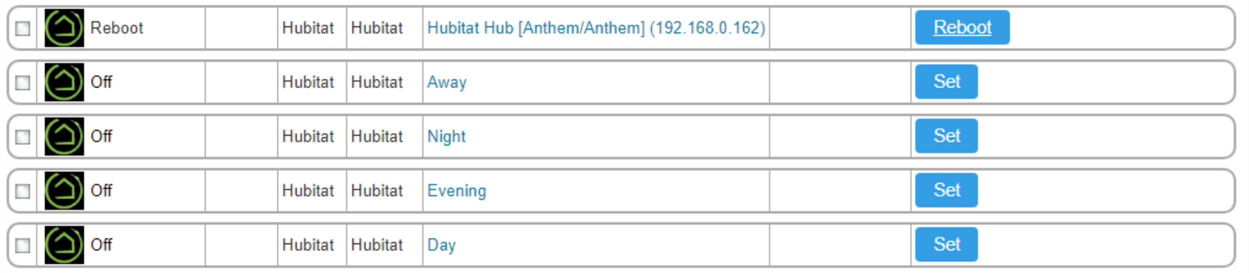
The only required setup parameter is for the row "Hubitat Maker API URL". Into this text box use the copied link from the bolded line in Section 1.3.1. With HS4, provisions exist to manage up to six separate Hubitat Elevation units with each requiring the Maker API URL.

The HS Room and Floor (Location & Location2) device properties will be set to "Hubitat" unless a different default is selected in the next two setup rows. In your HomeSeer Device page you will need to use the drop downs to make sure the room/floor/Device Types are checked to make the devices visible. If you suspect any problems go back to the plug-in setup and enable debug logging.

The connection timeout is the number of seconds to wait for Hubitat Elevation to respond to any request from the plug-in. The default of 100 seconds will work fine for most installations. For a Hubitat Elevation that is well loaded then this value should be increased to avoid log errors and to have successful communications.

Two log options are available. One is to produce debug information into a file hubitat_debug.txt created in the plugin's subdirectory of the \Data HS folder. The second is to use the HS log to record updates to HS device in the HS log from Hubitat Elevation. The update will look like "HS Feature 3311(MotionSensor) updated with value 0"

The controls associated with the Hubitat Elevation hub are located with the Hubitat feature in HS4 and split between the Settings page and the root Hubitat device in HS3. In HS4 a mouseover of the gear control will show the specific button label.



□	Reboot		Hubitat	Hubitat	Hubitat Hub [Anthem/Anthem] (192.168.0.162)		Reboot
□	Off		Hubitat	Hubitat	Away		Set
□	Off		Hubitat	Hubitat	Night		Set
□	Off		Hubitat	Hubitat	Evening		Set
□	Off		Hubitat	Hubitat	Day		Set

Figure 6 HS3 Hubitat Device Controls

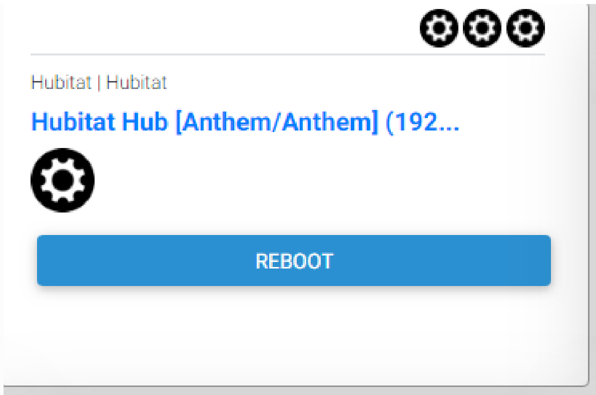


Figure 7 HS4 Hubitat Feature Controls

During startup the plug-in will read any previous setup information and then spawn a thread to make contact with each of the instances of the Hubitat Elevation Hub that have been setup. During this time until while making initial contact with each hub any commands being sent to the plug-in for delivery to the hub will be queued and then sent when the hub is ready. Status of progress is provided in the log and in the developers' console.

After initialization is complete any command delivered by HS will be directly forwarded to Hubitat and the plug-in will wait for a response. For the HS4 plug-in control will be returned to HS immediately after the request. For the HS3 plug-in control will be returned to HS after the Hubitat response is received.

Each time the plug-in starts it will query Hubitat MakerAPI to get the set of Hubitat devices that have been exposed. This query can also be done using the “Connect Re-Scan Devices” button in HS3 or Sync button in HS4.

A log of bidirectional communication between HS and Hubitat can be collected in the file “habitat_debug.txt”. This file is located in the \Data\hubitat for HS3 or \Data\mcsHubitat for HS4 subfolder of the HS install. In some cases, the communications contain the IP address and the Maker Key which are personal. These have been redacted in the log to enable sharing of this log. If additional information is considered personal in the log then it should be changed in a manner that does not compromise the integrity of the information in the log.

Selected lines from “habitat_debug.txt” can be pasted into the file “hubitat_simulate.json” that is in the same folder as “habitat_debug.txt”. When the “Simulate from …” button is used it will send those lines to the plug-in just as if they were sent from Hubitat Elevation. This feature is primarily a support feature used by the plug-in author to reproduce communications for devices that the author does not have and the plug-in is not behaving as expected.

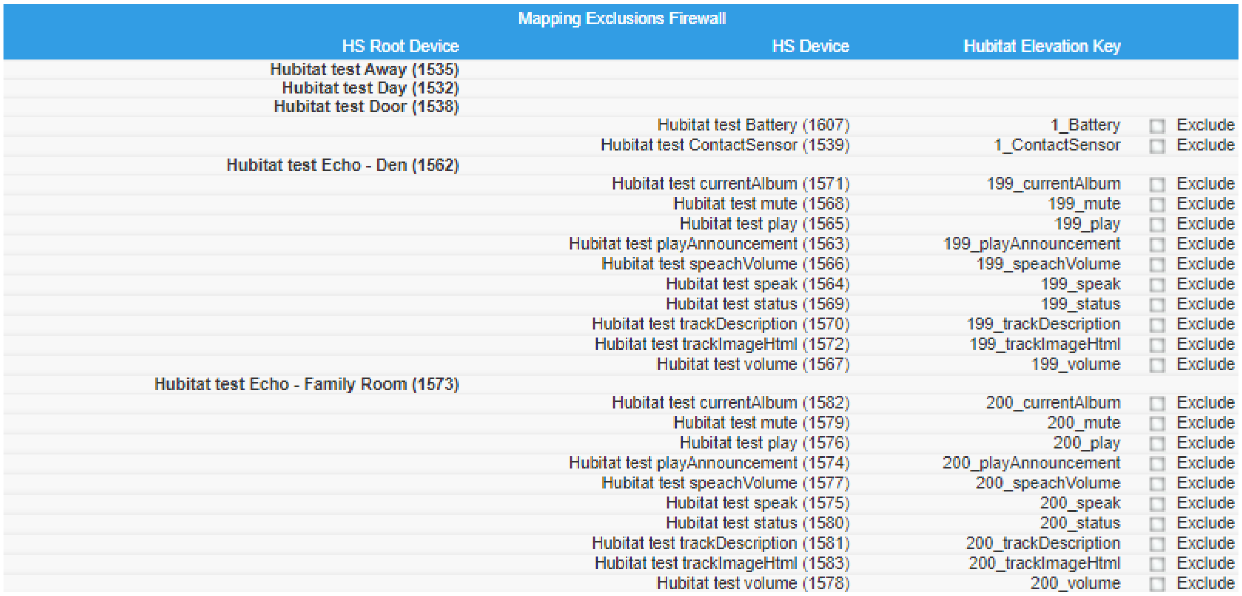
1.3.3 Exclusions Firewall

A set of HS devices are created based upon the information that is provided by Hubitat. There are cases where not all of the information available is of interest. It may be the case that the specific model of a widget that you have does not have some of the features of the generic model provided by Hubitat. It may be that the information from Hubitat will never be used so is just clutter that does not need updating.

The Config page Exclusions Firewall such as shown in Figure 8 is used to selectively remove HS3 child devices or HS4 features. A checked checkbox removes the HS device. It can be restored by removing the checkbox. It is restored automatically in HS4. In HS3 it is done following use of the Connect/Re-Scan button on the same Config page.

If a Hubitat device is to be totally removed, then this is done from the Maker API checkboxes that enable/disable inclusion of the Hubitat device.

Removal of features of a device from HS does not reduce the information from Hubitat, but it does eliminate the creation and update of the features in HS.



Mapping Exclusions Firewall			
HS Root Device	HS Device	Hubitat Elevation Key	
Hubitat test Away (1535)			
Hubitat test Day (1532)			
Hubitat test Door (1538)			
	Hubitat test Battery (1607)	1_Battery	□ Exclude
	Hubitat test ContactSensor (1539)	1_ContactSensor	□ Exclude
Hubitat test Echo - Den (1562)			
	Hubitat test currentAlbum (1571)	199_currentAlbum	□ Exclude
	Hubitat test mute (1568)	199_mute	□ Exclude
	Hubitat test play (1565)	199_play	□ Exclude
	Hubitat test playAnnouncement (1563)	199_playAnnouncement	□ Exclude
	Hubitat test speachVolume (1566)	199_speachVolume	□ Exclude
	Hubitat test speak (1564)	199_speak	□ Exclude
	Hubitat test status (1569)	199_status	□ Exclude
	Hubitat test trackDescription (1570)	199_trackDescription	□ Exclude
	Hubitat test trackImageHtml (1572)	199_trackImageHtml	□ Exclude
	Hubitat test volume (1567)	199_volume	□ Exclude
Hubitat test Echo - Family Room (1573)			
	Hubitat test currentAlbum (1582)	200_currentAlbum	□ Exclude
	Hubitat test mute (1579)	200_mute	□ Exclude
	Hubitat test play (1576)	200_play	□ Exclude
	Hubitat test playAnnouncement (1574)	200_playAnnouncement	□ Exclude
	Hubitat test speachVolume (1577)	200_speachVolume	□ Exclude
	Hubitat test speak (1575)	200_speak	□ Exclude
	Hubitat test status (1580)	200_status	□ Exclude
	Hubitat test trackDescription (1581)	200_trackDescription	□ Exclude
	Hubitat test trackImageHtml (1583)	200_trackImageHtml	□ Exclude
	Hubitat test volume (1578)	200_volume	□ Exclude

Figure 8 Mapping Exclusions Firewall

1.4 Support

The HomeSeer Hubitat Elevation Forum has been setup to communicate on the operation of the Hubitat plug-in. General operational questions and suggestions can be done at this location. It is best to start new threads for new subjects so the information becomes easier to search by others later.

When detailed support information is needed it will usually be most expedient to have information from the "habitat_debug.txt" file made available. It can either be posted in a zip file as an attachment or it can be emailed to mcsSolutions at CenturyTel dot net.

The debug file contains three types of communications. The HS to Hubitat communication will appear like:

```
http://###IP###/apps/api/1/devices/69/poll?###Token###
```

The Hubitat-sent data from startup or Re-Sync button will start with "HubitatDevice:". It will usually be a very long line that includes information about all the Hubitat devices exposed by MakerAPI.

HubitatDevice:[{"name":"DoorSensor","label":"Door","type":"Xiaomi Door/Window Sensor","id":"1","date":"2019-11-08T19:06:43+0000","model":null,"manufacturer":null,"capabilities":["Battery","ContactSensor","Sensor"],"attributes":{"battery":"100","dataType":"STRING","values":null,"batteryLastReplaced":null,"contact":"closed","lastCheckin":"1573240003252","lastClosed":"1573

```
086408704","lastOpened":"1572823005536"},"commands":[{"command":"resetBatteryReplacedDate"},{"command":"resetToClosed"},{"command":"resetToOpen"}]},{"name":"GreenWave PowerNode 6 Child Device","label":"Greenwave switch 1","type":"GreenWave PowerNode 6 Child Device","id":"70","date":"2019-11-07T21:29:15+0000","model":null,"manufacturer":null,"capabilities":["Switch","Refresh","EnergyMeter","PowerMeter"],"attributes":{"energy":"0.0239","dataType":"ENUM","values":["on","off"],"lastupdate":"Nov 07 2019 14:29","power":"0.5","switch":"on"},"commands":[{"command":"off"},{"command":"on"},{"command":"refresh"},{"command":"reset"}]},{"name":"GreenWave PowerNode 6 Child Device","label":"Greenwave switch 2","type":"GreenWave PowerNode 6 Child Device","id":"71","date":"2019-11-07T21:29:15+0000","model":null,"manufacturer":null,"capabilities":["Switch","Refresh","EnergyMeter","PowerMeter"],"attributes":{"energy":"0.0203","dataType":"ENUM","values":["on","off"],"lastupdate":"Nov 07 2019 14:29","power":"0.0","switch":"on"},"commands":[{"command":"off"},{"command":"on"},{"command":"refresh"},{"command":"reset"}]},{"name":"GreenWave PowerNode 6 Child Device","label":"Greenwave switch 3","type":"GreenWave PowerNode 6 Child Device","id":"72","date":"2019-11-07T21:29:15+0000","model":null,"manufacturer":null,"capabilities":["Switch","Refresh","EnergyMeter","PowerMeter"],"attributes":{"energy":"0.0000","dataType":"ENUM","values":["on","off"],"lastupdate":"Nov 07 2019 14:29","power":"0.0","switch":"on"},"commands":[{"command":"off"},{"command":"on"},{"command":"refresh"},{"command":"reset"}]},{"name":"GreenWave PowerNode 6 Child Device","label":"Greenwave switch 4","type":"GreenWave PowerNode 6 Child Device","id":"73","date":"2019-11-07T21:29:15+0000","model":null,"manufacturer":null,"capabilities":["Switch","Refresh","EnergyMeter","PowerMeter"],"attributes":{"energy":"0.0000","dataType":"ENUM","values":["on","off"],"lastupdate":"Nov 07 2019 14:29","power":"0.0","switch":"on"},"commands":[{"command":"off"},{"command":"on"},{"command":"refresh"},{"command":"reset"}]},{"name":"GreenWave PowerNode 6 Child Device","label":"Greenwave switch 5","type":"GreenWave PowerNode 6 Child Device","id":"74","date":"2019-11-07T21:29:15+0000","model":null,"manufacturer":null,"capabilities":["Switch","Refresh","EnergyMeter","PowerMeter"],"attributes":{"energy":"0.0000","dataType":"ENUM","values":["on","off"],"lastupdate":"Nov 07 2019 14:29","power":"0.0","switch":"on"},"commands":[{"command":"off"},{"command":"on"},{"command":"refresh"},{"command":"reset"}]}]
```

When a Hubitat device changes state an event message will be sent by Hubitat. It will start with “Message received:” and look something like the following:

{"source":"DEVICE","name":"switch","displayName":"Greenwave switch 1","value":"on","unit":null,"deviceId":70,"hubId":null,"locationId":null,"installedAppId":null,"descriptionText":null}

Following a command from the plug-in Hubitat will return a full disclosure of the subject device. This starts with “HubitatState:” and looks like the following:

{"id":"69","name":"GreenWave PowerNode 6","label":"PowerStrip6","attributes":[{"name":"energy","currentValue":"0.0368","dataType":"STRING"},{"name":"energy","currentValue":"0.0368","dataType":"NUMBER"},{"name":"energy1","currentValue":null,"dataType":"STRING"},{"name":"energy2","currentValue":null,"dataType":"STRING"},{"name":"energy3","currentValue":null,"dataType":"STRING"},{"name":"energy4","currentValue":null,"dataType":"STRING"},{"name":"energy5","currentValue":null,"dataType":"STRING"},{"name":"energy6","currentValue":null,"dataType":"STRING"},{"name":"lastupdate","currentValue":"Nov 08 2019 12:11","dataType":"STRING"},{"name":"power","currentValue":0.6,"dataType":"STRING"},{"name":"power","currentValue":0.6,"dataType":"NUMBER"},{"name":"power1","currentValue":null,"dataType":"STRING"},{"name":"power2","currentValue":null,"dataType":"STRING"},{"name":"power3","currentValue":null,"dataType":"STRING"},{"name":"power4","currentValue":null,"dataType":"STRING"},{"name":"power5","currentValue":null,"dataType":"STRING"},{"name":"power6","currentValue":null,"dataType":"STRING"},{"name":"switch","currentValue":"on","dataType":"ENUM","values":["on","off"]},{"name":"switch","currentValue":"on","dataType":"STRING"},{"name":"switch1","currentValue":null,"dataType":"STRING"},{"name":"switch2","currentValue":null,"dataType":"STRING"},{"name":"switch3","currentValue":null,"dataType":"STRING"},{"name":"switch4","currentValue":null,"dataType":"STRING"},{"name":"switch5","currentValue":null,"dataType":"STRING"},{"name":"switch6","currentValue":null,"dataType":"STRING"}],"capabilities":["Switch",{"attributes":[{"name":"switch","dataType":null}]},"Polling","Configuration","Refresh","EnergyMeter",{"attributes":[{"name":"energy","dataType":null}]},"PowerMeter",{"attributes":[{"name":"power","dataType":null}]}],"commands":["configure","off","off","off1","off2","off3","off4","off5","off6","on","on","on1","on2","on3","on4","on5","on6","poll","refresh","reset"]}

2 Use Cases and Awesome Stuff

2.1 HubConnect

This is an awesome Hubitat Community developed app.

GitHub: HubConnect

HubConnect Installation Instructions

Hubitat Community Forum Thread

HubConnect replaces the native HubLink/Link to Hub apps and includes the following enhancements:

- Support for multiple device attributes (i.e switch, power, voltage for a Zigbee Plug).
- Battery status is available for any device that supports it.
- Switches, Dimmers, RGB Bulbs, and Buttons are 2-way devices which can be controlled from the coordinator hub as well as the remote hub in which they are connected to.
- Fully bi-directional. Connect physical devices from remote hubs to the coordinator AND devices on the coordinator to remote hubs!
- When a remote device is controlled from master, status updates (i.e. switch) are instantly sent by the remote device to ensure the device actually responded to the command.
- Full bi-directional SmartThings support including 2-way devices.
- Hub Health ensures each remote hub checks in with the coordinator every minute. If a hub fails to respond for 5 minutes it is declared offline.
- A virtual "hub presence" device is created for every linked remote hub. If the remote hub is responding, the status is "present", if it's offline the status is "not present".
- This uses the "HubConnect Beacon Sensor" and is done so rules can be created to notify when a hub is not responding.
- Flexible oAuth endpoints; Hubs do not need to be on the same LAN or even same location.
- Remote hubs can be located anywhere with an internet connection.
- Communications between the master and remote hubs can be suspended for maintenance. (i.e. rebooting the master hub as it prevents the remotes from logging http errors)
- Publish user-defined device drivers without modifying HubConnect source code.
- Mode changes can be pushed from the Server to Remote hubs, including SmartThings.

- Remote hub “Reboot-Recovery” checks-in with the Server hub to set the current system mode anytime a remote hub is rebooted.

3 Built-in Devices

Built-in devices are those that are officially supported by Hubitat and for which the drivers are contained in the released firmware. Most devices follow a standard pattern where the Capabilities presented by the device are mapped into HS Devices. The Attributes of the device represent the current state. The attributes are sometimes an enumeration and sometimes a numeric. For enumerations the HS Devices have VSP definitions created. Graphics using the standard set of symbols provided by HomeSeer are also created for each HS Device.

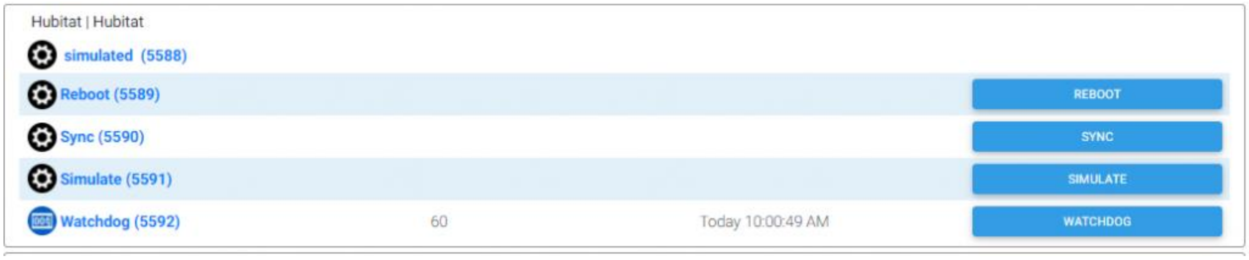
There are also special case devices that have variants of the standard operation and the plug-in needs to handle them in a special way or present a special UI. These special cases are described in this section.

3.1 Hub Device

The top level device for Hubitat Elevation contains the mode information and a set of controls to Reboot, Sync, Simulate and Watchdog. The Simulate and Sync functions are on the Config page for HS3 and as part of the Hub device in HS4.

Hubitat mode information typically takes on one of four states. In HS3 individual buttons are used and in HS4 it is a selector. Sync will establish a communication with Hubitat Elevation to request a disclosure of what devices have been made visible by MakerAPI App within Hubitat Elevation. The Reboot will make a request for the Hubitat Elevation to reboot. The Simulate will use a data file containing prior communication from Hubitat Elevation.

The Watchdog will count up in ten second increments since the last time a communication has been received from Hubitat Elevation. The time of that last communication is reflected in the Last Change property of the Watchdog feature. The Watchdog button is equivalent to the Sync button. The watchdog device is internally generated by mcsHubitat with the intention of providing a means to monitor and trigger events should communications with Hubitat Elevation has been lost.



Hubitat | Hubitat

simulated (5588)

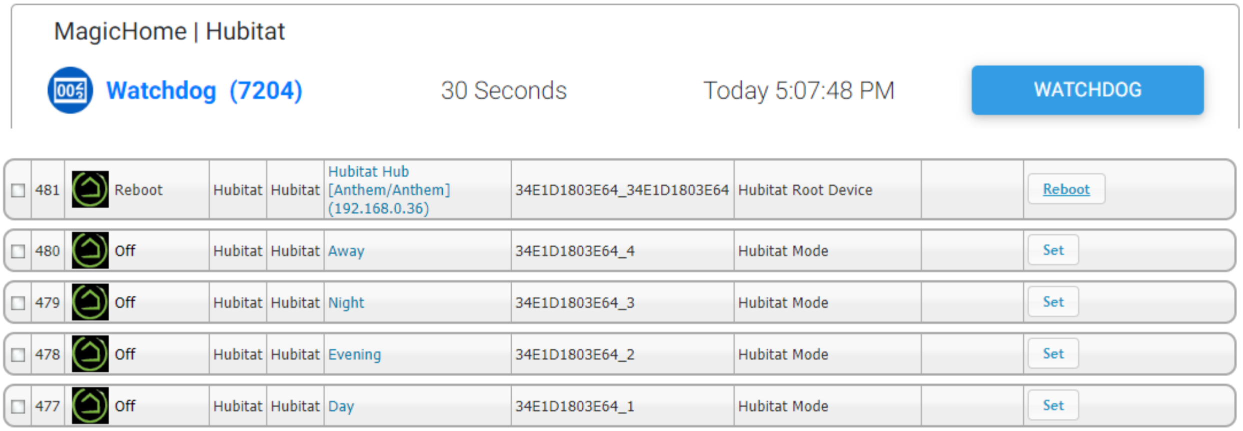
Reboot (5589)

Sync (5590)

Simulate (5591)

Watchdog (5592) 60 Today 10:00:49 AM

Figure 9 HS4 Hub Status and Controls



MagicHome | Hubitat

Watchdog (7204) 30 Seconds Today 5:07:48 PM WATCHDOG

□	481	Reboot	Hubitat	Hubitat	Hubitat Hub [Anthem/Anthem] (192.168.0.36)	34E1D1803E64_34E1D1803E64	Hubitat Root Device		Reboot
□	480	Off	Hubitat	Hubitat	Away	34E1D1803E64_4	Hubitat Mode		Set
□	479	Off	Hubitat	Hubitat	Night	34E1D1803E64_3	Hubitat Mode		Set
□	478	Off	Hubitat	Hubitat	Evening	34E1D1803E64_2	Hubitat Mode		Set
□	477	Off	Hubitat	Hubitat	Day	34E1D1803E64_1	Hubitat Mode		Set

Figure 10 HS3 Hub Status and Controls

3.2 Generic Capabilities

Generic capabilities are those recognized by the plug-in, but not tied specifically to a specific device mode. These are shown in Table 1.

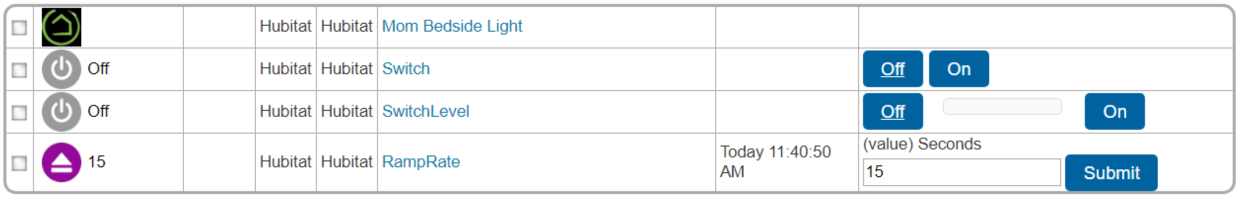
Table 1 Supported Capabilities

AccelerationSensor
Alarm
AtmosphericPressure
Battery
CarbonMonoxideDetector
colorMode
colorName
ColorTemperature
ContactSensor
currentAlbum
EnergyMeter
FanControl
hue
IlluminanceMeasurement
MotionSensor
Mute
play
playAnnouncement
Polling
PowerMeter
PowerSource
PresenceSensor
RampRate

RelativeHumidityMeasurement
RGB
saturation
say
SecurityKeypad
SensitivityLevel
setColor
setStateColor
SmokeDetector
speachVolume
speak
Status
Switch
SwitchLevel
TamperAlert
TemperatureMeasurement
ThreeAxis
TiltAngle
track
trackDescription
trackImageHtml
tvChannel
tvPower
Valve
VibrationSensor
VoltageMeasurement
volume
WaterSensor
WeatherConditions
WindDirection
WindSpeed

3.3 SwitchLevel Capability

SwitchLevel Capability is used for Hubitat devices that can take on a range of levels. These are typically dimmable lights or multi-speed fans. The plug-in creates a RampRate Device for Hubitat devices that are not fans. The HS DeviceValue in the RampRate Device is the number of seconds to go from lowest to highest level of the device.



□			Hubitat	Hubitat	Mom Bedside Light		
□	Off		Hubitat	Hubitat	Switch		Off On
□	Off		Hubitat	Hubitat	SwitchLevel		Off On
□	15		Hubitat	Hubitat	RampRate	Today 11:40:50 AM	(value) Seconds 15 Submit

Figure 11 SwitchLevel with Ramp Rate

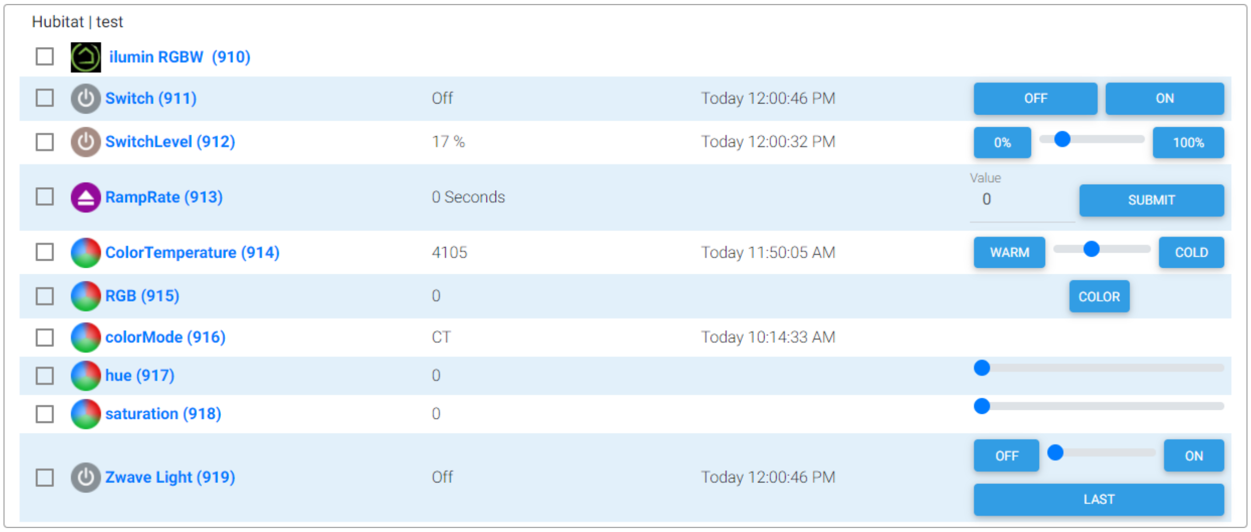
When RampRate device is set to 0 the SwitchLevel will commanded for immediate change. When RampRate is between 1 and 59 seconds the command to Hubitat will include the RampRate and Hubitat will manage the transition time. For RampRate values over 59 seconds the plug-in will send out commands in 1% increments until the target level has been reached.

If one desires a constant transition rate every time the HS Device is changed then the RampRate can be set once and then never changed again.

If the desire is to normally transition immediately, but sometimes transition slowly, then one can setup an event that has four actions. The first is to set the RampRate device to the desired transition rate. The second is to set the device to the desired level. The third is to delay until the device reaches the target level and the fourth to set the RampRate back to 0.

3.4 Emulated Zwave Light Capability

When a Hubitat device contains both a SwitchLevel and a Switch capability then the plug-in will create an additional feature that emulates the operation of a Zwave Light. This provides a single set of controls for On, Off, Brightness and the addition of Last. The Last control will restore the light to the brightness level that existed at the time the On control was used.



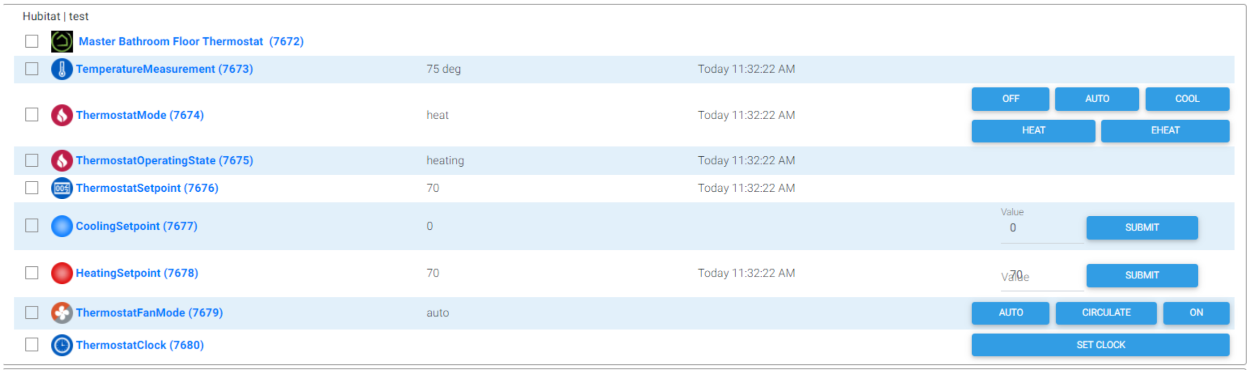
Hubitat | test

□	ilumin RGBW (910)			
□	Switch (911)	Off	Today 12:00:46 PM	OFF ON
□	SwitchLevel (912)	17 %	Today 12:00:32 PM	0% 100%
□	RampRate (913)	0 Seconds		Value 0 SUBMIT
□	ColorTemperature (914)	4105	Today 11:50:05 AM	WARM COLD
□	RGB (915)	0		COLOR
□	colorMode (916)	CT	Today 10:14:33 AM	
□	hue (917)	0		
□	saturation (918)	0		
□	Zwave Light (919)	Off	Today 12:00:46 PM	OFF ON LAST

Figure 12 ZWave Light Capability

3.5 Thermostat Capability

The Hubitat Thermostat Capabilty is expanded by the plug-in into ThermostatSetpoint, CoolingSetpoint, HeatingSetpoint, ThermostatFanMode, ThermostatMode and ThermostatOperatingState HS Devices. ThermostatClock is also added for the Sinope TH1300ZB thermostat.The TemperatureMeasurement Capability that will often be made visible is mapped one-to-one in HS Device.



Hubitat | test

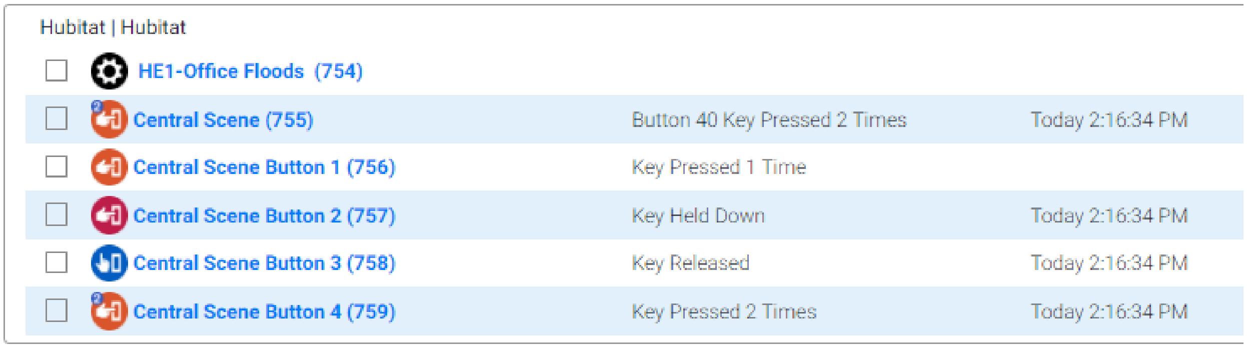
□	Master Bathroom Floor Thermostat (7672)		
□	TemperatureMeasurement (7673)	75 deg	Today 11:32:22 AM
□	ThermostatMode (7674)	heat	Today 11:32:22 AM
□	ThermostatOperatingState (7675)	heating	Today 11:32:22 AM
□	ThermostatSetpoint (7676)	70	Today 11:32:22 AM
□	CoolingSetpoint (7677)	0	
□	HeatingSetpoint (7678)	70	Today 11:32:22 AM
□	ThermostatFanMode (7679)	auto	
□	ThermostatClock (7680)		

Figure 13 Thermostat Capability

3.6 PushableButton / Central Scene Capability

When a Hubitat devices presents a PushableButton Capability for a device type of “FanControl” or "Xiaomi Aqara Vibration Sensor" Capability no HS Device is created for the PushableButon. ReleasableButton and HoldableButton are other button types.

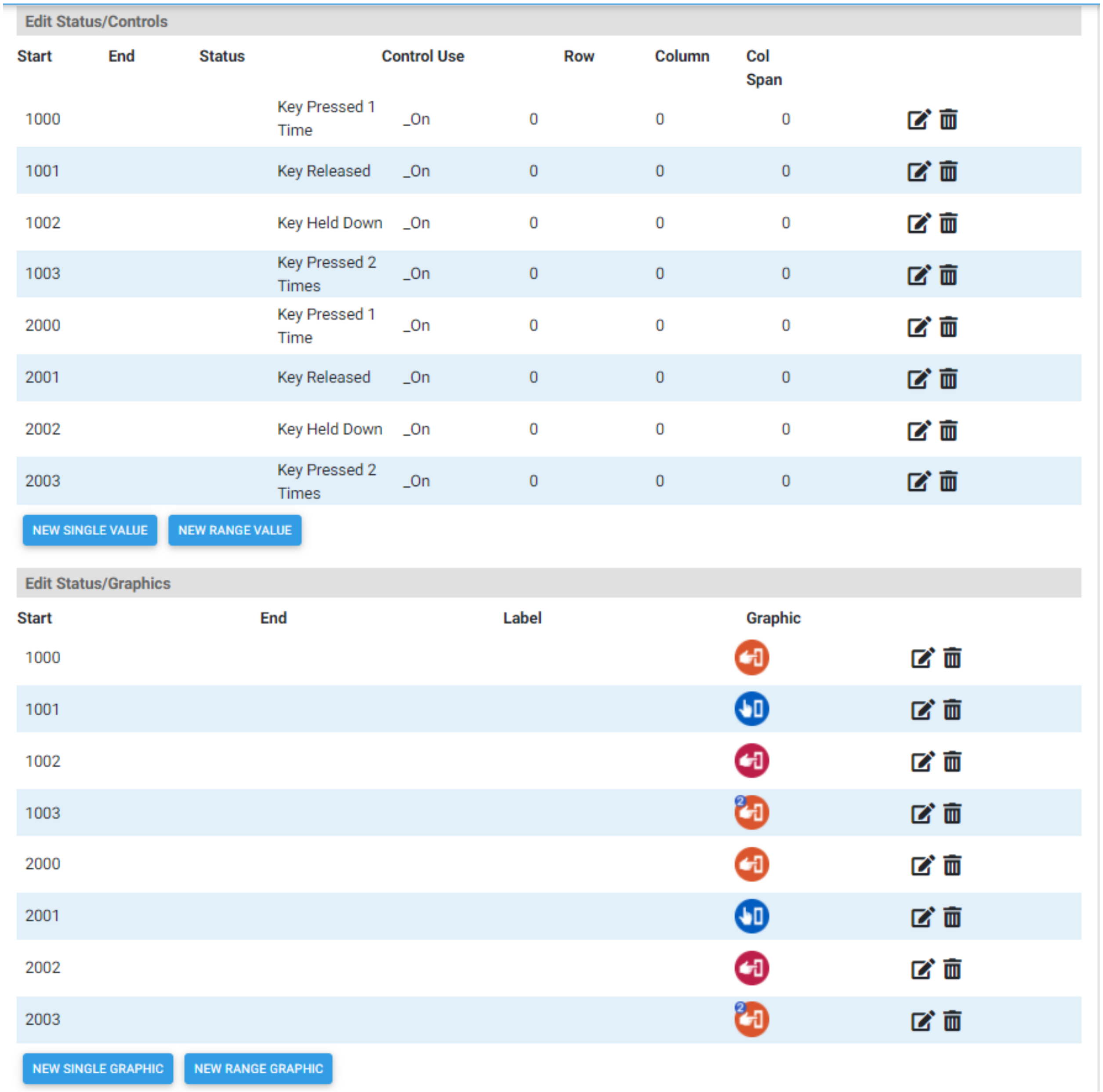
When a PushableButton, HoldableButton and ReleasableButton capability exists with a device then a Central Scene device will be created rather than individual buttons. It will consist of a single feature that encodes the button information as B00A where B is the button number and A is the button action. It will also create a feature for each button that contains the last action reported for that button.



Hubitat | Hubitat

□	HE1-Office Floods (754)		
□	Central Scene (755)	Button 40 Key Pressed 2 Times	Today 2:16:34 PM
□	Central Scene Button 1 (756)	Key Pressed 1 Time	
□	Central Scene Button 2 (757)	Key Held Down	Today 2:16:34 PM
□	Central Scene Button 3 (758)	Key Released	Today 2:16:34 PM
□	Central Scene Button 4 (759)	Key Pressed 2 Times	Today 2:16:34 PM

Figure 14 Central Scene Capability Device



Edit Status/Controls

Start	End	Status	Control Use	Row	Column	Col Span	
1000		Key Pressed 1 Time	_On	0	0	0	
1001		Key Released	_On	0	0	0	
1002		Key Held Down	_On	0	0	0	
1003		Key Pressed 2 Times	_On	0	0	0	
2000		Key Pressed 1 Time	_On	0	0	0	
2001		Key Released	_On	0	0	0	
2002		Key Held Down	_On	0	0	0	
2003		Key Pressed 2 Times	_On	0	0	0	

Edit Status/Graphics

Start	End	Label	Graphic	
1000				
1001				
1002				
1003				
2000				
2001				
2002				
2003				

Figure 15 Central Scene Capabilty Status/Graphics

3.7 ColorControl Capability

Hubitat ColorControl Capability is expanded into HS Devices for RGB, ColorMode, hue, saturation, and colorName.

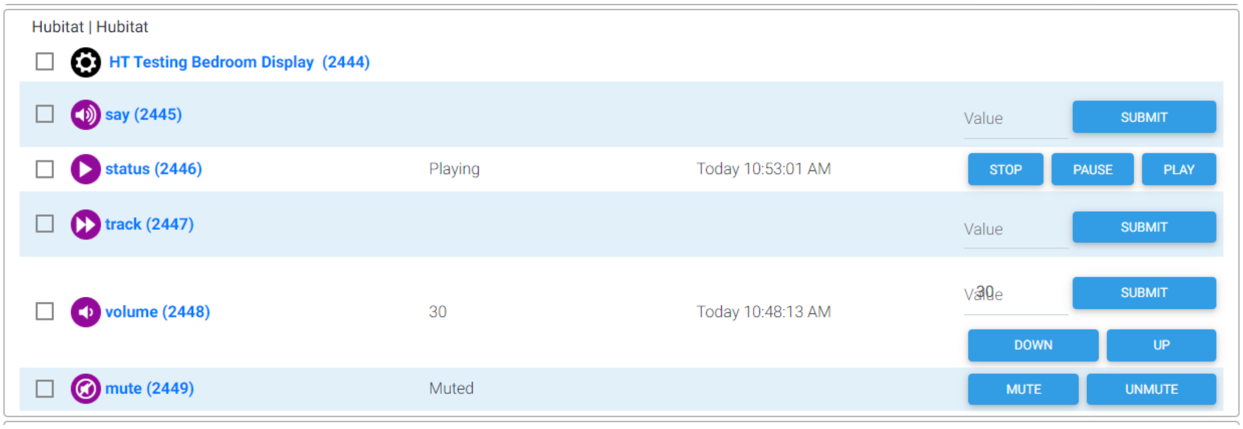
3.8 FanControl Capability

Hubitat device types of “Lutron” and “GE” exhibit non-standard behaviors for FanControl Capability. It is not clear what functionality (if any) exists for these two types of devices.

3.9 Chromecast Video

Chromecast Video adds the SpeechSynthesis capability that is mapped to HS as “say”. This provides for use of Google speaker devices for Text to Speech. Also included are HS devices for media player functions for stop/pause/play and track selection.

AudioVolume capability is represented as a volume device and a mute control



Hubitat | Hubitat

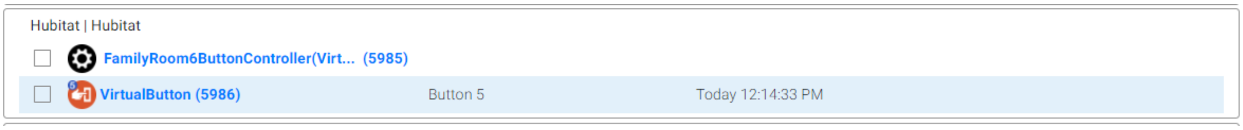
□	HT Testing Bedroom Display (2444)				
□	say (2445)			Value	SUBMIT
□	status (2446)	Playing	Today 10:53:01 AM		STOP PAUSE PLAY
□	track (2447)			Value	SUBMIT
□	volume (2448)	30	Today 10:48:13 AM	Value 30	SUBMIT DOWN UP
□	mute (2449)	Muted			MUTE UNMUTE

Figure 16 Chromecast Video Devices

3.10 Virtual Button Cooper Aspire 6 Button Zwave Controler

The Cooper Aspire controller contains six buttons and advertises a various button push capabilities such as double tap. The Hubitat integration identifies the device type as “Virtual Button”, provides for five buttons and reports only on the “pushed” event with the value of this event being the button number (1 to 5) that was pushed.

The HS Device and Feature shown in Figure 17 will hold the DeviceValue of the last pushed button reported.



Hubitat | Hubitat

□ FamilyRoom6ButtonController(Virt... (5985)

□ VirtualButton (5986) Button 5 Today 12:14:33 PM

Figure 17 Cooper Aspire 6 Button Controller Virtual Button Device and Feature

4 User Supported Devices

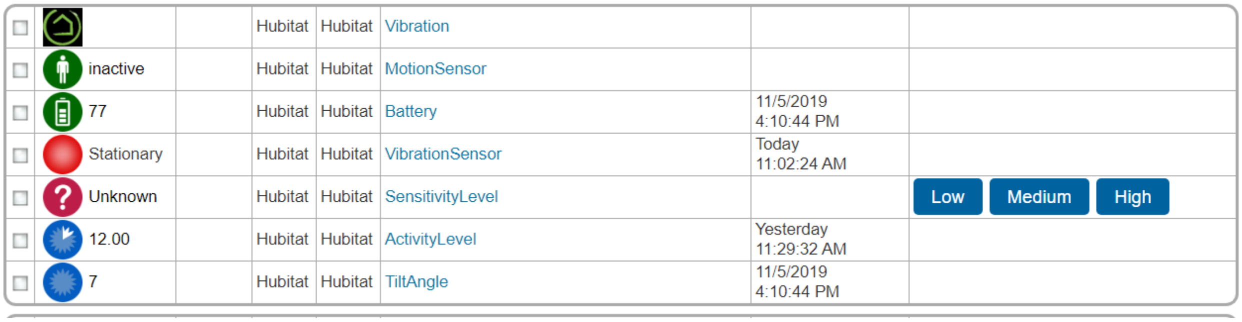
User supported devices are those where the user community has developed drivers and these drivers have not been integrated into the Hubitat Elevation firmware. The procedure for installation of a driver is at https://docs.hubitat.com/index.php?title=How_to_Install_Custom_Drivers.

The following sections provide the integration details of the devices. This includes where the drivers were obtained and a copy of those drivers that was installed in Hubitat Elevation. It also shows the Maker API sourced communication. Red font was used to isolate those parts of the communication that was actually used in the integration via the plug-in.

4.1 Xiaomi Aqara Vibration Sensor DJT11LM

The AccelerationSensor Capabilty for the DJT11LM spawns four HS Devices. SensitivityLevel, VibrationSensor, ActivityLevel and TiltAngle. The MotionSensor and Battery Capabilities are mapped one-to-one to HS Devices.

While SensitivityLevel settings are commanded, the status returned for Sensitivity is always null so it appears that the Hubitat driver may not actually support setting of the sensitivity level.



□			Hubitat	Hubitat	Vibration		
□	inactive		Hubitat	Hubitat	MotionSensor		
□	77		Hubitat	Hubitat	Battery	11/5/2019 4:10:44 PM	
□	Stationary		Hubitat	Hubitat	VibrationSensor	Today 11:02:24 AM	
□	Unknown		Hubitat	Hubitat	SensitivityLevel		Low Medium High
□	12.00		Hubitat	Hubitat	ActivityLevel	Yesterday 11:29:32 AM	
□	7		Hubitat	Hubitat	TiltAngle	11/5/2019 4:10:44 PM	

Figure 18 Vibration Device

4.1.1 Driver Version 0.7.3b

https://raw.githubusercontent.com/veeceeoh/xiaomi-hubitat/master/devicedrivers/xiaomi-aqara-vibration-sensor-hubitat.src/xiaomi-aqara-vibration-sensor-hubitat.groovy

```
/**
 *  Xiaomi Aqara Vibration Sensor - model DJT11LM
 *  Device Driver for Hubitat Elevation hub
 *  Version 0.8b
 *
 *
 *  Licensed under the Apache License, Version 2.0 (the "License"); you may
not use this file except
 *  in compliance with the License. You may obtain a copy of the License at:
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 *  Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed
 *  on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
express or implied. See the License
 *  for the specific language governing permissions and limitations under the
License.
 *
 *  Based on SmartThings device handler code by a4refillpad
 *  Reworked for use with Hubitat Elevation hub by veeceeoh
 *  Additional contributions to code by alecm, alixjg, bspranger, gn0st1c,
foz333, jmagnuson, mike.maxwell, oltman, rinkek, ronvandegraaf, snalee,
tmleafs, twonk, & veeceeoh
 *
 *  Known issues:
 *  + Xiaomi devices send reports based on changes, and a status report every
50-60 minutes. These settings cannot be adjusted.
 *  + The battery level / voltage is not reported at pairing. Wait for the
first status report, 50-60 minutes after pairing.
 *    However, the Aqara Door/Window sensor battery level can be retrieved
immediately with a short-press of the reset button.
 *  + Pairing Xiaomi devices can be difficult as they were not designed to
use with a Hubitat hub.
 *    Holding the sensor's reset button until the LED blinks will start
pairing mode.
 *    3 quick flashes indicates success, while one long flash means pairing
has not started yet.
 *    In either case, keep the sensor "awake" by short-pressing the reset
button repeatedly, until recognized by Hubitat.
 *  + The connection can be dropped without warning. To reconnect, put
Hubitat in "Discover Devices" mode, and follow
 *    the same steps for pairing. As long as it has not been removed from the
Hubitat's device list, when the LED
 *    flashes 3 times, the Aqara Motion Sensor should be reconnected and will
resume reporting as normal
 */

metadata {
        definition (name: "Xiaomi Aqara Vibration Sensor", namespace:
"veeceeoh", author: "veeceeoh") {
                capability "Acceleration Sensor"
                capability "Battery"
                capability "Configuration"
                capability "Contact Sensor"
                capability "Motion Sensor"
                capability "PushableButton"
                capability "Sensor"

                attribute "activityLevel", "String"
                attribute "batteryLastReplaced", "String"
                attribute "lastCheckinEpoch", "String"
                attribute "lastCheckinTime", "Date"
                attribute "lastDropEpoch", "String"
                attribute "lastDropTime", "Date"
                attribute "lastStationaryEpoch", "String"
                attribute "lastStationaryTime", "Date"
                attribute "lastTiltEpoch", "String"
                attribute "lastTiltTime", "Date"
                attribute "lastVibrationEpoch", "String"
                attribute "lastVibrationTime", "Date"
                attribute "sensitivityLevel", "String"
                attribute "sensorStatus", "enum", ["vibrating", "tilted",
"dropped", "Stationary"]
                attribute "tiltAngle", "String"

                fingerprint profileId: "0104", deviceId: "000A", inClusters:
"0000,0003,0019,0101", outClusters: "0000,0004,0003,0005,0019,0101",
manufacturer: "LUMI", model: "lumi.vibration.aq1"

                command "resetBatteryReplacedDate"
                command "setOpenPosition"
                command "setClosedPosition"
                command "SetSensitivityLevelToLow"
                command "SetSensitivityLevelToMedium"
                command "SetSensitivityLevelToHigh"
        }

        preferences {
                //Reset to No Motion Config
                input "motionreset", "number", title: "After
vibration/movement is detected, wait ___ second(s) until resetting 'motion
active' to 'inactive'. (default = 65)", description: "", range: "1..7200"
                //3-Axis Angle Open/Close Position Range Margin of Error
Config
                input name: "marginOfError", title: "Margin of error to use
comparing sensor position to user-set open/close positions (default = 10.0)",
description: "", type: "decimal", range: "0..100"
                //Sensitivity "Lock" Config
                input name: "enableSensCommands", type: "bool", title: "Enable
sensitivity level change command 'buttons' (DISABLES reset button level
change mechanism)", description: ""
                //Battery Voltage Range
                input name: "voltsmin", title: "Min Volts (0% battery = ___
volts, range 2.0 to 2.9). Default = 2.9 Volts", description: "", type:
"decimal", range: "2..2.9"
                input name: "voltsmax", title: "Max Volts (100% battery = ___
volts, range 2.95 to 3.4). Default = 3.05 Volts", description: "", type:
"decimal", range: "2.95..3.4"
                //Date/Time Stamp Events Config
                input name: "lastCheckinEnable", type: "bool", title: "Enable
custom date/time stamp events for lastCheckin", description: ""
```

```
                input name: "otherDateTimeEnable", type: "bool", title: 
"Enable custom date/time stamp events for lastDrop, lastStationary, lastTilt, 
and lastVibration", description: ""
                //Logging Message Config
                input name: "infoLogging", type: "bool", title: "Enable info 
message logging", description: ""
                input name: "debugLogging", type: "bool", title: "Enable debug 
message logging", description: ""
                //Firmware 2.0.5 Compatibility Fix Config
                input name: "oldFirmware", type: "bool", title: "DISABLE 2.0.5 
firmware compatibility fix (for users of 2.0.4 or earlier)", description: ""
        }
}

// Parse incoming device messages to generate events
def parse(String description) {
        displayDebugLog("Parsing message: ${description}")
        Map eventMap = [:]
        def eventType
        if (description?.startsWith('re')) {
                description = description - "read attr - "
                Map descMap = (description).split(",").inject([:]) {
                        map, param ->
                        def nameAndValue = param.split(":")
                        map += 
[(nameAndValue[0].trim()):nameAndValue[1].trim()]
                }
                displayDebugLog("Map of message: ${descMap}")
                def intEncoding = Integer.parseInt(descMap.encoding, 16)
                if (!oldFirmware && descMap.value != null && intEncoding > 
0x18 && intEncoding < 0x3e) {
                        displayDebugLog("Data type of message payload is 
little-endian; reversing byte order")
                        // Reverse order of bytes in description's payload for 
LE data types - required for Hubitat firmware 2.0.5 or newer
                        descMap.value = reverseHexString(descMap.value)
                        displayDebugLog("Reversed payload value: 
${descMap.value}")
                }
                // Send message data to appropriate parsing function based on 
the type of report
                if (descMap.attrId == "0055") {
                        // Handles vibration (value 01), tilt (value 02), and 
drop (value 03) event messages
                        if (descMap.value?.endsWith('0002')) {
                                eventType = 2
                                parseTiltAngle(descMap.value[0..3])
                        } else {
                                eventType = Integer.parseInt(descMap.value,16)
                        }
                        eventMap = mapSensorEvent(eventType)
                } else if (descMap.attrId == "0508") {
                        // Handles XYZ Accelerometer values to determine 
position
                        convertAccelValues(descMap.value)
                } else if (descMap.attrId == "0505") {
                        // Handles recent activity level value reports
                        eventMap = mapActivityLevel(descMap.value)
                } else if (descMap.cluster == "0000" & descMap.attrId ==
"0005") {
                        displayDebugLog "Reset button was short-pressed"
                        if (descMap.value.length() > 45)
                                eventMap =
parseBattery(descMap.value.split("01FF")[1])
                        if (!enableSensCommands) {
                                if (state.sensChangeRequested == true) {
                                        state.sensChangeRequested = false
                                } else {
                                        // set requested level to next in
sequence - low > medium > high > low, etc.
                                        state.requestedSensLevel =
(state.currentSensLevel == null || state.currentSensLevel == 2) ? 0 :
state.currentSensLevel + 1
                                        // Send sensitivity level change
command in 300 milliseconds
                                        runInMillis(300, sendSensLevelCommand)
                                        state.sensChangeRequested = true
                                }
                        }
                } else if (descMap.cluster == "0000" & (descMap.attrId ==
"FF01" || descMap.attrId == "FF02")) {
                        // Parse battery level from hourly announcement message
                        eventMap = (descMap.value.size() > 30) ?
parseBattery(descMap.value) : [:]
                } else
                        displayDebugLog "Unknown read attribute message type,
message not parsed"
        } else if (description?.startsWith('cat')) {
                eventMap = changeSensLevelEvent()
        } else
                displayDebugLog("Unknown message type, message not parsed")
        if (eventMap != [:]) {
                displayDebugLog("Creating event $eventMap")
                return createEvent(eventMap)
        }
}

// Reverses order of bytes in hex string
def reverseHexString(hexString) {
        def reversed = ""
        for (int i = hexString.length(); i > 0; i -= 2) {
                reversed += hexString.substring(i - 2, i )
        }
        return reversed
}

// Convert XYZ Accelerometer values to 3-Axis angle position
private Map convertAccelValues(value) {
        short x = (short)Integer.parseInt(value[8..11],16)
        short y = (short)Integer.parseInt(value[4..7],16)
        short z = (short)Integer.parseInt(value[0..3],16)
        def Psi = Math.round(Math.atan(x/Math.sqrt(z*z+y*y))*1800/Math.PI)/10
        def Phi = Math.round(Math.atan(y/Math.sqrt(x*x+z*z))*1800/Math.PI)/10
        def Theta =
Math.round(Math.atan(z/Math.sqrt(x*x+y*y))*1800/Math.PI)/10
        def descText = "Calculated angles are Psi = ${Psi}°, Phi = ${Phi}°,
Theta = ${Theta}° "
        displayDebugLog("Raw accelerometer XYZ axis values = $x, $y, $z")
        displayDebugLog(descText)
        state.currentAngleX = Psi
        state.currentAngleY = Phi
        state.currentAngleZ = Theta
        if (!state.closedX || !state.openX)
                displayInfoLog("Open/Closed position is unknown because Open
and/or Closed positions have not been set")
        else {
                def cX = state.closedX
                def cY = state.closedY
                def cZ = state.closedZ
                def oX = state.openX
                def oY = state.openY
                def oZ = state.openZ
                // the margin of error value is used to increase/decrease the
area of possible positions considered as open / closed
                def e = (marginOfError) ? marginOfError : 10.0
                def ocPosition = "unknown"
                if ((Psi < cX + e) && (Psi > cX - e) && (Phi < cY + e) && (Phi
> cY - e) && (Theta < cZ + e) && (Theta > cZ - e))
                        ocPosition = "closed"
                else if ((Psi < oX + e) && (Psi > oX - e) && (Phi < oY + e) &&
(Phi > oY - e) && (Theta < oZ + e) && (Theta > oZ - e))
                        ocPosition = "open"
                else
                        displayDebugLog("The current calculated angle position
does not match either stored open/closed positions")
                sendpositionEvent(ocPosition)
        }
}

// Handles Recent Activity level value messages
private Map mapActivityLevel(value) {
        def level = Integer.parseInt(value[0..3],16)
        def descText = "Recent activity level reported at $level"
        displayInfoLog(descText)
        return [
                name: 'activityLevel',
                value: level,
                descriptionText: descText,
        ]
}

// Create map of values to be used for vibration, tilt, or drop event
private Map mapSensorEvent(value) {
        def seconds = (value == 1 || value == 4) ? (motionreset ? motionreset
: 65) : 2
        def time = new Date(now() + (seconds * 1000))
        def statusType = ["Stationary", "Vibration", "Tilt", "Drop", "", ""]
        def eventName = ["", "motion", "acceleration", "button", "motion",
"acceleration"]
```

```
        def eventType = ["", "active", "active", "pushed", "inactive",
"inactive"]
        def eventMessage = ["Sensor is stationary", "Vibration/movement
detected (Motion active)", "Tilt detected (Acceleration active)", "Drop
detected (Button pushed)", "Motion reset to inactive after $seconds seconds",
"Acceleration reset to inactive"]
        if (value < 4) {
                sendEvent(name: "sensorStatus", value: statusType[value],
descriptionText: eventMessage[value])
                updateDateTimeStamp(statusType[value])
        }
        displayInfoLog("${eventMessage[value]}")
        if (value == 0)
                return
        else if (value == 1) {
                runOnce(time, clearmotionEvent)
                state.motionactive = 1
        }
        else if (value == 2)
                runOnce(time, clearaccelEvent)
        else if (value == 3)
                runOnce(time, cleardropEvent)
        return [
                name: eventName[value],
                value: eventType[value],
                descriptionText: (value > 3) ? eventMessage[value] : ""
        ]
}

// Handles tilt angle change message and posts event to update UI tile
display
private parseTiltAngle(value) {
        def angle = Integer.parseInt(value,16)
        def descText = "Tilt angle changed by $angle°"
        sendEvent(
                name: 'tiltAngle',
                value: angle,
                descriptionText : descText,
                isStateChange:true
        )
        displayInfoLog(descText)
}

// Convert raw 4 digit integer voltage value into percentage based on
minVolts/maxVolts range
private parseBattery(description) {
        displayDebugLog("Battery parse string = ${description}")
        def MsgLength = description.size()
        def rawValue
        for (int i = 4; i < (MsgLength-3); i+=2) {
                if (description[i..(i+1)] == "21") { // Search for byte
preceeding battery voltage bytes
                        rawValue = Integer.parseInt((description[(i+4)..(i+5)]
+ description[(i+2)..(i+3)]),16)
                        break
                }
        }
        def rawVolts = rawValue / 1000
        def minVolts = voltsmin ? voltsmin : 2.9
        def maxVolts = voltsmax ? voltsmax : 3.05
        def pct = (rawVolts - minVolts) / (maxVolts - minVolts)
        def roundedPct = Math.min(100, Math.round(pct * 100))
        def descText = "Battery level is ${roundedPct}% (${rawVolts} Volts)"
        // lastCheckinEpoch is for apps that can use Epoch time/date and
lastCheckinTime can be used with Hubitat Dashboard
        if (lastCheckinEnable) {
                sendEvent(name: "lastCheckinEpoch", value: now())
                sendEvent(name: "lastCheckinTime", value: new
Date().toLocaleString())
        }
        displayInfoLog(descText)
        return [
                name: 'battery',
                value: roundedPct,
                unit: "%",
                descriptionText: descText
        ]
}

// Generate lastStationaryEpoch/Time, lastVibrationEpoch/Time,
lastTiltEpoch/Time, or lastDropEpoch/Time event for Epoch time/date app or
Hubitat dashboard use
def updateDateTimeStamp(timeStampType) {
        if (otherDateTimeEnable) {
                displayDebugLog("Setting last${timeStampType}Epoch and
last${timeStampType}Time to current date/time")
                sendEvent(name: "last${timeStampType}Epoch", value: now(),
descriptionText: "Updated button${timeStampType}Epoch")
                sendEvent(name: "last${timeStampType}Time", value: new
Date().toLocaleString(), descriptionText: "Updated
button${timeStampType}Time")
        }
}

def clearmotionEvent() {
        def result = [:]
        if (device.currentState('sensorStatus')?.value == "Vibration")
                mapSensorEvent(0)
        result = mapSensorEvent(4)
        state.motionactive = 0
        displayDebugLog("Sending event $result")
        sendEvent(result)
}

def clearaccelEvent() {
        def result = [:]
        if (device.currentState('sensorStatus')?.value == "Tilt") {
                if (state.motionactive == 1)
                        sendEvent(name: "sensorStatus", value: "Vibration")
                else
                        mapSensorEvent(0)
        }
        result = mapSensorEvent(5)
        displayDebugLog("Sending event $result")
        sendEvent(result)
}

def cleardropEvent() {
        if (device.currentState('sensorStatus')?.value == "Drop") {
                if (state.motionactive == 1)
                        sendEvent(name: "sensorStatus", value: "Vibration")
                else
                        mapSensorEvent(0)
        }
}

def setClosedPosition() {
        if (state.currentAngleX) {
                state.closedX = state.currentAngleX
                state.closedY = state.currentAngleY
                state.closedZ = state.currentAngleZ
                sendpositionEvent("closed")
                displayInfoLog("Closed position successfully set")
                displayDebugLog("Closed position set to $state.closedX°, 
$state.closedY°, $state.closedZ°")
        }
        else
                displayDebugLog("Closed position NOT set because no 3-axis 
accelerometer reports have been received yet")
}

def setOpenPosition() {
        if (state.currentAngleX) {
                state.openX = state.currentAngleX
                state.openY = state.currentAngleY
                state.openZ = state.currentAngleZ
                sendpositionEvent("open")
                displayInfoLog("Open position successfully set")
                displayDebugLog("Open position set to $state.openX°, 
$state.openY°, $state.openZ°")
        }
        else
                displayDebugLog("Open position NOT set because no 3-axis 
accelerometer reports have been received yet")
}

def sendpositionEvent(String ocPosition) {
        def descText = "Calculated position is $ocPosition"
        displayInfoLog(descText)
        sendEvent(name: "contact", value: ocPosition, descriptionText: 
descText)
}

def SetSensitivityLevelToLow() {
        if (enableSensCommands) {
                state.requestedSensLevel = 0
                runInMillis(300, sendSensLevelCommand)
        } else
                log.warn "Set Sensitivity Level Command 'buttons' are 
disabled. Toggle preference setting to enable."
}
```

```
def SetSensitivityLevelToMedium() {
        if (enableSensCommands) {
                state.requestedSensLevel = 1
                runInMillis(300, sendSensLevelCommand)
        } else
                log.warn "Set Sensitivity Level Command 'buttons' are 
disabled. Toggle preference setting to enable."
}

def SetSensitivityLevelToHigh() {
        if (enableSensCommands) {
                state.requestedSensLevel = 2
                runInMillis(300, sendSensLevelCommand)
        } else
                log.warn "Set Sensitivity Level Command 'buttons' are 
disabled. Toggle preference setting to enable."
}

def sendSensLevelCommand() {
        // sensitivity level attribute payload to send - low = 0x15, medium = 
0x0B, high = 0x01
        def attrValue = ["15", "0B", "01"]
        def levelText = ["Low", "Medium", "High"]
        // def cmds = zigbee.writeAttribute(0x0000, 0xFF0D, 0x20, 
attrValue[state.currentSensLevel], [mfgCode: "0x115F"])
        def cmds = [
                "he wattr 0x${device.deviceNetworkId} 0x01 0x0000 0xFF0D 0x20 
{${attrValue[state.requestedSensLevel]}} {115F}", "delay 200"
        ]
        displayDebugLog("Sending commands: $cmds")
        displayInfoLog("Sending request to set sensitivity level to 
${levelText[state.requestedSensLevel]}")
        return cmds
}

def changeSensLevelEvent() {
        def timeDif = now() - 
device.latestState('sensitivityLevel').date.getTime()
        displayDebugLog("Time diff since last sensitivity change event = 
$timeDif")
        if (timeDif > 700) {
                state.currentSensLevel = state.requestedSensLevel
                def levelText = ["Low", "Medium", "High"]
                def descText = "Sensitivity level set to 
${levelText[state.currentSensLevel]}"
                state.sensChangeRequested = false
                displayInfoLog(descText)
                return [
                        name: "sensitivityLevel",
                        value: levelText[state.currentSensLevel],
                        descriptionText: descText
                ]
        } else
                return [:]
}
```

```
// installed() runs just after a sensor is paired
def installed() {
        displayInfoLog("Installing")
        state.prefsSetCount = 0
        mapSensorEvent(0)
        init()
}

// configure() runs after installed() when a sensor is paired
def configure() {
        displayInfoLog("Configuring")
        mapSensorEvent(0)
        init()
        state.prefsSetCount = 1
}

// updated() runs every time user presses save in preference settings page
def updated() {
        displayInfoLog("Updating preference settings")
        init()
        if (lastCheckinEnable)
                displayInfoLog("Last checkin events enabled")
        if (otherDateTimeEnable)
                displayInfoLog("Other date/time stamp events enabled")
        displayInfoLog("Info message logging enabled")
        displayDebugLog("Debug message logging enabled")
}

def init() {
        if (!device.currentValue('batteryLastReplaced'))
                resetBatteryReplacedDate(true)
        if (state.currentSensLevel == null) {
                sendEvent(name: "sensitivityLevel", value: "Unknown",
descriptionText: "Sensitivity level is currently unknown")
                displayInfoLog("Sensitivity level is currently unknown. Please
either use reset button mechanism or one of the Level command buttons to set
the level.")
        }
        sendEvent(name: "numberOfButtons", value: 1)
}

//Reset the batteryLastReplaced date to current date
def resetBatteryReplacedDate(paired) {
        def newlyPaired = paired ? " for newly paired sensor" : ""
        sendEvent(name: "batteryLastReplaced", value: new Date().format("MMM
dd yyyy", location.timeZone))
        displayInfoLog("Setting Battery Last Replaced to current
date${newlyPaired}")
}

private def displayDebugLog(message) {
        if (debugLogging)
                log.debug "${device.displayName}: ${message}"
}

private def displayInfoLog(message) {
        if (infoLogging || state.prefsSetCount != 1)
```

```
                log.info "${device.displayName}: ${message}"
}
```

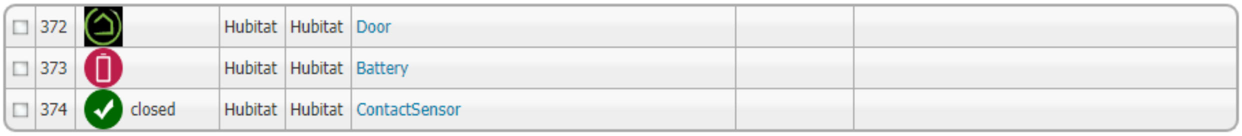
4.1.2 Maker API Full Details

```
{"name":"VibrationSensor",
"label":"Vibration",
"type":"Xiaomi Aqara Vibration Sensor",
"id":"2",
"date":"2019-10-28T18:09:36+0000",
"model":null,
"manufacturer":null,

"capabilities":[
"Battery",          attribute battery
"ContactSensor",
"MotionSensor",     attribute motion
"Configuration",
"AccelerationSensor", attribute sensorStatus shows as VibrationSensor in HS
"Sensor",
"PushableButton"],

"attributes":{
"acceleration":null,
"dataType":"STRING",
"values":null,
  "activityLevel":"6", (49)
  "battery":"100", (100)
"batteryLastReplaced":null,
"contact":null,
"lastCheckinEpoch":null,
"lastCheckinTime":null,
"lastDropEpoch":null,
"lastDropTime":null,
"lastStationaryEpoch":null,
"lastStationaryTime":null,
"lastTiltEpoch":null,
"lastTiltTime":null,
"lastVibrationEpoch":null,
"lastVibrationTime":null,
  "motion":null, (active, inactive)
"numberOfButtons":null,
  "pushed":null,  -- name is button & value is pushed
"sensitivityLevel":null,
  "sensorStatus":null, (Vibration, Drop, Stationary)
"tiltAngle":null},
"commands":[{"command":"SetSensitivityLevelToHigh"},
{"command":"SetSensitivityLevelToLow"},
{"command":"SetSensitivityLevelToMedium"},{"command":"configure"},
{"command":"resetBatteryReplacedDate"},
{"command":"setClosedPosition"},
{"command":"setOpenPosition"}]}]
```

4.1.3 Maker API Event

```
descriptionText: null
deviceId: 2
displayName: "Vibration"
hubId: null
installedAppId: null
locationId: null
name: "button"   --- note pushbutton capability not ported to HS
source: "DEVICE"
unit: null
value: "pushed"

descriptionText: "Sensor is stationary"
deviceId: 2
displayName: "Vibration"
hubId: null
installedAppId: null
locationId: null
name: "sensorStatus"
source: "DEVICE"
unit: null
value: "Stationary"

descriptionText: "Motion reset to inactive after 65 seconds"
deviceId: 2
displayName: "Vibration"
hubId: null
installedAppId: null
locationId: null
name: "motion"
unit: null
value: "inactive"

descriptionText: "Recent activity level reported at 49"
deviceId: 2
displayName: "Vibration"
hubId: null
installedAppId: null
locationId: null
name: "activityLevel"
source: "DEVICE"
unit: null
value: "49"

descriptionText: "Battery level is 100% (3.055 Volts)"
deviceId: 2
displayName: "Vibration"
hubId: null
installedAppId: null
locationId: null
name: "battery"
source: "DEVICE"
unit: "%"
value: "100"
```

4.2 Xiaomi Door/Window Sensor MCCGQ01LM and MCCGQ11LM



□	372	⌂		Hubitat	Hubitat	Door		
□	373	🔋		Hubitat	Hubitat	Battery		
□	374	✓	closed	Hubitat	Hubitat	ContactSensor		

Figure 19 Window-Door Device

4.2.1 Driver Version 0.7.2

https://raw.githubusercontent.com/veeceeoh/xiaomi-hubitat/master/devicedrivers/xiaomi-door-window-sensor-hubitat.src/xiaomi-door-window-sensor-hubitat.groovy

```
/**
 *  Xiaomi "Original" Door/Window Sensor - model MCCGQ01LM
 *  & Aqara Door/Window Sensor - model MCCGQ11LM
 *  Device Driver for Hubitat Elevation hub
 *  Version 0.7.2
 *
 *
 *  Licensed under the Apache License, Version 2.0 (the "License"); you may
not use this file except
 *  in compliance with the License. You may obtain a copy of the License at:
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 *  Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed
 *  on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
express or implied. See the License
 *  for the specific language governing permissions and limitations under the
License.
 *
 *  Based on SmartThings device handler code by a4refillpad
 *  With contributions by alecm, alixjg, bspranger, gn0st1c, foz333,
jmagnuson, mike.maxwell, rinkek, ronvandegraaf, snalee, tmleafs, twonk, &
veeceeoh
 *  Code reworked for use with Hubitat Elevation hub by veeceeoh
 *
 *  Known issues:
 *  + Xiaomi devices send reports based on changes, and a status report every
50-60 minutes. These settings cannot be adjusted.
```

```
 *  + The battery level / voltage is not reported at pairing. Wait for the
first status report, 50-60 minutes after pairing.
 *    However, the Aqara Door/Window sensor battery level can be retrieved
immediately with a short-press of the reset button.
 *  + Pairing Xiaomi devices can be difficult as they were not designed to
use with a Hubitat hub.
 *    Holding the sensor's reset button until the LED blinks will start
pairing mode.
 *    3 quick flashes indicates success, while one long flash means pairing
has not started yet.
 *    In either case, keep the sensor "awake" by short-pressing the reset
button repeatedly, until recognized by Hubitat.
 *  + The connection can be dropped without warning. To reconnect, put
Hubitat in "Discover Devices" mode, and follow
 *    the same steps for pairing. As long as it has not been removed from the
Hubitat's device list, when the LED
 *    flashes 3 times, the Aqara Motion Sensor should be reconnected and will
resume reporting as normal
 *
 */

metadata {
        definition (name: "Xiaomi Door/Window Sensor", namespace: "veeceeoh",
author: "veeceeoh") {
                capability "Contact Sensor"
                capability "Sensor"
                capability "Battery"

                attribute "lastCheckin", "String"
                attribute "lastOpened", "String"
                attribute "lastClosed", "String"
                attribute "batteryLastReplaced", "String"

                // fingerprint for Xiaomi "Original" Door/Window Sensor
                fingerprint endpointId: "01", profileId: "0104", deviceId:
"0104", inClusters: "0000,0003,FFFF,0019", outClusters:
"0000,0004,0003,0006,0008,0005,0019", manufacturer: "LUMI", model:
"lumi.sensor_magnet"

                // fingerprint for Xiaomi Aqara Door/Window Sensor
                fingerprint endpointId: "01", profileId: "0104", deviceId:
"5F01", inClusters: "0000,0003,FFFF,0006", outClusters: "0000,0004,FFFF",
manufacturer: "LUMI", model: "lumi.sensor_magnet.aq2"

                command "resetBatteryReplacedDate"
                command "resetToClosed"
                command "resetToOpen"
        }

        preferences {
                //Battery Voltage Range
                input name: "voltsmin", title: "Min Volts (0% battery = ___
volts, range 2.0 to 2.7). Default = 2.5 Volts", description: "", type:
"decimal", range: "2..2.7"
                input name: "voltsmax", title: "Max Volts (100% battery = ___
volts, range 2.8 to 3.4). Default = 3.0 Volts", description: "", type:
"decimal", range: "2.8..3.4"
```

```
                //Logging Message Config
                input name: "infoLogging", type: "bool", title: "Enable info
message logging", description: ""
                input name: "debugLogging", type: "bool", title: "Enable debug
message logging", description: ""
                //Firmware 2.0.5 Compatibility Fix Config
                input name: "oldFirmware", type: "bool", title: "DISABLE 2.0.5
firmware compatibility fix (for users of 2.0.4 or earlier)", description: ""
        }
}

// Parse incoming device messages to generate events
def parse(String description) {
        def cluster = description.split(",").find {it.split(":")[0].trim() ==
"cluster"}?.split(":")[1].trim()
        def attrId = description.split(",").find {it.split(":")[0].trim() ==
"attrId"}?.split(":")[1].trim()
        def encoding = Integer.parseInt(description.split(",").find
{it.split(":")[0].trim() == "encoding"}?.split(":")[1].trim(), 16)
        def valueHex = description.split(",").find {it.split(":")[0].trim() ==
"value"}?.split(":")[1].trim()
        Map map = [:]

        if (!oldFirmware & valueHex != null & encoding > 0x18 & encoding <
0x3e) {
                displayDebugLog("Data type of payload is little-endian;
reversing byte order")
                // Reverse order of bytes in description's payload for LE data
types - required for Hubitat firmware 2.0.5 or newer
                valueHex = reverseHexString(valueHex)
        }

        displayDebugLog("Parsing message: ${description}")
        displayDebugLog("Message payload: ${valueHex}")

        // lastCheckin can be used with webCoRE
        sendEvent(name: "lastCheckin", value: now())

        // Send message data to appropriate parsing function based on the type
of report
        if (cluster == "0006") {
                // Parse open / closed status report
                map = parseContact(Integer.parseInt(valueHex))
        } else if (attrId == "0005") {
                displayDebugLog("Reset button was short-pressed")
                // Parse battery level from longer type of announcement
message
                map = (valueHex.size() > 60) ?
parseBattery(valueHex.split('FF42')[1]) : [:]
        } else if (attrId == "FF01" || attrId == "FF02") {
                // Parse battery level from hourly announcement message
                map = (valueHex.size() > 30) ? parseBattery(valueHex) : [:]
        } else {
                displayDebugLog("Unable to parse message")
        }

        if (map != [:]) {
                displayInfoLog(map.descriptionText)
                displayDebugLog("Creating event $map")
                return createEvent(map)
        } else
                return [:]
}

// Reverses order of bytes in hex string
def reverseHexString(hexString) {
        def reversed = ""
        for (int i = hexString.length(); i > 0; i -= 2) {
                reversed += hexString.substring(i - 2, i )
        }
        return reversed
}

// Parse open/close report
private parseContact(closedOpen) {
        def value = ["closed", "open"]
        def desc = ["closed", "opened"]
        def coreEvent = ["lastClosed", "lastOpened"]
        displayDebugLog("Setting ${coreEvent[closedOpen]} to current date/time
for webCoRE")
        sendEvent(name: coreEvent[closedOpen], value: now(), descriptionText:
"Updated ${coreEvent[closedOpen]} (webCoRE)")
        return [
                name: 'contact',
                value: value[closedOpen],
                isStateChange: true,
                descriptionText: "Contact was ${desc[closedOpen]}"
        ]
}

// Convert raw 4 digit integer voltage value into percentage based on
minVolts/maxVolts range
private parseBattery(description) {
        displayDebugLog("Battery parse string = ${description}")
        def MsgLength = description.size()
        def rawValue
        for (int i = 4; i < (MsgLength-3); i+=2) {
                if (description[i..(i+1)] == "21") { // Search for byte
preceeding battery voltage bytes
                        rawValue = Integer.parseInt((description[(i+4)..(i+5)]
+ description[(i+2)..(i+3)]),16)
                        break
                }
        }
        def rawVolts = rawValue / 1000
        def minVolts = voltsmin ? voltsmin : 2.5
        def maxVolts = voltsmax ? voltsmax : 3.0
        def pct = (rawVolts - minVolts) / (maxVolts - minVolts)
        def roundedPct = Math.min(100, Math.round(pct * 100))
        def descText = "Battery level is ${roundedPct}% (${rawVolts} Volts)"
        displayInfoLog(descText)
        def result = [
                name: 'battery',
                value: roundedPct,
                unit: "%",
                isStateChange: true,
                descriptionText: descText
        ]
        return result
}

// Manually override contact state to closed
def resetToClosed() {
        if (device.currentState('contact')?.value == "open") {
                displayInfoLog("Manually reset to closed")
                sendEvent(parseContact(0))
        }
}

// Manually override contact state to open
def resetToOpen() {
        if (device.currentState('contact')?.value == "closed") {
                displayInfoLog("Manually reset to open")
                sendEvent(parseContact(1))
        }
}

private def displayDebugLog(message) {
        if (debugLogging)
                log.debug "${device.displayName}: ${message}"
}

private def displayInfoLog(message) {
        if (infoLogging || state.prefsSetCount != 1)
                log.info "${device.displayName}: ${message}"
}

//Reset the batteryLastReplaced date to current date
def resetBatteryReplacedDate(paired) {
        def newlyPaired = paired ? " for newly paired sensor" : ""
        sendEvent(name: "batteryLastReplaced", value: new Date())
        displayInfoLog("Setting Battery Last Replaced to current
date${newlyPaired}")
}

// installed() runs just after a sensor is paired
def installed() {
        displayInfoLog("Installing")
        state.prefsSetCount = 0
}

// configure() runs after installed() when a sensor is paired or reconnected
def configure() {
        displayInfoLog("Configuring")
        init()
        state.prefsSetCount = 1
        return
}

// updated() will run every time user saves preferences
def updated() {
        displayInfoLog("Updating preference settings")
        init()
        displayInfoLog("Info message logging enabled")
        displayDebugLog("Debug message logging enabled")
}

def init() {
        if (!device.currentState('batteryLastReplaced')?.value)
                resetBatteryReplacedDate(true)
}
```

4.2.2 Maker API Full Details

```
{"name":"DoorSensor",
"label":"Door",
"type":"Xiaomi Door/Window Sensor",
"id":"1",
"date":"2019-10-28T21:20:11+0000",
"model":null,
"manufacturer":null,

"capabilities":[
"Battery",              battery attribute
"ContactSensor",        contact attribute
"Sensor"],

"attributes":{
"battery":"100",
"dataType":"STRING",
"values":null,
"batteryLastReplaced":null,
"contact":null,
"lastCheckin":"1572297611412",
"lastClosed":null,
"lastOpened":null},

"commands":[{
"command":"resetBatteryReplacedDate"},{
"command":"resetToClosed"},{
"command":"resetToOpen"}]},
```

4.2.3 Maker API Event

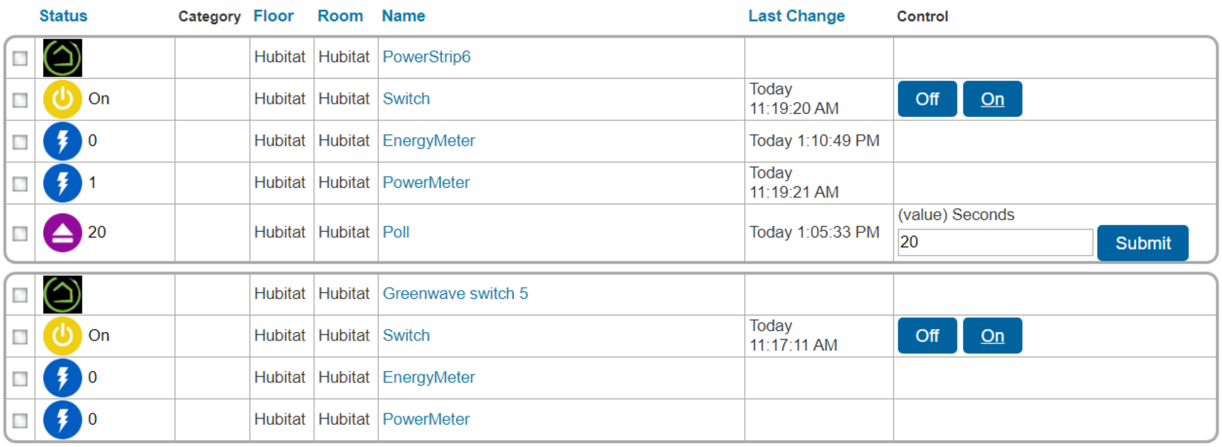
```
{"source":"DEVICE",
"name":"contact",
"displayName":"Door",
"value":"open",
"unit":null,
"deviceId":1,
"hubId":null,
"locationId":null,
"installedAppId":null,
"descriptionText":"Contact was opened"}
```

4.3 Greenwave Power Strip

https://www.monoprice.com/product?p_id=39425&gclid=Cj0KCQiAno_uBRC1ARIsAB496IXwBZJRQ9Lr4gTc7jcdwlpz_7huY9MoimYOjoQ3FaTY7US28idtSzAaArtSEALw_wcB

This device supports individual control of each plug on the powerstrip. On/Off status is provided at time of the on/off control. Energy use is provided when polled. The polling rate can be specified as a HS device value.

.



	Status	Category	Floor	Room	Name	Last Change	Control
□			Hubitat	Hubitat	PowerStrip6		
□	On		Hubitat	Hubitat	Switch	Today 11:19:20 AM	Off On
□	0		Hubitat	Hubitat	EnergyMeter	Today 1:10:49 PM	
□	1		Hubitat	Hubitat	PowerMeter	Today 11:19:21 AM	
□	20		Hubitat	Hubitat	Poll	Today 1:05:33 PM	(value) Seconds 20 Submit
□			Hubitat	Hubitat	Greenwave switch 5		
□	On		Hubitat	Hubitat	Switch	Today 11:17:11 AM	Off On
□	0		Hubitat	Hubitat	EnergyMeter		
□	0		Hubitat	Hubitat	PowerMeter		

Figure 20 Powerstip Device

4.3.1 Driver

Two drivers are installed. One for the power strip and one for reach of the individual recepticles.

4.3.1.1 Parent Driver

Source:
https://github.com/Roysteroonie/Hubitat/blob/master/Greenwave/PowerNode_5/GreenWavePowerNode_5.groovy

```
/***************************************************************************************
*****************************
 *  Copyright: Nick Veenstra
 *
 *  Name: GreenWave PowerNode 5
 *
 *  Date: 2018-01-04
 *
 *  Version: 1.00
 *
 *  Source and info:
https://github.com/CopyCat73/SmartThings/tree/master/devicetypes/copycat73/greenwave-
powernode-6.src
 *
 *  Author: Nick Veenstra
 *  Thanks to David Lomas, Cooper Lee and Eric Maycock for code inspiration
 *
 *  Description: Device handler for the GreenWave PowerNode (multi socket) Z-Wave power outlet
 *
 *  License:
 *   Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except
 *   in compliance with the License. You may obtain a copy of the License at:
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 *   Unless required by applicable law or agreed to in writing, software distributed under the License is
distributed
 *   on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied. See the License
 *   for the specific language governing permissions and limitations under the License.
 *
 *   30/08/2019 Updated by @Royski to work with Hubitat, with help from @chuck.schwer and
@stephack
 *          Solution here https://community.hubitat.com/t/parent-child-device-not-actioning-on-
off/21694/30?u=royski
 *

*****************************************************************************************
****************************/
metadata {
        definition (name: "GreenWave PowerNode 5", namespace: "copycat73", author: "Nick
Veenstra", ocfDeviceType: "oic.d.switch") {
                capability "Energy Meter"
                capability "Switch"
                capability "Power Meter"
                capability "Polling"
                capability "Refresh"
                capability "Configuration"
```

```
		attribute "switch", "string"
		attribute "switch1", "string"
		attribute "switch2", "string"
		attribute "switch3", "string"
		attribute "switch4", "string"
		attribute "switch5", "string"
		//attribute "switch6", "string"
		attribute "power", "string"
		attribute "power1", "string"
		attribute "power2", "string"
		attribute "power3", "string"
		attribute "power4", "string"
		attribute "power5", "string"
		//attribute "power6", "string"
		attribute "energy", "string"
		attribute "energy1", "string"
		attribute "energy2", "string"
		attribute "energy3", "string"
		attribute "energy4", "string"
		attribute "energy5", "string"
		//attribute "energy6", "string"
		attribute "lastupdate", "string"


		command "on"
		command "off"
		command "on1"
		command "off1"
		command "on2"
		command "off2"
		command "on3"
		command "off3"
		command "on4"
		command "off4"
		command "on5"
		command "off5"
		//command "on6"
		//command "off6"
		command "reset"

		fingerprint inClusters :
"0x25,0x32,0x60,0x72,0x86,0x20,0x71,0x70,0x27,0x85,0x87,0x75,0x56"
		fingerprint mfr:"0099", prod:"0003", model:"0004", deviceJoinName: "GreenWave
PowerNode 6"
```

```
    }

    preferences {
        // input name:"updateLight", type:"number", title:"After how many minutes the
GreenWave device should start flashing if the controller didn't communicate with this device",
defaultValue:255


        input name: "logEnable", type: "bool", title: "Enable debug logging", defaultValue: true
    input name: "txtEnable", type: "bool", title: "Enable descriptionText logging", defaultValue: true
    }

}
/*******************************************************************************
****************************
 *  SmartThings System Commands:

********************************************************************************
***************************/
def logsOff(){
   log.warn "debug logging disabled..."
   device.updateSetting("logEnable",[value:"false",type:"bool"])
}

def parse(String description) {
    def result = null
    def cmd = zwave.parse(description)
    if (cmd) {
        result = zwaveEvent(cmd)
        //log.debug "Parsed ${cmd} to ${result.inspect()}"
    } else {
        //log.debug "Non-parsed event: ${description}"
    }
    return result
}

def installed() {
    log.debug "installed"
    createChildDevices()


    //command(zwave.manufacturerSpecificV1.manufacturerSpecificGet())
```

```
		//command(zwave.configurationV1.configurationSet(configurationValue: [255], 
parameterNumber: 1, size: 1))
}

def uninstalled() {
		log.debug "uninstalled()"
		if (childDevices) {
				log.debug "removing child devices"
				removeChildDevices(getChildDevices())
		}
}

private removeChildDevices(delete) {
		delete.each {
				deleteChildDevice(it.deviceNetworkId)
		}
}

def refresh() {
		log.info "Refresh"
		pollNodes()
}

def updated(){
log.info "Greenwave Parent updated"
   log.warn "debug logging is: ${logEnable == true}"
   log.warn "description logging is: ${txtEnable == true}"
   if (logEnable) runIn(1800,logsOff)
}

def initialize() {
		unschedule()
		runEvery5Minutes(pollNodes)
}

def createChildDevices() {
		log.debug "creating child devices"

		try {
		//Unomment 1..6 line below if a 6 strip, else Uncomment line 1..5 for a 5 strip
				//for (i in 1..6) {
				for (i in 1..5) {
				def node = i as String
				def devLabel = "Greenwave switch "+node
```

```
        addChildDevice("copycat73", "GreenWave PowerNode 6 Child Device",
"${device.deviceNetworkId}-ep${i}", [completedSetup: true, label: devLabel,
            isComponent: false, componentName: "switch$i", componentLabel: devLabel])
        }
    } catch (e) {
        log.debug "${e}"
        showAlert("Child device creation failed. Please make sure that the \"GreenWave
PowerNode 6 Child Device\" is installed and published.","childDeviceCreation","failed")
    }
}

private showAlert(text,name,value) {
    sendEvent(
        descriptionText: text,
        eventType: "ALERT",
        name: name,
        value: value,
        displayed: true,
    )
}

def configure() {
    if (txtEnable) log.info "configure()"
    def cmds = []
    cmds << zwave.configurationV1.configurationSet(configurationValue: [255], parameterNumber:
1, size: 1).format()
    delayBetween(cmds)
}

def on() {
    log.info "on"
    [
        zwave.basicV1.basicSet(value: 0xFF).format(),
        zwave.switchBinaryV1.switchBinaryGet().format(),
        "delay 3000",
        zwave.meterV2.meterGet(scale: 2).format()
    ]
}
def off() {
    log.info "off"
    [
        zwave.basicV1.basicSet(value: 0x00).format(),
        zwave.switchBinaryV1.switchBinaryGet().format(),
        "delay 3000",
                        zwave.meterV2.meterGet(scale: 2).format()
            ]
}
def on1() {
            switchOn(1)
}
def off1() {
            switchOff(1)
}
def on2() {
            switchOn(2)
}
def off2() {
            switchOff(2)
}
def on3() {
            switchOn(3)
}
def off3() {
            switchOff(3)
}
def on4() {
            switchOn(4)
}
def off4() {
            switchOff(4)
}
def on5() {
            switchOn(5)
}
def off5() {
            switchOff(5)
}
def on6() {
            switchOn(6)
}
def off6() {
            switchOff(6)
}

void switchOn(Integer node) {
            log.info "${device.label} node ${node} On"
            def cmds = []
            cmds << command(encap(zwave.basicV1.basicSet(value: 0xFF), node))
```

```
        cmds << command(encap(zwave.switchBinaryV1.switchBinaryGet(), node))
        sendHubCommand(new hubitat.device.HubMultiAction(cmds, hubitat.device.Protocol.ZWAVE))
}

void switchOff(node) {
        log.info "${device.label} node ${node} Off"
        def cmds = []
        cmds << command(encap(zwave.basicV1.basicSet(value: 0x00), node))
        cmds << command(encap(zwave.switchBinaryV1.switchBinaryGet(), node))
        sendHubCommand(new hubitat.device.HubMultiAction(cmds, hubitat.device.Protocol.ZWAVE))
}

def poll() {
        pollNodes()
}

def pollNodes() {
        if (logEnable) log.debug "Polling Powerstrip - ${device.label} node ${node}"
        if (txtEnable) log.info "Polling Powerstrip Nodes"
        def cmds = []
        for ( i in 1..5 ) {
                cmds << command(encap(zwave.switchBinaryV1.switchBinaryGet(), i))
                cmds << command(encap(zwave.meterV2.meterGet(scale:0),i))
                cmds << command(encap(zwave.meterV2.meterGet(scale:2),i))
        }
        cmds << zwave.switchBinaryV1.switchBinaryGet().format()
        cmds << zwave.meterV2.meterGet(scale:0).format()
        cmds << zwave.meterV2.meterGet(scale:2).format()
        sendHubCommand(new hubitat.device.HubMultiAction(cmds, hubitat.device.Protocol.ZWAVE))
}

def pollNode(endpoint)  {
        if (logEnable) log.debug "Polling Powerstrip node ${endpoint}"
        def cmds = []
        cmds << command(encap(zwave.switchBinaryV1.switchBinaryGet(),endpoint))
        cmds << command(encap(zwave.meterV2.meterGet(scale:0),endpoint))
        cmds << command(encap(zwave.meterV2.meterGet(scale:2),endpoint))
        sendHubCommand(new hubitat.device.HubMultiAction(cmds, hubitat.device.Protocol.ZWAVE))
}

def updateChildLabel(endpoint) {
        log.debug "update tile label for endpoint $endpoint"
        // tbd
}
```

```
def lastUpdated(time) {
        def timeNow = now()
        def lastUpdate = ""
        if(location.timeZone == null) {
                if (logEnable) log.debug "Cannot set update time : location not defined in app"
        }
        else {
                lastUpdate = new Date(timeNow).format("MMM dd yyyy HH:mm", location.timeZone)
        }
        return lastUpdate
}

def ping() {
        if (logEnable) log.debug "ping() called"
        refresh()
}

def reset() {
        log.info "Resetting kWh for all endpoints"
        def cmds = []
        for ( i in 1..5 ) {
                cmds << command(encap(zwave.meterV2.meterReset(), i))
                cmds << command(encap(zwave.meterV2.meterGet(scale:0),i))
        }
        cmds << command(zwave.meterV2.meterReset())
        cmds << command(zwave.meterV2.meterGet(scale:0))
        sendHubCommand(new hubitat.device.HubMultiAction(cmds, hubitat.device.Protocol.ZWAVE))
}

def resetNode(endpoint) {
        log.info "Resetting kWh for endpoint $endpoint"
        def cmds = []
        cmds << command(encap(zwave.meterV2.meterReset(),endpoint))
        cmds << command(encap(zwave.meterV2.meterGet(scale:0),endpoint))
        sendHubCommand(new hubitat.device.HubMultiAction(cmds, hubitat.device.Protocol.ZWAVE))
}

private encap(cmd, endpoint) {
        if (endpoint) {

        zwave.multiChannelV3.multiChannelCmdEncap(destinationEndPoint:endpoint).encapsulate(cm
d)
        } else {
```

```
                    cmd
          }
}


private command(hubitat.zwave.Command cmd) {
          if (state.sec) {
                    zwave.securityV1.securityMessageEncapsulation().encapsulate(cmd).format()
          } else {
                    cmd.format()
          }
}

private commands(commands, delay=1000) {
          delayBetween(commands.collect{ command(it) }, delay)
}

/*********************************************************************************
****************************
 *  Z-wave Event Handlers.

**********************************************************************************
***************************/

def zwaveEvent(hubitat.zwave.commands.basicv1.BasicReport cmd, ep=null)
{
          if (logEnable) log.debug "Greenwave v1 basic report received"
          if (ep) {
                    def childDevice = childDevices.find{it.deviceNetworkId == "$device.deviceNetworkId-
ep$ep"}
                    if (childDevice)
                    childDevice.sendEvent(name: "switch", value: cmd.value ? "on" : "off")
          } else {
                    def result = createEvent(name: "switch", value: cmd.value ? "on" : "off", type: "digital")
                    def cmds = []
                    cmds << encap(zwave.switchBinaryV1.switchBinaryGet(), 1)
                    cmds << encap(zwave.switchBinaryV1.switchBinaryGet(), 2)

                    return result
          }
}

def zwaveEvent(hubitat.zwave.commands.switchbinaryv1.SwitchBinaryReport cmd, ep=null) {
              if (logEnable) log.debug "Greenwave v1 switchbinary report received for endpoint $ep value
$cmd.value"
              if (ep) {
                    def childDevice = childDevices.find{it.deviceNetworkId == "$device.deviceNetworkId-
ep$ep"}
                    if (childDevice)
                        childDevice.sendEvent(name: "switch", value: cmd.value ? "on" : "off")
                        childDevice.sendEvent(name: 'lastupdate', value: lastUpdated(now()), unit: "")
          } else {
                    def result = createEvent(name: "switch", value: cmd.value ? "on" : "off", type: "digital")
                    sendEvent(name: 'lastupdate', value: lastUpdated(now()), unit: "")
                    def cmds = []
                    cmds << encap(zwave.switchBinaryV1.switchBinaryGet(), 1)
                    cmds << encap(zwave.switchBinaryV1.switchBinaryGet(), 2)

                    return result
          }
}


def zwaveEvent(hubitat.zwave.commands.meterv3.MeterReport cmd, ep=null)
{
          if (logEnable) log.debug "Greenwave v3 meter report received for endpoint $ep scale
$cmd.scale value $cmd.scaledMeterValue"
          def result
          def cmds = []
          if (cmd.scale == 0) {
                    result = [name: "energy", value: cmd.scaledMeterValue, unit: "kWh"]
          } else if (cmd.scale == 1) {
                    result = [name: "energy", value: cmd.scaledMeterValue, unit: "kVAh"]
          } else {
                    result = [name: "power", value: cmd.scaledMeterValue, unit: "W"]
          }
          if (ep) {//
                    def childDevice = childDevices.find{it.deviceNetworkId == "$device.deviceNetworkId-
ep$ep"}
                    if (childDevice)
                        childDevice.sendEvent(result)
          } else {
            sendEvent(name: 'lastupdate', value: lastUpdated(now()), unit: "")
            sendEvent(result)
            (1..5).each { endpoint ->
                        cmds << encap(zwave.meterV2.meterGet(scale: 0), endpoint)
                        cmds << encap(zwave.meterV2.meterGet(scale: 2), endpoint)
```

```
          }

          return result
        }
}
def zwaveEvent(hubitat.zwave.commands.multichannelv3.MultiChannelCmdEncap cmd) {
        if (logEnable) log.debug "Greenwave v3 cMultiChannelCmdEncap command received"
        def encapsulatedCommand = cmd.encapsulatedCommand([0x30: 1, 0x31: 1])
        if (encapsulatedCommand) {
                return zwaveEvent(encapsulatedCommand,cmd.sourceEndPoint)
        }
}

def zwaveEvent(hubitat.zwave.commands.configurationv1.ConfigurationReport cmd) {
        if (logEnable) log.debug "Greenwave v1 configuration report received"
}

def zwaveEvent(hubitat.zwave.commands.configurationv2.ConfigurationReport cmd) {
        if (logEnable) log.debug "Greenwave v2 configuration report received"
}
def zwaveEvent(hubitat.zwave.commands.multichannelv3.MultiChannelCapabilityReport cmd) {
        if (logEnable) log.debug "Greenwave v3 multi channel capability report received"
}
```

4.3.1.2 Child Driver

Source:
https://github.com/Roysteroonie/Hubitat/blob/master/Greenwave/PowerNode_5/GreenWavePowerNode_5_ChildDevice.groovy

```
/*******************************************************************************************
*******************************

 *  Copyright: Nick Veenstra

 *

 *  Name: GreenWave PowerNode 6 Child Device

 *

 *  Date: 2018-01-04

 *

 *  Version: 1.00
 *
 *  Source and info: https://github.com/CopyCat73/SmartThings/tree/master/devicetypes/copycat73/greenwave-powernode-6-child-device.src
 *
 *  Author: Nick Veenstra
 *  Thanks to Eric Maycock for code inspiration
 *
 *  Description: Device handler for the GreenWave PowerNode (multi socket) Z-Wave power outlet child nodes
 *
 *  License:
 *   Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except
 *   in compliance with the License. You may obtain a copy of the License at:
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 *   Unless required by applicable law or agreed to in writing, software distributed under the License is distributed
 *   on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License
 *   for the specific language governing permissions and limitations under the License.
 *
 *   30/08/2019 Updated by @Royski to work with Hubitat, with help from @chuck.schwer and @stephack
 *          Solution here https://community.hubitat.com/t/parent-child-device-not-actioning-on-off/21694/30?u=royski
 *
*****************************************************************************************************************/
metadata {
        definition (name: "GreenWave PowerNode 5 Child Device", namespace: "copycat73", author: 
"Nick Veenstra") {

                capability "Switch"

                capability "Energy Meter"

                capability "Power Meter"

                capability "Refresh"


                attribute "lastupdate", "string"

                command "reset"

        }


        preferences {

                input name: "logEnable", type: "bool", title: "Enable debug logging", defaultValue: true

    input name: "txtEnable", type: "bool", title: "Enable descriptionText logging", defaultValue: true

        }

}

def logsOff(){

  log.warn "debug logging disabled..."

  device.updateSetting("logEnable",[value:"false",type:"bool"])

}

def installed() {

        log.debug "Greenwave child installed"

}

def updated(){

log.info "Greenwave child updated"

  log.warn "debug logging is: ${logEnable == true}"

  log.warn "description logging is: ${txtEnable == true}"

  if (logEnable) runIn(1800,logsOff)

        parent.updateChildLabel(splitChannel(device.deviceNetworkId))
```

```
}
def on() {
        if (logEnable) log.debug "$device.label - On"
        parent.switchOn(splitChannel(device.deviceNetworkId))
}
def off() {
        if (logEnable) log.debug "$device.label - Off"
        parent.switchOff(splitChannel(device.deviceNetworkId))
}
def refresh() {
        if (logEnable) log.debug "$device.label - Refresh"
        parent.pollNode(splitChannel(device.deviceNetworkId))
}
def reset() {
        if (logEnable) log.debug "$device.label - Reset"
        parent.resetNode(splitChannel(device.deviceNetworkId))
}
private splitChannel(String channel) {
   channel.split("-ep") [-1] as Integer
}
```

4.3.2 Maker API Full Details

{"name":"Device",
"label":"PowerStrip",
"type":"GreenWave PowerNode 5",
"id":"33",
"date":"2019-11-05T23:56:31+0000",
"model":null,
"manufacturer":null,

"capabilities":["Switch","Polling","Configuration","Refresh","EnergyMeter","PowerMeter"],

"attributes":{

```
"energy":"0.0000",
"dataType":"STRING",
"values":null,
"energy1":null,
"energy2":null,
"energy3":null,
"energy4":null,
"energy5":null,
"lastupdate":"Nov 05 2019 16:56",
"power":"0.0",
"power1":null,
"power2":null,
"power3":null,
"power4":null,
"power5":null,
"switch":"on",
"switch1":null,
"switch2":null,
"switch3":null,
"switch4":null,
"switch5":null},

"commands":[
{"command":"configure"},
{"command":"off"},
{"command":"off"},
{"command":"off1"},
{"command":"off2"},
{"command":"off3"},
{"command":"off4"},
{"command":"off5"},
{"command":"on"},
{"command":"on"},
{"command":"on1"},
{"command":"on2"},
{"command":"on3"},
{"command":"on4"},
{"command":"on5"},
{"command":"poll"},
{"command":"refresh"},
{"command":"reset"}]},
```

4.3.3 Maker API Event

```
{"source":"DEVICE",
"name":"energy",
"displayName":"PowerStrip",
"value":"0.0000",
"unit":"kWh",
"deviceId":33,
"hubId":null,
"locationId":null,
"installedAppId":null,
"descriptionText":null}

{"source":"DEVICE",
"name":"power",
"displayName":"PowerStrip",
"value":"0.0",
"unit":"W",
"deviceId":33,
"hubId":null,
"locationId":null,
"installedAppId":null,
"descriptionText":null}

{"source":"DEVICE",
"name":"switch",
"displayName":"Greenwave switch 3",
"value":"on",
"unit":null,
"deviceId":72,
"hubId":null,
"locationId":null,
"installedAppId":null,
"descriptionText":null}
```

4.4 Xiaomi Mi Aqara Majic Cube

https://www.ebay.com/itm/Xiaomi-Mi-Wireless-Smart-Home-AQara-Magic-Cube-Controller-6-Actions-Operation/352522717141?_trkparms=aid%3D555018%26algo%3DPL.SIM%26ao%3D1%26asc%3D20190212102350%26meid%3D7b5307bd3ed642569961ae4d38537ac4%26pid%3D100012%26rk%3D2%26rkt%3D12%26sd%3D163747520039%26itm%3D352522717141%26pmt%3D1%26noa%3D0%26pg%3D2047675&_trksid=p2047675.c100012.m1985

4.4.1 Driver

https://github.com/veeceeoh/xiaomi-hubitat/blob/master/devicedrivers/xiaomi-cube-controller-hubitat.src/xiaomi-aqara-wireless-switch.groovy

```
/**
 *  IMPORT URL: https://raw.githubusercontent.com/veeceeoh/xiaomi-hubitat/master/devicedrivers/xiaomi-cube-controller-hubitat.src/xiaomi-aqara-wireless-switch.groovy
 *
 *  Xiaomi Mi Cube Controller - model MFKZQ01LM
 *  Device Driver for Hubitat Elevation hub
 *  Version: 0.3.3b
 *
 *
 *  Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except
 *  in compliance with the License. You may obtain a copy of the License at:
```

```
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 *  Unless required by applicable law or agreed to in writing, software distributed under the License is distributed
 *  on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License
 *  for the specific language governing permissions and limitations under the License.
 *
 *  Based on original SmartThings device handler code by Oleg "DroidSector" Smirnov and Artur Draga
 *  With contributions by alecm, alixjg, bspranger, gn0st1c, foz333, jmagnuson, mike.maxwell, rinkek, ronvandegraaf, snalee, tmleafs, twonk, & veeceeoh
 *  Reworked and additional code for use with Hubitat Elevation hub by veeceeoh
 *
 *  Known issues:
 *  + Xiaomi devices send reports based on changes, and a status report every 50-60 minutes. These settings cannot be adjusted.
 *  + The battery level / voltage is not reported at pairing. Wait for the first status report, 50-60 minutes after pairing.
 *  + Pairing Xiaomi devices can be difficult as they were not designed to use with a Hubitat hub.
 *    Holding the Cube's reset button (inside, next to the battery) until the LED blinks will start pairing mode.
 *    3 quick flashes indicates success, while one long flash means pairing has not started yet.
 *    In either case, keep the sensor "awake" by short-pressing the reset button repeatedly, until recognized by Hubitat.
 *  + The connection can be dropped without warning. To reconnect, put Hubitat in "Discover Devices" mode, and follow
 *    the same steps for pairing. As long as it has not been removed from the Hubitat's device list, when the LED
 *    flashes 3 times, the sensor should be reconnected and will resume reporting as normal
 *
 */
metadata {
       definition (name: "Xiaomi Mi Cube Controller", namespace: "veeceeoh", author: "veeceeoh", importUrl: "https://raw.githubusercontent.com/veeceeoh/xiaomi-hubitat/master/devicedrivers/xiaomi-cube-controller-hubitat.src/xiaomi-aqara-wireless-switch.groovy") {
               capability "Battery"
               capability "Configuration"
```

```
        capability "PushableButton"
        capability "Three Axis"
        capability "Sensor"
        attribute "angle", "number"
        attribute "face", "number"
        attribute "lastCheckinEpoch", "String"
        attribute "lastCheckinTime", "String"
        attribute "batteryLastReplaced", "String"
        // Fingerprint data used to match driver to device during pairing
        fingerprint profileId: "0104", inClusters: "0000,0003,0019,0012", outClusters: "0000,0004,0003,0005,0019,0012", deviceJoinName: "Xiaomi Mi Cube"
        fingerprint profileId: "0104", inClusters: "0000,0003,0019,0012", outClusters: "0000,0004,0003,0005,0019,0012", model: "lumi.sensor_cube", deviceJoinName: "Xiaomi Mi Cube"
        fingerprint profileId: "0104", deviceId: "5F01", inClusters: "0000,0003,0019,0012", outClusters: "0000,0004,0003,0005,0019,0012", model: "lumi.sensor_cube", deviceJoinName: "Xiaomi Mi Cube"
        command "setFace0"
        command "setFace1"
        command "setFace2"
        command "setFace3"
        command "setFace4"
        command "setFace5"
        command "flip90"
        command "flip180"
        command "slide"
        command "knock"
        command "rotateR"
        command "rotateL"
        command "shake"
        command "resetBatteryReplacedDate"
    }
    preferences {
        //Cube Mode Config
        input (name: "cubeMode", title: "Cube Mode: Select how many buttons to control", description: "", type: "enum", options: [0: "Simple - 7 buttons", 1: "Advanced - 36 buttons", 2: "Combined - 43 buttons"])
        //Battery Voltage Range
        input name: "voltsmin", title: "Min Volts (0% battery = ___ volts, range 2.0 to 2.9). Default = 2.9 Volts", description: "", type: "decimal", range: "2..2.9"
        input name: "voltsmax", title: "Max Volts (100% battery = ___ volts, range 2.95 to 3.4). Default = 3.05 Volts", description: "", type: "decimal", range: "2.95..3.4"
```

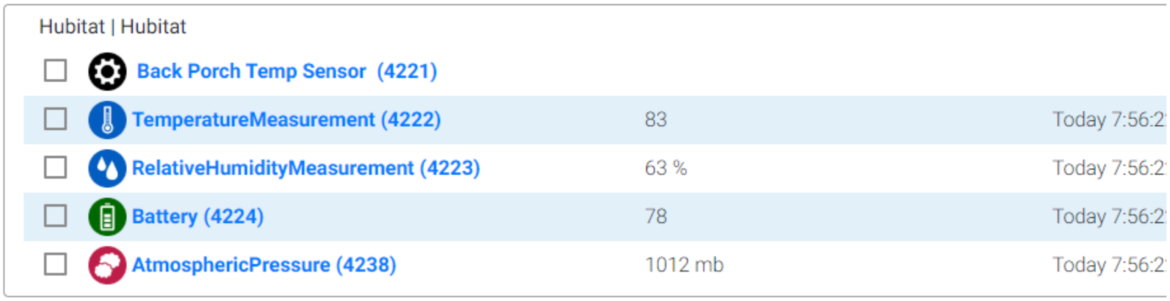
```
                //Date/Time Stamp Events Config
                input name: "lastCheckinEnable", type: "bool", title: "Enable custom
date/time stamp events for lastCheckin", description: ""
                //Logging Message Config
                input name: "infoLogging", type: "bool", title: "Enable info message
logging", description: ""
                input name: "debugLogging", type: "bool", title: "Enable debug message
logging", description: ""
                //Firmware 2.0.5 Compatibility Fix Config
                input name: "oldFirmware", type: "bool", title: "DISABLE 2.0.5 firmware
compatibility fix (for users of 2.0.4 or earlier)", description: ""
        }
}
// Parse incoming device messages to generate events
def parse(String description) {
        displayDebugLog("Parsing message: ${description}")
        Map eventMap = [:]
        if (description?.startsWith('re')) {
                description = description - "read attr - "
                Map descMap = (description).split(",").inject([:]) {
                        map, param ->
                        def nameAndValue = param.split(":")
                        map += [(nameAndValue[0].trim()):nameAndValue[1].trim()]
                }
                displayDebugLog("Map of message: ${descMap}")
                def intEncoding = Integer.parseInt(descMap.encoding, 16)
                if (!oldFirmware && descMap.value != null && intEncoding > 0x18 &&
intEncoding < 0x3e) {
                        displayDebugLog("Data type of message payload is little-endian;
reversing byte order")
                        // Reverse order of bytes in description's payload for LE data
types - required for Hubitat firmware 2.0.5 or newer
                        descMap.value = reverseHexString(descMap.value)
                        displayDebugLog("Reversed payload value: ${descMap.value}")
                }
                // Send message data to appropriate parsing function based on the type
of report
                if (descMap.cluster == "0012" && descMap.attrId == "0055") {
                        // Shake, flip, knock, slide messages
                        getMotionResult(descMap.value)
                } else if (descMap.cluster == "000C" && descMap.attrId == "FF05") {
                        // Rotation (90 and 180 degrees)
                        getRotationResult(descMap.value)
                } else if (descMap.cluster == "0000" && descMap.attrId == "0005") {
                        displayInfoLog("Reset button was short-pressed")
                        // Parse battery level from longer type of announcement message
                        eventMap = (descMap.value.size() > 60) ?
parseBattery(descMap.value.split('FF42')[1]) : [:]
                } else if (descMap.cluster == "0000" & (descMap.attrId == "FF01" ||
descMap.attrId == "FF02")) {
                        // Parse battery level from hourly announcement message
                        eventMap = (descMap.value.size() > 30) ?
parseBattery(descMap.value) : [:]
                } else
                        displayDebugLog("Unable to parse message")
        } else if (description?.startsWith('cat')) {
                displayDebugLog("No action taken on 'catchall' message")
        } else
                displayDebugLog("Unknown message type, message not parseable")
        if (eventMap != [:]) {
                displayInfoLog(eventMap.descriptionText)
                displayDebugLog("Creating event $eventMap")
                return createEvent(eventMap)
        } else
                return eventMap
}
// Reverses order of bytes in hex string
def reverseHexString(hexString) {
        def reversed = ""
        for (int i = hexString.length(); i > 0; i -= 2) {
                reversed += hexString.substring(i - 2, i )
        }
        return reversed
}
private Map getMotionResult(String value) {
        String motionType = value[0..1]
        String binaryValue = hexToBin(value[2..3])
        Integer sourceFace = Integer.parseInt(binaryValue[2..4],2)
        Integer targetFace = Integer.parseInt(binaryValue[5..7],2)
        displayDebugLog("motionType: $motionType, binaryValue: $binaryValue, sourceFace:
$sourceFace, targetFace: $targetFace")
        if (motionType == "00") {
                switch(binaryValue[0..1]) {
                        case "00":
                                if (targetFace == 0)
                                        shakeEvents()
                                break
                        case "01":
                                flipEvents(targetFace, "90")
                                break
                        case "10":
                                flipEvents(targetFace, "180")
                                break
                }
        } else if (motionType == "01") {
                slideEvents(targetFace)
        } else if (motionType == "02") {
                knockEvents(targetFace)
        }
}
private Map getRotationResult(value) {
        Integer angle =
Math.round(Float.intBitsToFloat(Long.parseLong(value[0..7],16).intValue()))
        rotateEvents(angle)
}
def String hexToBin(String thisByte, Integer size = 8) {
        String binaryValue = new BigInteger(thisByte, 16).toString(2);
        return String.format("%${size}s", binaryValue).replace(' ', '0')
}
def Map shakeEvents() {
        def descText
        if (!settings.cubeMode || settings.cubeMode in ['0','2'] ) {
                descText = (settings.cubeMode == '0') ? "Shake detected (button 1
pushed)" : null
                sendEvent([
                        name: "pushed",
                        value: 1,
                        data: [face: device.currentValue("face")],
                        descriptionText: descText,
                        isStateChange: true
                ])
                if (descText)
                        displayInfoLog(descText)
        }
        if (settings.cubeMode in ['1','2'] ){
                def buttonNum = (device.currentValue("face") as Integer) +
((settings.cubeMode == '1') ? 31 : 38)
                descText = "Shake detected with face #${device.currentValue("face")} up
(button $buttonNum pushed)"
```

```
                sendEvent([
                        name: "pushed",
                        value: buttonNum,
                        data: [face: device.currentValue("face")],
                        descriptionText: descText,
                        isStateChange: true
                ])
                displayInfoLog(descText)
        }
}
def flipEvents(Integer faceId, String flipType) {
        def descText
        if (flipType == "0") {
                sendEvent([
                        name: 'face',
                        value: -1,
                        isStateChange: false
                ])
                sendEvent([
                        name: 'face',
                        value: faceId,
                        isStateChange: false
                ])
        } else if (flipType == "90") {
                descText = (settings.cubeMode == '0') ? "90° flip detected (button 2
pushed)" : null
                if (settings.cubeMode in ['0','2']) {
                        sendEvent([
                                name: "pushed",
                                value: 2 ,
                                data: [face: faceId],
                                descriptionText: descText,
                                isStateChange: true
                        ])
                        if (descText)
                                displayInfoLog(descText)
                }
        } else if (flipType == "180") {
                descText = (settings.cubeMode == '0') ? "180° flip detected (button 3
pushed)" : null
                if (settings.cubeMode in ['0','2']) {
                        sendEvent([
                                name: "pushed",
                value: 3 ,
                data: [face: faceId],
                descriptionText: descText,
                isStateChange: true
            ])
            if (descText)
                displayInfoLog(descText)
        }
    }
    sendEvent([
        name: 'face',
        value: faceId,
        displayed: false
    ])
    if (settings.cubeMode in ['1','2']) {
        def buttonNum = faceId + ((settings.cubeMode == '1') ? 1 : 8)
        descText = "Flip to face #$faceId detected (button $buttonNum pushed)"
        sendEvent([
            name: "pushed",
            value: buttonNum,
            data: [face: faceId],
            descriptionText: descText,
            isStateChange: true
        ])
        displayInfoLog(descText)
    }
    switch (faceId) {
        case 0: sendEvent([name: "threeAxis", value: "[x:0,y:-1000,z:0]"]);
break
        case 1: sendEvent([name: "threeAxis", value: "[x:-1000,y:0,z:0]"]);
break
        case 2: sendEvent([name: "threeAxis", value: "[x:0,y:0,z:1000]"]); break
        case 3: sendEvent([name: "threeAxis", value: "[x:1000,y:0,z:0]"]); break
        case 4: sendEvent([name: "threeAxis", value: "[x:0,y:1000,z:0]"]); break
        case 5: sendEvent([name: "threeAxis", value: "[x:0,y:0,z:-1000]"]);
break
    }
}
def Map slideEvents(Integer targetFace) {
    def descText
    if (targetFace != device.currentValue("face") as Integer) {
        displayInfoLog("Stale face data, updating")
        setFace(targetFace)
    }
    if (!settings.cubeMode || settings.cubeMode in ['0','2']) {
        descText = (settings.cubeMode == '0') ? "Slide detected (button 4
pushed)" : null
        sendEvent([
            name: "pushed",
            value: 4,
            data: [face: targetFace],
            descriptionText: descText,
            isStateChange: true
        ])
        if (descText)
            displayInfoLog(descText)
    }
    if (settings.cubeMode in ['1','2']) {
        def buttonNum = targetFace+((settings.cubeMode == '1') ? 7 : 14)
        descText = "Slide detected with face #$targetFace up (button $buttonNum
pushed)"
        sendEvent([
            name: "pushed",
            value: buttonNum,
            data: [face: targetFace],
            descriptionText: descText,
            isStateChange: true
        ])
        displayInfoLog(descText)
    }
}
def knockEvents(Integer targetFace) {
    def descText
    if (targetFace != device.currentValue("face") as Integer) {
        displayInfoLog("Stale face data, updating")
        setFace(targetFace)
    }
    if (!settings.cubeMode || settings.cubeMode in ['0','2']) {
        descText = (settings.cubeMode == '0') ? "Knock detected (button 5
pushed)" : null
        sendEvent([
            name: "pushed",
            value: 5,
            data: [face: targetFace],
            descriptionText: descText,
            isStateChange: true
        ])
        if (descText)
            displayInfoLog(descText)
    }
    if (settings.cubeMode in ['1','2']) {
        def buttonNum = targetFace+((settings.cubeMode == '1') ? 13 : 20)
        descText = "Knock detected with face #$targetFace up (button $buttonNum
pushed)"
        sendEvent([
            name: "pushed",
            value: buttonNum,
            data: [face: targetFace],
            descriptionText: descText,
            isStateChange: true
        ])
        displayInfoLog(descText)
     }
}
def rotateEvents(Integer angle) {
    sendEvent([
        name: "angle",
        value: angle,
        isStateChange: true
    ])
    displayInfoLog("Rotated by $angle°")
    def descText
    def buttonNum
    if (angle > 0) {
        if (!settings.cubeMode || settings.cubeMode in ['0','2'] ) {
            descText = (settings.cubeMode == '0') ? "Right rotation (button 6
pushed)" : null
            sendEvent([
                name: "pushed",
                value: 6,
                data: [face: device.currentValue("face"), angle: angle],
                descriptionText: descText,
                isStateChange: true
            ])
            if (descText)
                displayInfoLog(descText)
        }
        if (settings.cubeMode in ['1','2']) {
            buttonNum = (device.currentValue("face") as Integer) +
((settings.cubeMode == '1') ? 19 : 26)
            descText = "Right rotation on face
#${device.currentValue("face")} (button $buttonNum pushed)"
            sendEvent([
                name: "pushed",
                value: buttonNum,
                data: [face: device.currentValue("face")],
                descriptionText: descText,
                isStateChange: true
            ])
            displayInfoLog(descText)
        }
    } else {
        if (!settings.cubeMode || settings.cubeMode in ['0','2']) {
            descText = (settings.cubeMode == '0') ? "Left rotation (button 7
pushed)" : null
            sendEvent([
                name: "pushed",
                value: 7,
                data: [face: device.currentValue("face"), angle: angle],
                descriptionText: descText,
                isStateChange: true
            ])
            if (descText)
                displayInfoLog(descText)
        }
        if (settings.cubeMode in ['1','2']) {
            buttonNum = (device.currentValue("face") as Integer) +
((settings.cubeMode == '1') ? 25 : 32)
            descText = "Left rotation on face #${device.currentValue("face")}
(button $buttonNum pushed)"
            sendEvent([
                name: "pushed",
                value: buttonNum,
                data: [face: device.currentValue("face")],
                descriptionText: descText,
                isStateChange: true
            ])
            displayInfoLog(descText)
        }
    }
}
def setFace(Integer faceId) {
        def Integer prevFaceId = device.currentValue("face")
        if (prevFaceId == faceId) {
                flipEvents(faceId, "0")
        } else if ((prevFaceId == 0 && faceId == 3)||(prevFaceId == 1 && faceId ==
4)||(prevFaceId == 2 && faceId == 5)||(prevFaceId == 3 && faceId == 0)||(prevFaceId ==
4 && faceId == 1)||(prevFaceId == 5 && faceId == 2)){
                flipEvents(faceId, "180")
        } else {
                flipEvents(faceId, "90")
        }
}
// Convert raw 4 digit integer voltage value into percentage based on minVolts/maxVolts
range
private parseBattery(description) {
        displayDebugLog("Battery parse string = ${description}")
        def MsgLength = description.size()
        def rawValue
        for (int i = 4; i < (MsgLength-3); i+=2) {
                if (description[i..(i+1)] == "21") { // Search for byte preceeding
battery voltage bytes
                        rawValue = Integer.parseInt((description[(i+4)..(i+5)] +
description[(i+2)..(i+3)]),16)
                        break
                }
        }
        def rawVolts = rawValue / 1000
        def minVolts = voltsmin ? voltsmin : 2.9
        def maxVolts = voltsmax ? voltsmax : 3.05
        def pct = (rawVolts - minVolts) / (maxVolts - minVolts)
        def roundedPct = Math.min(100, Math.round(pct * 100))
        def descText = "Battery level is ${roundedPct}% (${rawVolts} Volts)"
        // lastCheckinEpoch is for apps that can use Epoch time/date and lastCheckinTime
can be used with Hubitat Dashboard
        if (lastCheckinEnable) {
                sendEvent(name: "lastCheckinEpoch", value: now())
                sendEvent(name: "lastCheckinTime", value: new Date().toLocaleString())
        }
        def result = [
                name: 'battery',
                value: roundedPct,
                unit: "%",
                descriptionText: descText
        ]
        return result
}
private def displayDebugLog(message) {
        if (debugLogging)
                log.debug "${device.displayName}: ${message}"
}
private def displayInfoLog(message) {
        if (infoLogging || state.prefsSetCount != 1)
                log.info "${device.displayName}: ${message}"
}
//Reset the batteryLastReplaced date to current date
def resetBatteryReplacedDate(paired) {
        def newlyPaired = paired ? " for newly paired sensor" : ""
        sendEvent(name: "batteryLastReplaced", value: new Date())
        displayInfoLog("Setting Battery Last Replaced to current date${newlyPaired}")
}
// installed() runs just after a device is paired
def installed() {
        state.prefsSetCount = 0
        displayInfoLog("Installing")
}
// configure() runs after installed() when a device is paired or reconnected
def configure() {
        displayInfoLog("Configuring")
        if (!device.currentState('batteryLastReplaced')?.value)
                resetBatteryReplacedDate(true)
        setNumButtons()
        state.prefsSetCount = 1
        return
}
// updated() runs every time user saves preferences
def updated() {
        displayInfoLog("Updating preference settings")
        if (!device.currentState('batteryLastReplaced')?.value)
                resetBatteryReplacedDate(true)
        setNumButtons()
        displayInfoLog("Info message logging enabled")
        displayDebugLog("Debug message logging enabled")
}
// Set number of buttons available to Apps based on user setting
def setNumButtons() {
        def numButtons = [7, 36, 43]
        def cubeModeInt = (settings.cubeMode) ? settings.cubeMode as Integer : null
        if (settings.cubeMode && (state.cubeMode != cubeModeInt)) {
                sendEvent(name: "numberOfButtons", value: numButtons[cubeModeInt])
                state.cubeMode = cubeModeInt
                displayInfoLog("Number of buttons set to ${numButtons[cubeModeInt]}")
        } else if (state.cubeMode ==  null || (settings.cubeMode == null &&
state.cubeMode != 0)) {
                sendEvent(name: "numberOfButtons", value: 7)
                state.cubeMode = 0
                displayInfoLog("Number of buttons set to default of 7")
        }
}
// This section is functions used for driver commands
def setFace0() {
        setFace(0)
}
def setFace1() {
        setFace(1)
}
def setFace2() {
        setFace(2)
}
def setFace3() {
        setFace(3)
}
def setFace4() {
        setFace(4)
}
def setFace5() {
        setFace(5)
}
def flip90() {
        def flipMap = [0:5, 1:2, 2:0, 3:2, 4:5, 5:3]
        flipEvents(flipMap[device.currentValue("face") as Integer], "90")
}
def flip180() {
        def flipMap = [0:3, 1:4, 2:5, 3:0, 4:1, 5:2]
        flipEvents(flipMap[device.currentValue("face") as Integer], "180")
}
def rotateL() {
        rotateEvents(-90)
}
def rotateR() {
        rotateEvents(90)
```

```
}
def slide() {
        slideEvents(device.currentValue("face") as Integer)
}
def knock() {
        knockEvents(device.currentValue("face") as Integer)
}
def shake() {
        shakeEvents()
}
```

4.5 Xiaomi Aqara Temperature and Humidity Sensor

https://www.amazon.com/Aqara-WSDCGQ11LM-Temperature-Humidity-Sensor/dp/B07D37FKGY



Hubitat | Hubitat

□	Back Porch Temp Sensor (4221)		
□	TemperatureMeasurement (4222)	83	Today 7:56:2
□	RelativeHumidityMeasurement (4223)	63 %	Today 7:56:2
□	Battery (4224)	78	Today 7:56:2
□	AtmosphericPressure (4238)	1012 mb	Today 7:56:2

Figure 21 Aqara Temperature and Humidty HS Devices

4.5.1 Driver

https://raw.githubusercontent.com/veeceeoh/xiaomi-hubitat/master/devicedrivers/xiaomi-temperature-humidity-sensor-hubitat.src/xiaomi-temperature-humidity-sensor-hubitat.groovy

```
/**
 * Xiaomi "Original" Temperature Humidity Sensor - model RTCGQ01LM
 * & Aqara Temperature Humidity Sensor - model WSDCGQ11LM
 * Device Driver for Hubitat Elevation hub
 * Version 1.0.1
 *
 *
 * Licensed under the Apache License, Version 2.0 (the "License"); you
may not use this file except
 * in compliance with the License. You may obtain a copy of the License
at:
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,
either express or implied. See the License
 * for the specific language governing permissions and limitations under
the License.
 *
 * Based on SmartThings device handler code by a4refillpad
 * With contributions by alecm, alixjg, bspranger, gn0st1c, foz333,
guyeeba, jmagnuson, mike.maxwell, rinkek, ronvandegraaf, snalee,
tmleafs, twonk, & veeceeoh
 * Code reworked for use with Hubitat Elevation hub by veeceeoh
 *
 * Known issues:
 * + Xiaomi devices send reports based on changes, and a status report
every 50-60 minutes. These settings cannot be adjusted.
 * + The battery level / voltage is not reported at pairing. Wait for
the first status report, 50-60 minutes after pairing.
 * + Pairing Xiaomi devices can be difficult as they were not designed
```

```
to use with a Hubitat hub.
 *   Holding the sensor's reset button until the LED blinks will start
pairing mode.
 *   3 quick flashes indicates success, while one long flash means
pairing has not started yet.
 *   In either case, keep the sensor "awake" by short-pressing the reset
button repeatedly, until recognized by Hubitat.
 * + The connection can be dropped without warning. To reconnect, put
Hubitat in "Discover Devices" mode, then short-press
 *   the sensor's reset button, and wait for the LED - 3 quick flashes
indicates reconnection. Otherwise, short-press again.
 *
 */

metadata {
        definition (name: "Xiaomi Temperature Humidity Sensor",
namespace: "veeceeoh", author: "veeceeoh", importUrl:
"https://raw.githubusercontent.com/veeceeoh/xiaomi-
hubitat/master/devicedrivers/xiaomi-temperature-humidity-sensor-
hubitat.src/xiaomi-temperature-humidity-sensor-hubitat.groovy") {
                capability "Battery"
                capability "PressureMeasurement"
                capability "RelativeHumidityMeasurement"
                capability "Sensor"
                capability "TemperatureMeasurement"

                command "resetBatteryReplacedDate"

                attribute "lastCheckinEpoch", "String"
                attribute "lastCheckinTime", "Date"
                attribute "batteryLastReplaced", "String"

                //fingerprint for Xioami "original" Temperature Humidity
Sensor - model RTCGQ01LM
                fingerprint profileId: "0104", inClusters:
"0000,0003,0019,FFFF,0012", outClusters:
"0000,0004,0003,0005,0019,FFFF,0012", model: "lumi.sens"
                fingerprint profileId: "0104", inClusters:
"0000,0003,0019,FFFF,0012", outClusters:
"0000,0004,0003,0005,0019,FFFF,0012", model: "lumi.sensor_ht"
                //fingerprint for Xioami Aqara Temperature Humidity
Sensor - model WSDCGQ11LM
                fingerprint profileId: "0104", inClusters:
"0000,0003,FFFF,0402,0403,0405", outClusters: "0000,0004,FFFF", model:
"lumi.weather"
        }

        preferences {
                //Temp and Humidity Offsets
                input "tempOffset", "decimal", title:"Temperature
Offset", description:"", range:"*..*"
                input "humidityOffset", "decimal", title:"Humidity
Offset", description:"", range: "*..*"
                if (getDataValue("modelType") == "Aqara WSDCGQ11LM" ||
getDataValue("modelType") == "unknown") {
                        input name: "pressOffset", type: "decimal",
title: "Pressure Offset", description: "", range: "*..*"
```

```
                        input name: "pressureUnits", type: "enum", title:
"Pressure Units (default: mbar)", description: "", options: ["mbar",
"kPa", "inHg", "mmHg"], default: "mbar"
                }
                //Battery Voltage Range
                input name: "voltsmin", type: "decimal", title: "Min
Volts (0% battery = ___ volts). Default = 2.8 Volts", description: ""
                input name: "voltsmax", type: "decimal", title: "Max
Volts (100% battery = ___ volts). Default = 3.05 Volts", description: ""
                //Date/Time Stamp Events Config
                input name: "lastCheckinEnable", type: "bool", title:
"Enable custom date/time stamp events for lastCheckin", description: ""
                //Event Generation Behavior Config
                input name: "recordAllReadings", type: "bool", title:
"Enable events for unchanged sensor readings between hourly check-in",
description: ""
                input name: "recordAllBattery", type: "bool", title:
"Enable events for unchanged battery level", description: ""
                //Logging Message Config
                input name: "infoLogging", type: "bool", title: "Enable
info message logging", description: ""
                input name: "debugLogging", type: "bool", title: "Enable
debug message logging", description: ""
        }
}

// Parse incoming device messages to generate events
def parse(String description) {
        displayDebugLog("Parsing message: $description")
        if (description?.startsWith('cat')) {
                Map descMap = zigbee.parseDescriptionAsMap(description)
                displayDebugLog("Zigbee parse map of catchall =
$descMap")
                displayDebugLog("No action taken on catchall message")
        } else if (description?.startsWith('re')) {
                description = description - "read attr - "
                Map descMap = (description).split(",").inject([:]) {
                        map, param ->
                        def nameAndValue = param.split(":")
                        map +=
[(nameAndValue[0].trim()):nameAndValue[1].trim()]
                }
                // Reverse payload byte order for little-endian data
types - required for Hubitat firmware 2.0.5 or newer
                def intEncoding = Integer.parseInt(descMap.encoding, 16)
                if (descMap.value != null && intEncoding > 0x18 &&
intEncoding < 0x3e) {
                        descMap.value = reverseHexString(descMap.value)
                        displayDebugLog("Little-endian payload data type;
Hex value reversed to: ${descMap.value}")
                }
        // Send message data to appropriate parsing function based on
the type of report
                switch (descMap.cluster) {
                        case "0000": // Announcement or Check-in report
                                if (descMap.attrId == "0005")
                                        displayDebugLog("Reset button was
short-pressed")
                                else if (descMap.attrId == "FF01" ||
descMap.attrId == "FF02") // Hourly check-in report

        parseCheckinMessage(descMap.value)
                                break
                        case "0402": // Temperature report
                                parseTemperature(descMap.value, false)
                                break
                        case "0403": // Pressure report (Aqara model
only)

        parsePressure(Integer.parseInt(descMap.value[0..3], 16)*10,
false)
                                break
                        case "0405": // Humidity report
                                parseHumidity(descMap.value, false)
                                break
                        default:
                                displayDebugLog("Unknown read attribute
message")
                }
        }
        return [:]
}

// Reverses order of bytes in hex string
def reverseHexString(hexString) {
        def reversed = ""
        for (int i = hexString.length(); i > 0; i -= 2) {
                reversed += hexString.substring(i - 2, i )
        }
        return reversed
}

// Calculate temperature with 0.01 precision in C or F unit as set by
hub location settings
private parseTemperature(hexString, checkin) {
        float temp = hexStrToSignedInt(hexString)/100
        def tempScale = location.temperatureScale
        def debugText = "Reported temperature: raw = $temp°C"
        if (temp < -50) {
                log.warn "${device.displayName}: Out-of-bounds
temperature value received. Battery voltage may be too low."
        } else {
                if (tempScale == "F") {
                        temp = ((temp * 1.8) + 32)
                        debugText += ", converted = $temp°F"
                }
                if (tempOffset) {
                        temp = (temp + tempOffset)
                        debugText += ", offset = $tempOffset"
                }
                displayDebugLog(debugText)
                temp = temp.round(2)
                generateEvent([
                        name: 'temperature',
                        value: temp,
                        unit: "°$tempScale",
                        descriptionText: "Temperature is 
$temp°$tempScale",
                        translatable:true
                        ], checkin)
        }
}

// Calculate humidity with 0.1 precision
private parseHumidity(hexString, checkin) {
        float humidity = Integer.parseInt(hexString,16)/100
        def debugText = "Reported humidity: raw = $humidity"
        if (humidity > 100) {
                log.warn "${device.displayName}: Out-of-bounds humidity 
value received. Battery voltage may be too low."
                return ""
        } else {
                if (humidityOffset) {
                        debugText += ", offset = $humidityOffset"
                        humidity = (humidity + humidityOffset)
                }
                displayDebugLog(debugText)
                humidity = humidity.round(1)
                generateEvent([
                        name: 'humidity',
                        value: humidity,
                        unit: "%",
                        descriptionText: "Humidity is $humidity%"
                        ], checkin)
        }
}

// Parse pressure report
private parsePressure(float pressureval, checkin) {
        def debugText = "Reported pressure: raw = $pressureval Pa"
        if (!(pressureUnits)) {
                pressureUnits = "mbar"
        }
        switch (pressureUnits) {
                case "mbar":
                        pressureval = (pressureval/100) as Float
                        pressureval = pressureval.round(1);
                        break;
                case "kPa":
                        pressureval = (pressureval/1000) as Float
                        pressureval = pressureval.round(2);
                        break;
                case "inHg":
                        pressureval = (((pressureval/100) as Float) * 
0.0295300)
                        pressureval = pressureval.round(2);
                        break;
                case "mmHg":
                        pressureval = (((pressureval/100) as Float) * 
0.750062)
                        pressureval = pressureval.round(2);
                        break;
        }
        debugText += ", converted = $pressureval $pressureUnits"
        if (pressOffset) {
                debugText += ", offset = $pressOffset"
                pressureval = (pressureval + pressOffset)
        }
        displayDebugLog(debugText)
        pressureval = pressureval.round(2)
        generateEvent([
                name: 'pressure',
                value: pressureval,
                unit: pressureUnits,
                descriptionText: "Pressure is $pressureval
$pressureUnits"
                ], checkin)
}

// Convert raw 4 digit integer voltage value into percentage based on
minVolts/maxVolts range
private parseBattery(hexString) {
        def rawVolts = Integer.parseInt(hexString,16) / 1000
        def minVolts = voltsmin ? voltsmin : 2.8
        def maxVolts = voltsmax ? voltsmax : 3.05
        displayDebugLog("Battery report: $rawVolts Volts, calculating
level based on min/max range of $minVolts to $maxVolts")
        def pct = (rawVolts - minVolts) / (maxVolts - minVolts)
        def roundedPct = Math.min(100, Math.round(pct * 100))
        generateEvent([
                name: 'battery',
                value: roundedPct,
                unit: "%",
                descriptionText: "Battery level is $roundedPct%
($rawVolts Volts)"
                ], false)
}

def generateEvent(eventResult, checkin) {
        eventResult.descriptionText += checkin ? " (check-in report)" :
""
        eventResult = (recordAllReadings || checkin ||
((eventResult.name == battery) && recordAllBattery)) ? (eventResult +
[isStateChange: true]) : eventResult
        displayDebugLog("Creating event $eventResult")
        sendEvent(eventResult)
        displayInfoLog(eventResult.descriptionText)
}

def parseCheckinMessage(hexString) {
        if (lastCheckinEnable) {
        // lastCheckinEpoch is for apps that can use Epoch time/date and
lastCheckinTime can be used with Hubitat Dashboard
                sendEvent(name: "lastCheckinEpoch", value: now())
                sendEvent(name: "lastCheckinTime", value: new
Date().toLocaleString())
        }
        displayDebugLog("Received check-in message")
        def result
        // First byte of hexString is UINT8 of payload length in bytes,
so it is skipped
        def strPosition = 2
        def strLength = hexString.size() - 2
        while (strPosition < strLength) {
                def dataTag =
Integer.parseInt(hexString[strPosition++..strPosition++], 16)  // Each
attribute of the check-in message payload is preceded by a unique 1-byte
tag value
                def dataType =
Integer.parseInt(hexString[strPosition++..strPosition++], 16)  // After
each attribute tag, the following byte gives the data type of the
attribute data
                def dataLength = DataType.getLength(dataType)  // This
looks up the length of data for the determined data type
                def dataPayload  // This is used to collect the payload
data of each check-in message attribute
                if (dataLength == null || dataLength == -1 || dataLength
== 0) {  // A length of null or -1 means the data type is probably
variable-length, and 0 length is invalid
                        displayDebugLog("Check-in message contains
unsupported dataType 0x${Integer.toHexString(dataType)} for dataTag
0x${Integer.toHexString(dataTag)} with dataLength $dataLength")
                        return
                } else {
                        if (strPosition > (strLength - dataLength)) {
                                displayDebugLog("Ran out of data before
finishing parse of check-in message")
                                return
                        }
                        dataPayload =
hexString[strPosition++..(strPosition+=(dataLength * 2) - 1)-1]  //
Collect attribute tag payload according to data length of its data type
                        dataPayload = reverseHexString(dataPayload)  //
Reverse order of bytes for big endian payload
                        def dataDebug1 = "Check-in message: Found dataTag
0x${Integer.toHexString(dataTag)}"
                        def dataDebug2 = "dataType
0x${Integer.toHexString(dataType)}, dataLength $dataLength, dataPayload
$dataPayload"
                        switch (dataTag) {
                                case 0x01:  // Battery voltage
                                        displayDebugLog("$dataDebug1
(battery), $dataDebug2")
                                        parseBattery(dataPayload)
                                        break
                                case 0x05:  // RSSI dB
                                        def convertedPayload =
Integer.parseInt(dataPayload,16)
                                        displayDebugLog("$dataDebug1
(RSSI dB), $dataDebug2 ($convertedPayload)")
                                        state.RSSI = convertedPayload
                                        break
                                case 0x06:  // LQI
                                        def convertedPayload =
Integer.parseInt(dataPayload,16)
```

```
                                        displayDebugLog("$dataDebug1
(LQI), $dataDebug2 ($convertedPayload)")
                                        state.LQI = convertedPayload
                                        break
                                case 0x64:  // Temperature in Celcius
                                        displayDebugLog("$dataDebug1
(temperature), $dataDebug2")
                                        parseTemperature(dataPayload,
true)
                                        break
                                case 0x65:  // Relative humidity
                                        displayDebugLog("$dataDebug1
(humidity), $dataDebug2")
                                        parseHumidity(dataPayload, true)
                                        break
                                case 0x66:  // Atmospheric pressure
                                        displayDebugLog("$dataDebug1
(pressure), $dataDebug2")

        parsePressure(Integer.parseInt(dataPayload,16), true)
                                        break
                                case 0x0A:  // ZigBee parent DNI (device
network identifier)
                                        displayDebugLog("$dataDebug1
(ZigBee parent DNI), $dataDebug2")
                                        state.zigbeeParentDNI =
dataPayload
                                        break
                                default:
                                        displayDebugLog("$dataDebug1
(unknown), $dataDebug2")
                        }
                }
        }
}

// installed() runs just after a sensor is paired
def installed() {
        displayDebugLog("Installing")
        init()
}

// configure() runs after installed() when a sensor is paired or re-
joined
def configure() {
        displayInfoLog("Configuring")
        init()
        device.updateSetting("infoLogging",[value:"true",type:"bool"])
        device.updateSetting("debugLogging",[value:"true",type:"bool"])
        displayInfoLog("Info and debug message logging automatically
enabled for 2 hours. This can be overidden by saving preferences for
this device.")
        runIn(7200, stopAutoLogging, [data: [stopLogging: true]])
}

// updated() will run every time user saves preferences
def updated() {
        displayInfoLog("Updating preference settings")
        init()
        displayInfoLog("Info message logging enabled")
        displayDebugLog("Debug message logging enabled")
        runIn(2, stopAutoLogging, [data: [stopLogging: false]])
}

def init() {
        if (!(getDataValue("modelType")) || getDataValue("modelType") ==
"unknown") {
                def sensorModel = "unknown"
                if (device.data.model != "") {
                        if (device.data.model[5] == "s")  // for model:
"lumi.sensor_ht" or "lumi.sens"
                                sensorModel = "Xiaomi RTCGQ01LM"
                        else if (device.data.model[5] == "w")  // for
model: "lumi.weather"
                                sensorModel = "Aqara WSDCGQ11LM"
                }
                updateDataValue("modelType", sensorModel)
                displayInfoLog("Detected sensor model is
${sensorModel}")
        }
        if (!device.currentValue('batteryLastReplaced'))
                resetBatteryReplacedDate(true)
}

def stopAutoLogging(data) {
        if (data.stopLogging) {
                displayInfoLog("Automatic info and debug message logging
now finished")

        device.updateSetting("infoLogging",[value:"false",type:"bool"])

        device.updateSetting("debugLogging",[value:"false",type:"bool"])
        }
}

//Reset the batteryLastReplaced date to current date
def resetBatteryReplacedDate(paired) {
        def newlyPaired = paired ? " for newly paired sensor" : ""
        sendEvent(name: "batteryLastReplaced", value: new
Date().format("MMM dd yyyy", location.timeZone))
        displayInfoLog("Setting Battery Last Replaced to current
date${newlyPaired}")
}

private def displayDebugLog(message) {
        if (debugLogging) log.debug "${device.displayName}: ${message}"
}

private def displayInfoLog(message) {
        if (infoLogging || state.prefsSetCount != 1)
                log.info "${device.displayName}: ${message}"
}
```

4.6 Echo Speaks

Echo Speaks is a Hubitat (and Smartthings) App with associated drivers that provides integration of Hubitat Elevation with Amazon Echo. There is much exposed via Maker API for Echo Speaks. mcsHubitat has selected a subset that seems practical for integration with HS.

The primary focus is ability for Homeseer to use the Echo speakers as output devices. The second capability that was integrated is to provide the status of what is currently playing through the Echo speaker. These are illustrated in Figure 22.

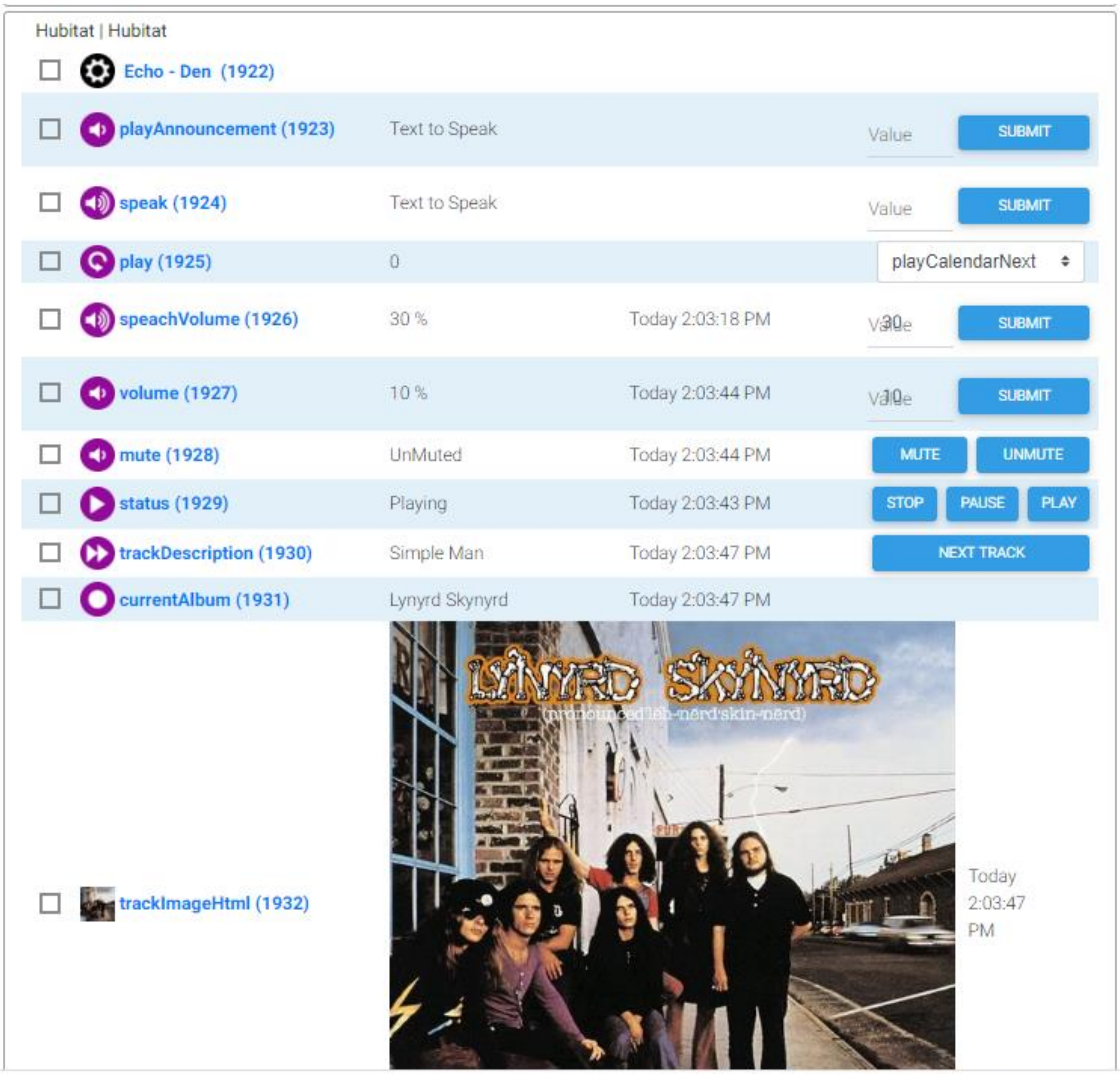
Two options exist for text to speech. One is the playAnnouncement device. Whatever is stored in the Device/Feature String will be verbalized on the associated Echo speaker. It will be preceded with a ding to provide warning.

The second is with the combination of speachVolume and speak Devices/Features. The volume will first be changed to the Value in speachVolume and then any changes in the speak String will be spoken at the temporary speachVolume level. The volume will then be restored to prior level that is shown in the Volume Device/Feature.

A similar volume change and restore is done when using the play device. A selector is used to identify the desired pre-canned type of information such as weather, traffic, calendar, or a joke.

A mute control is also provided.

When using Echo as a music player the information about what is being played in the track, album and cover art devices. Play/Pause/Stop controls are also provided.



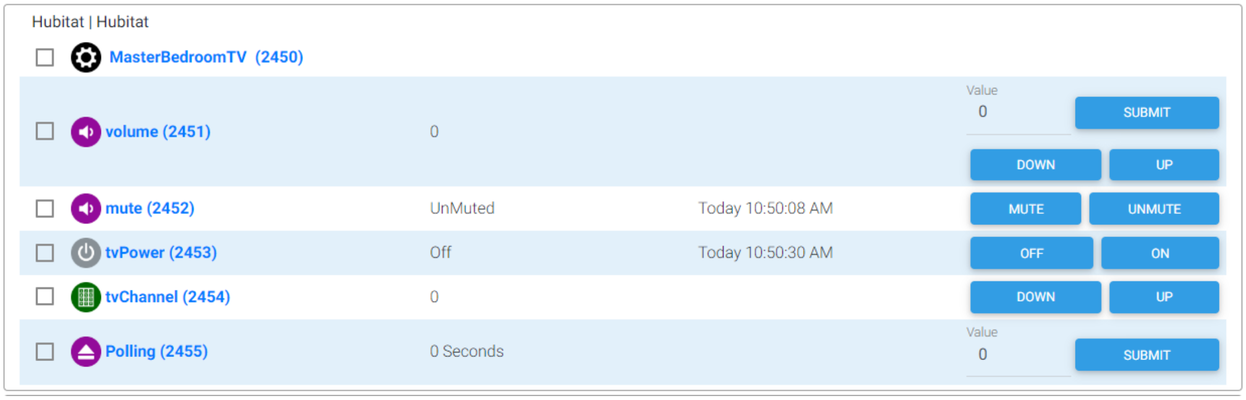
Hubitat | Hubitat

□ Echo - Den (1922)			
□ playAnnouncement (1923)	Text to Speak		Value SUBMIT
□ speak (1924)	Text to Speak		Value SUBMIT
□ play (1925)	0		playCalendarNext
□ speachVolume (1926)	30 %	Today 2:03:18 PM	30 SUBMIT
□ volume (1927)	10 %	Today 2:03:44 PM	10 SUBMIT
□ mute (1928)	UnMuted	Today 2:03:44 PM	MUTE UNMUTE
□ status (1929)	Playing	Today 2:03:43 PM	STOP PAUSE PLAY
□ trackDescription (1930)	Simple Man	Today 2:03:47 PM	NEXT TRACK
□ currentAlbum (1931)	Lynyrd Skynyrd	Today 2:03:47 PM	
□ trackImageHtml (1932)		Today 2:03:47 PM	

Figure 22 Echo Speaks HS Devices

4.7 RokuTV

The community driver for RokuTV can be found at https://github.com/apwelsh/hubitat



Hubitat | Hubitat

□ MasterBedroomTV (2450)

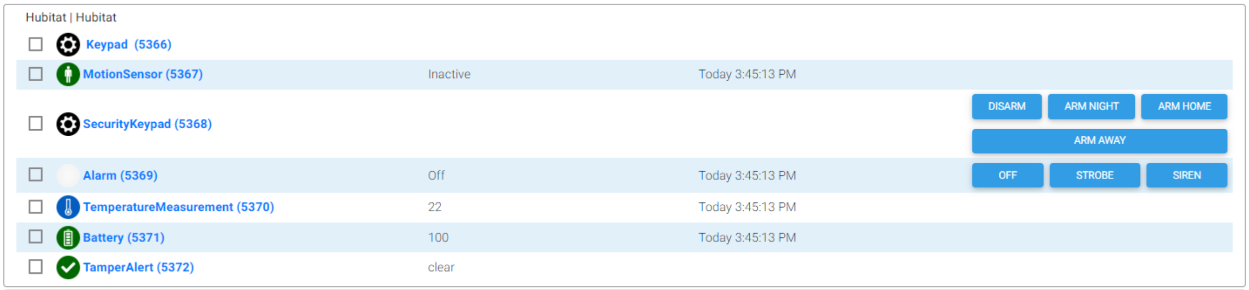
□ volume (2451)	0		Value 0
□ mute (2452)	UnMuted	Today 10:50:08 AM	
□ tvPower (2453)	Off	Today 10:50:30 AM	
□ tvChannel (2454)	0		
□ Polling (2455)	0 Seconds		Value 0

Figure 23 Roku TV Devices

4.8 Centralite, IRIS, Xfinity Security Keypad

Some models of the keypad contain Alarm output and others do not. The Alarm is created in HS as a feature of the keypad device, but may not be functional depending upon the model of the keypad being used.

Lock codes are also made available by MakerAPI, but these are not made visible in HS.



Hubitat | Hubitat

□ Keypad (5366)			
□ MotionSensor (5367)	Inactive	Today 3:45:13 PM	
□ SecurityKeypad (5368)			DISARM ARM NIGHT ARM HOME ARM AWAY
□ Alarm (5369)	Off	Today 3:45:13 PM	OFF STROBE SIREN
□ TemperatureMeasurement (5370)	22	Today 3:45:13 PM	
□ Battery (5371)	100	Today 3:45:13 PM	
□ TamperAlert (5372)	clear		

Figure 24 Security Keypad Devices

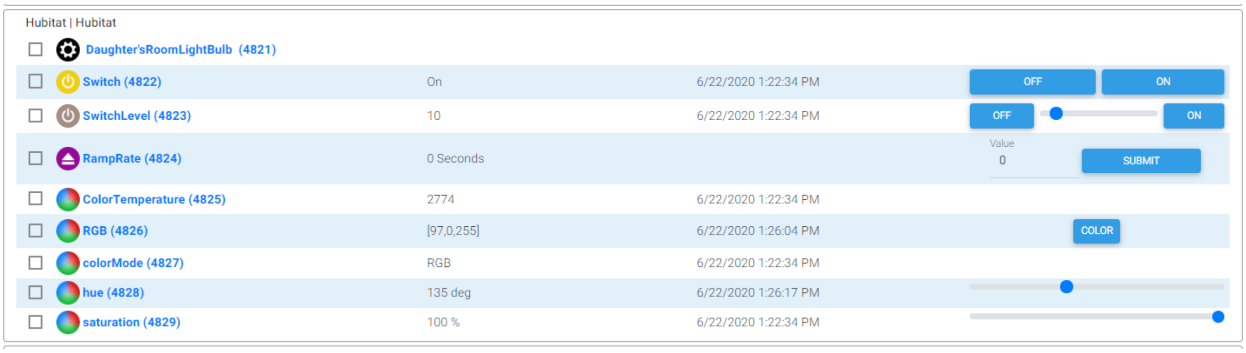
4.9 Inovelli Bulb Multi-Color LZW42

Driver is available at https://raw.githubusercontent.com/In...r-lzw42.groovy

MakerAPI as of June 24, 2020 provides a HSB interface with this color bulb. mcsHubitat uses and HS Color picker for color selection. HS provides a RGB value. This is converted by the plug-in to HSB JSON as expected by MakerAPI.

Separate controls, in the form of Sliders are also available to individually set the Hue, Saturation and Brightness Level.

mcsHubitat inserts a Transition Rate device which allows user to ramp the Brightness level over a specified interval.



Hubitat | Hubitat

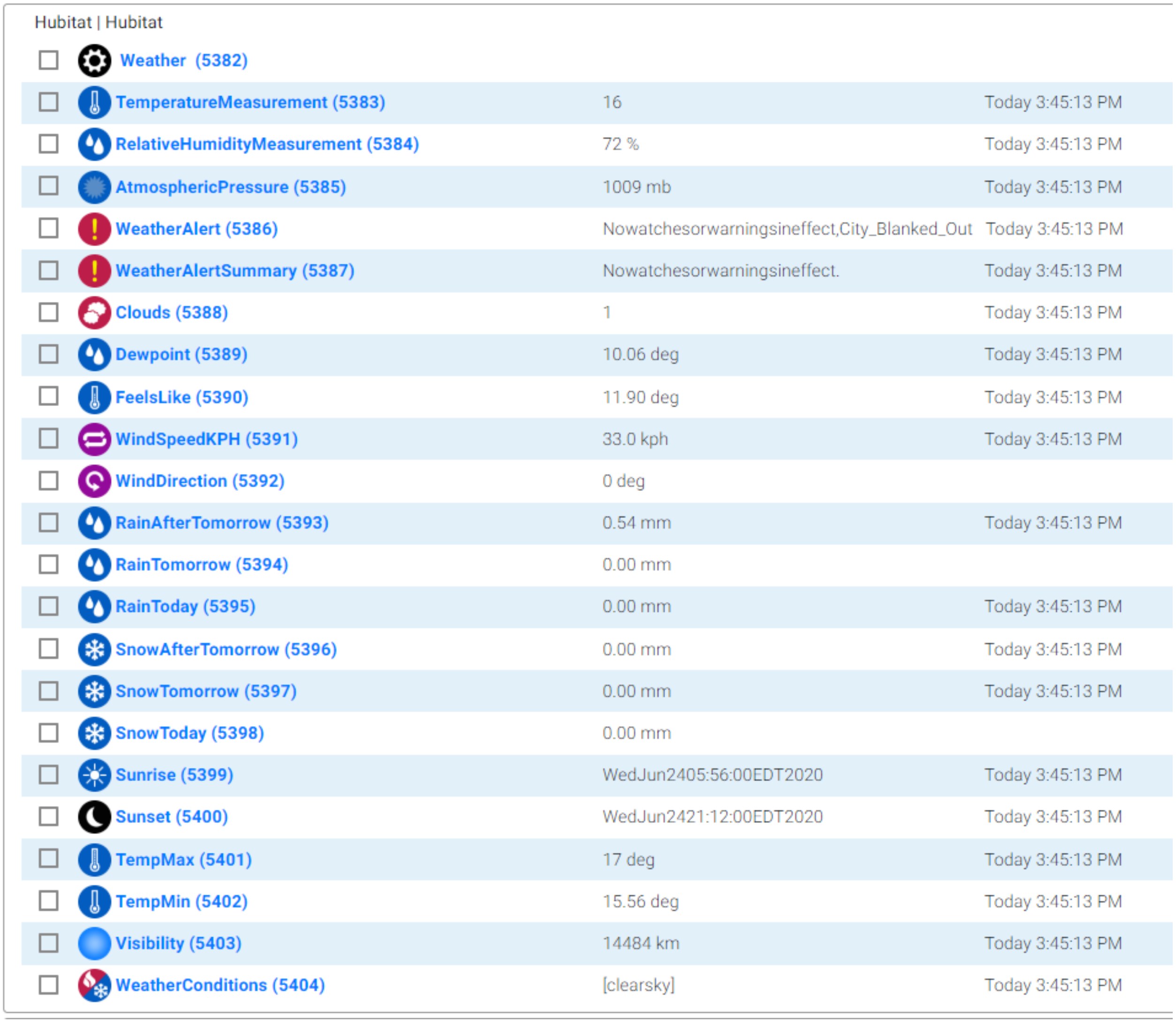
□	Daughter'sRoomLightBulb (4821)		
□	Switch (4822)	On	6/22/2020 1:22:34 PM
□	SwitchLevel (4823)	10	6/22/2020 1:22:34 PM
□	RampRate (4824)	0 Seconds	
□	ColorTemperature (4825)	2774	6/22/2020 1:22:34 PM
□	RGB (4826)	[97,0,255]	6/22/2020 1:26:04 PM
□	colorMode (4827)	RGB	6/22/2020 1:22:34 PM
□	hue (4828)	135 deg	6/22/2020 1:26:17 PM
□	saturation (4829)	100 %	6/22/2020 1:22:34 PM

Figure 25 Inovelli Bulb Multi-Color LZW42 Devices

4.10 Weather Canada (OWM-EC)

Driver developed by Mik3y on HomeSeer Message Board to pull data from Canada Enviromental. Driver at https://raw.githubusercontent.com/dm...OWM-EC).groovy

Most attributes from this driver have been exposed as HS devices. When numeric data is available it is placed in HS DeviceValue to facilitate event triggers. Other data is placed in DeviceString.



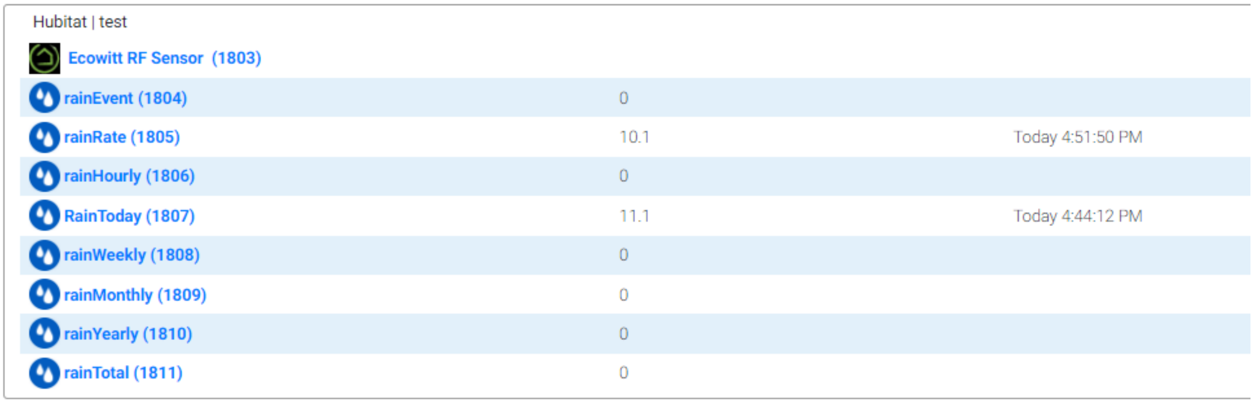
Hubitat | Hubitat

□	Weather (5382)		
□	TemperatureMeasurement (5383)	16	Today 3:45:13 PM
□	RelativeHumidityMeasurement (5384)	72 %	Today 3:45:13 PM
□	AtmosphericPressure (5385)	1009 mb	Today 3:45:13 PM
□	WeatherAlert (5386)	Nowatchesorwarningsineffect,City_Blanked_Out	Today 3:45:13 PM
□	WeatherAlertSummary (5387)	Nowatchesorwarningsineffect.	Today 3:45:13 PM
□	Clouds (5388)	1	Today 3:45:13 PM
□	Dewpoint (5389)	10.06 deg	Today 3:45:13 PM
□	FeelsLike (5390)	11.90 deg	Today 3:45:13 PM
□	WindSpeedKPH (5391)	33.0 kph	Today 3:45:13 PM
□	WindDirection (5392)	0 deg	
□	RainAfterTomorrow (5393)	0.54 mm	Today 3:45:13 PM
□	RainTomorrow (5394)	0.00 mm	
□	RainToday (5395)	0.00 mm	Today 3:45:13 PM
□	SnowAfterTomorrow (5396)	0.00 mm	Today 3:45:13 PM
□	SnowTomorrow (5397)	0.00 mm	Today 3:45:13 PM
□	SnowToday (5398)	0.00 mm	
□	Sunrise (5399)	WedJun2405:56:00EDT2020	Today 3:45:13 PM
□	Sunset (5400)	WedJun2421:12:00EDT2020	Today 3:45:13 PM
□	TempMax (5401)	17 deg	Today 3:45:13 PM
□	TempMin (5402)	15.56 deg	Today 3:45:13 PM
□	Visibility (5403)	14484 km	Today 3:45:13 PM
□	WeatherConditions (5404)	[clearsky]	Today 3:45:13 PM

Figure 26 Weather Canada (OWM-EC) Devices

4.11 Ecowitt RF Sensor

The Ecowitt RF Sensor interfaces multiple sensors. Special provisions are made for the water sensor that provides various forms of data related to the rain sensing. These are show in Figure 27.



Hubitat | test

Ecowitt RF Sensor (1803)

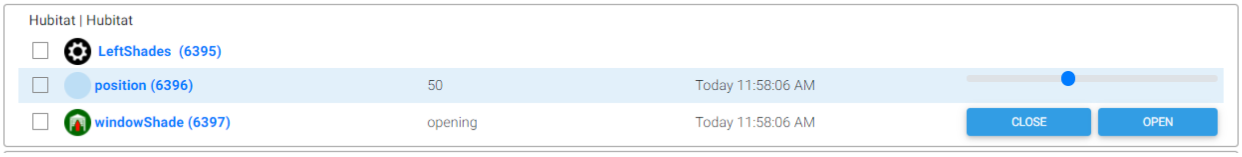
rainEvent (1804)	0	
rainRate (1805)	10.1	Today 4:51:50 PM
rainHourly (1806)	0	
RainToday (1807)	11.1	Today 4:44:12 PM
rainWeekly (1808)	0	
rainMonthly (1809)	0	
rainYearly (1810)	0	
rainTotal (1811)	0	

Figure 27 Ecowitt Rain Sensor Devices

4.12 Window Shades

Two integrations of Window Shades have been provided. The Bond controller interface provides position control and open/close/stop control with status feedback. One HS Device and two Features are used to support the interface as shown in Figure 28.

The iBlinds integration is similar to the Bond integration, but no stop control is provided. In the later case the "level" attribute is provided by Hubitat for the position while in the former the "position" attribute is provided. mcsHubitat normalizes this reporting to "position".



Hubitat | Hubitat

☐	LeftShades (6395)			
☐	position (6396)	50	Today 11:58:06 AM	
☐	windowShade (6397)	opening	Today 11:58:06 AM	CLOSE OPEN

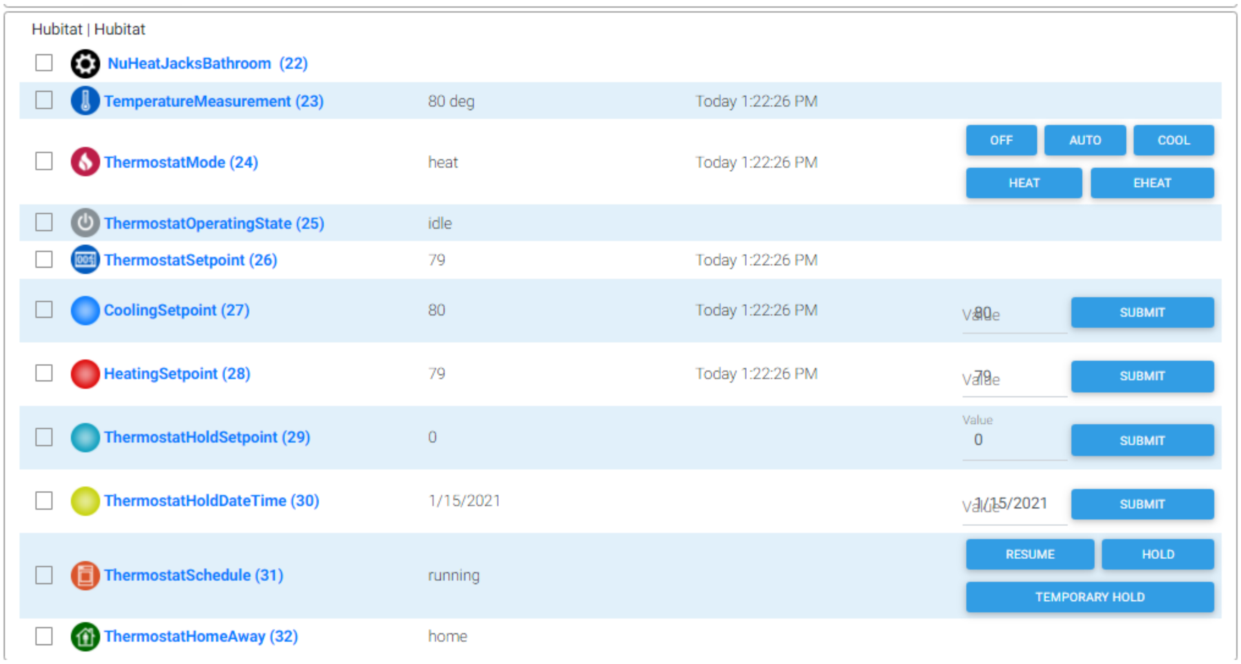
Figure 28 Window Shades Devices

4.13 NuHeat Thermostat

NuHeat is a variant implementation of a generic thermostat. It provides the additional capability of Home/Away status and means to manage a thermostat hold function. The last four features shown in Figure 29 support these additional capabilities.

The Hold function has two single button controls on the ThermostatSchedule features to resume and start the indefinite hold. The third button does a temporary hold of the schedule and establishes a new temporary temperature setPoint. The Hold setPoint and the duration of the temporary state are two additional HS features that need to be setup prior to use of the Temporary Hold button.

The TemporaryHoldDateTime feature accepts two input formats. One is MM/DD/YYYY which indicate the day when the Hold function is terminated. The other is a HHH format where then number of hours that the hold will be active.

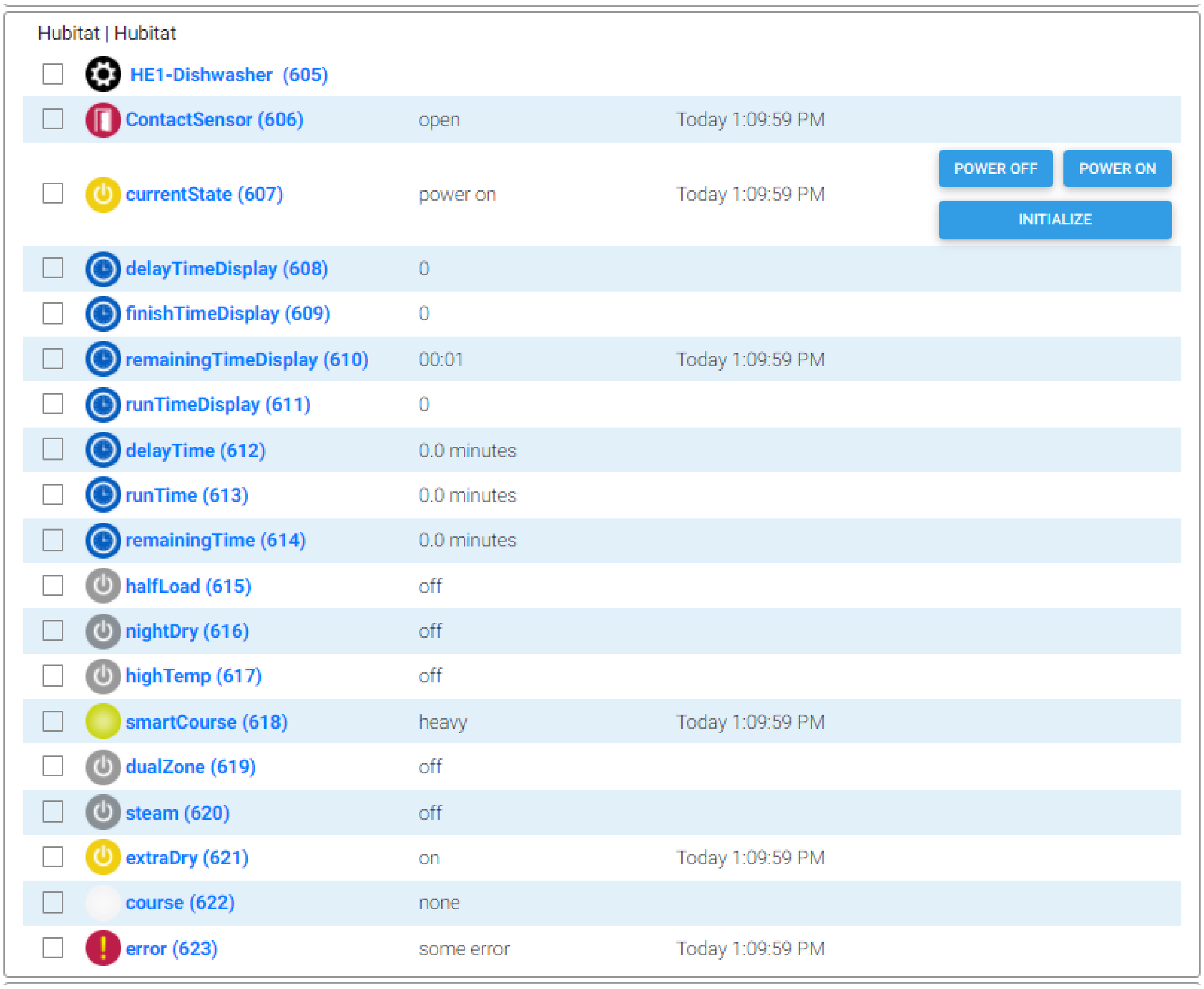


Hubitat | Hubitat

□	NuHeatJacksBathroom (22)			
□	TemperatureMeasurement (23)	80 deg	Today 1:22:26 PM	
□	ThermostatMode (24)	heat	Today 1:22:26 PM	OFF AUTO COOL HEAT EHEAT
□	ThermostatOperatingState (25)	idle		
□	ThermostatSetpoint (26)	79	Today 1:22:26 PM	
□	CoolingSetpoint (27)	80	Today 1:22:26 PM	Value 80 SUBMIT
□	HeatingSetpoint (28)	79	Today 1:22:26 PM	Value 79 SUBMIT
□	ThermostatHoldSetpoint (29)	0		Value 0 SUBMIT
□	ThermostatHoldDateTime (30)	1/15/2021		Value 1/15/2021 SUBMIT
□	ThermostatSchedule (31)	running		RESUME HOLD TEMPORARY HOLD
□	ThermostatHomeAway (32)	home		

Figure 29 NuHeat Thermostat Devices

4.14 LG ThinQ Dishwasher

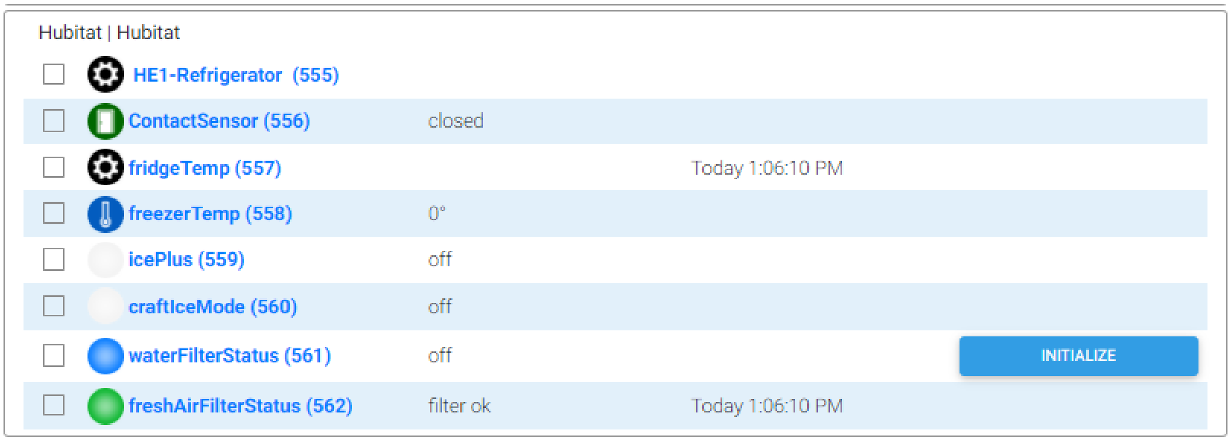


Hubitat | Hubitat

□	HE1-Dishwasher (605)		
□	ContactSensor (606)	open	Today 1:09:59 PM
□	currentState (607)	power on	Today 1:09:59 PM
□	delayTimeDisplay (608)	0	
□	finishTimeDisplay (609)	0	
□	remainingTimeDisplay (610)	00:01	Today 1:09:59 PM
□	runTimeDisplay (611)	0	
□	delayTime (612)	0.0 minutes	
□	runTime (613)	0.0 minutes	
□	remainingTime (614)	0.0 minutes	
□	halfLoad (615)	off	
□	nightDry (616)	off	
□	highTemp (617)	off	
□	smartCourse (618)	heavy	Today 1:09:59 PM
□	dualZone (619)	off	
□	steam (620)	off	
□	extraDry (621)	on	Today 1:09:59 PM
□	course (622)	none	
□	error (623)	some error	Today 1:09:59 PM

Figure 30 LG ThinQ Dishwasher HS Device and Features

4.15 LG ThinQ Fridge

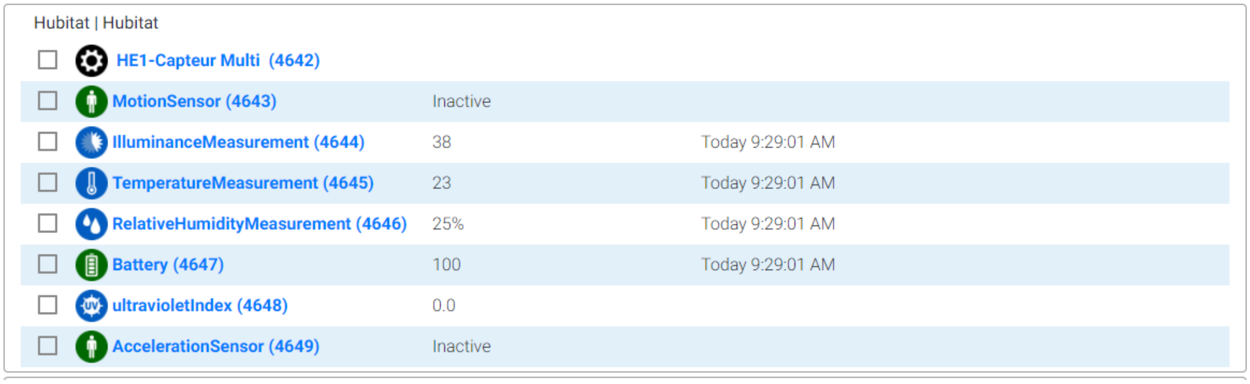


Hubitat | Hubitat

□	HE1-Refrigerator (555)			
□	ContactSensor (556)	closed		
□	fridgeTemp (557)		Today 1:06:10 PM	
□	freezerTemp (558)	0°		
□	icePlus (559)	off		
□	craftIceMode (560)	off		
□	waterFilterStatus (561)	off		INITIALIZE
□	freshAirFilterStatus (562)	filter ok	Today 1:06:10 PM	

Figure 31 LG ThinQ Fridge HS Device and Features

4.16 Aeon Multisensor 6

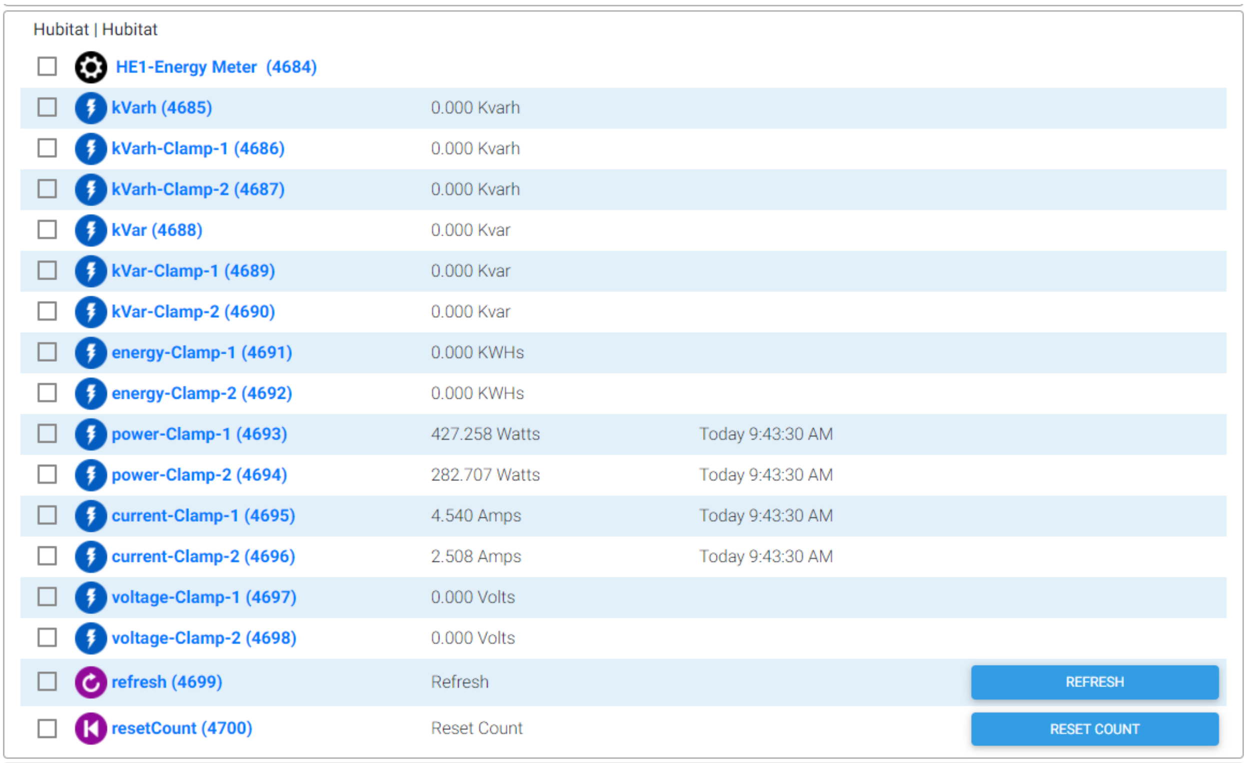


Hubitat | Hubitat

□	HE1-Capteur Multi (4642)		
□	MotionSensor (4643)	Inactive	
□	IlluminanceMeasurement (4644)	38	Today 9:29:01 AM
□	TemperatureMeasurement (4645)	23	Today 9:29:01 AM
□	RelativeHumidityMeasurement (4646)	25%	Today 9:29:01 AM
□	Battery (4647)	100	Today 9:29:01 AM
□	ultravioletIndex (4648)	0.0	
□	AccelerationSensor (4649)	Inactive	

Figure 32 Aeon Multisensor 6 HS Device and Features

4.17 Aeotec Home Energy Monitor (Gen 5)

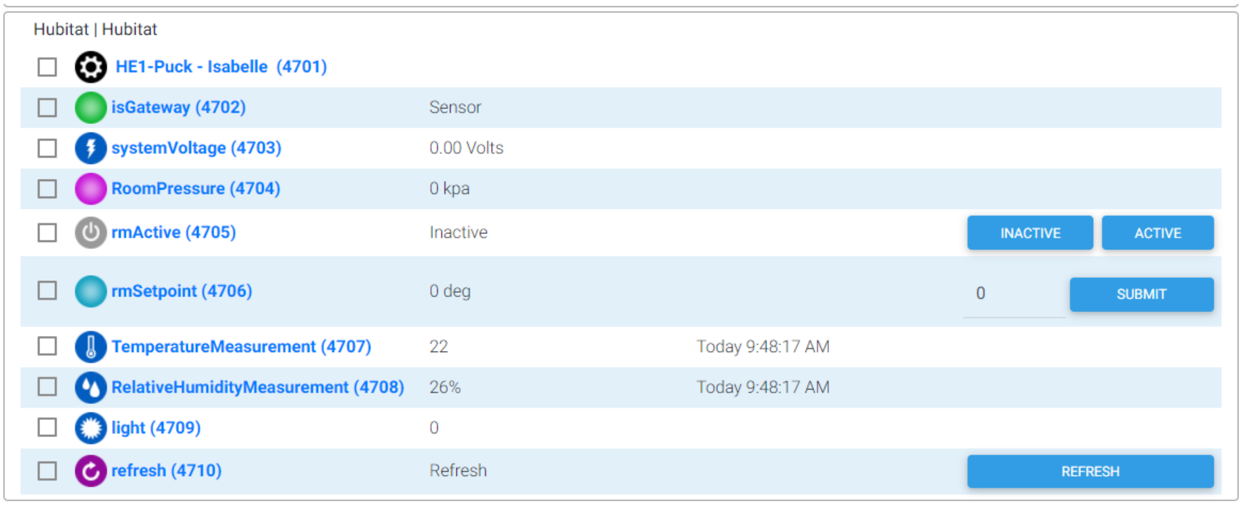


Hubitat | Hubitat

□	HE1-Energy Meter (4684)			
□	kVarh (4685)	0.000 Kvarh		
□	kVarh-Clamp-1 (4686)	0.000 Kvarh		
□	kVarh-Clamp-2 (4687)	0.000 Kvarh		
□	kVar (4688)	0.000 Kvar		
□	kVar-Clamp-1 (4689)	0.000 Kvar		
□	kVar-Clamp-2 (4690)	0.000 Kvar		
□	energy-Clamp-1 (4691)	0.000 KWHs		
□	energy-Clamp-2 (4692)	0.000 KWHs		
□	power-Clamp-1 (4693)	427.258 Watts	Today 9:43:30 AM	
□	power-Clamp-2 (4694)	282.707 Watts	Today 9:43:30 AM	
□	current-Clamp-1 (4695)	4.540 Amps	Today 9:43:30 AM	
□	current-Clamp-2 (4696)	2.508 Amps	Today 9:43:30 AM	
□	voltage-Clamp-1 (4697)	0.000 Volts		
□	voltage-Clamp-2 (4698)	0.000 Volts		
□	refresh (4699)	Refresh		REFRESH
□	resetCount (4700)	Reset Count		RESET COUNT

Figure 33 Aeotec Home Energy Monitor (Gen 5) HS Device and Features

4.18 My Puck Device

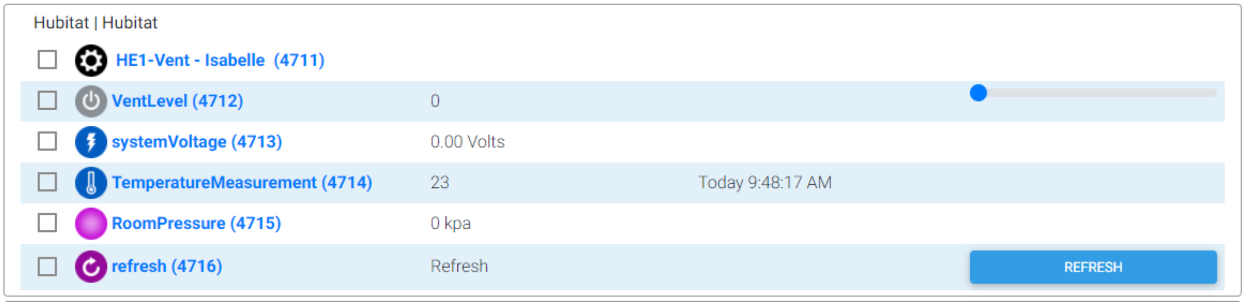


Hubitat | Hubitat

□	HE1-Puck - Isabelle (4701)			
□	isGateway (4702)	Sensor		
□	systemVoltage (4703)	0.00 Volts		
□	RoomPressure (4704)	0 kpa		
□	rmActive (4705)	Inactive		INACTIVE ACTIVE
□	rmSetpoint (4706)	0 deg		0 SUBMIT
□	TemperatureMeasurement (4707)	22	Today 9:48:17 AM	
□	RelativeHumidityMeasurement (4708)	26%	Today 9:48:17 AM	
□	light (4709)	0		
□	refresh (4710)	Refresh		REFRESH

Figure 34 My Puck Device HS Device and Features

4.19 My Flair Vent

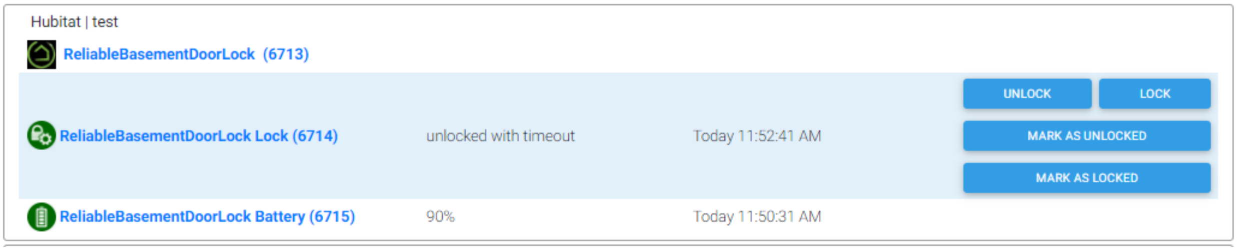


Hubitat | Hubitat

□	HE1-Vent - Isabelle (4711)			
□	VentLevel (4712)	0		
□	systemVoltage (4713)	0.00 Volts		
□	TemperatureMeasurement (4714)	23	Today 9:48:17 AM	
□	RoomPressure (4715)	0 kpa		
□	refresh (4716)	Refresh		REFRESH

Figure 35 My Flair Vent HS Device and Features

4.20 Reliable Lock Virtual Device



Hubitat | test

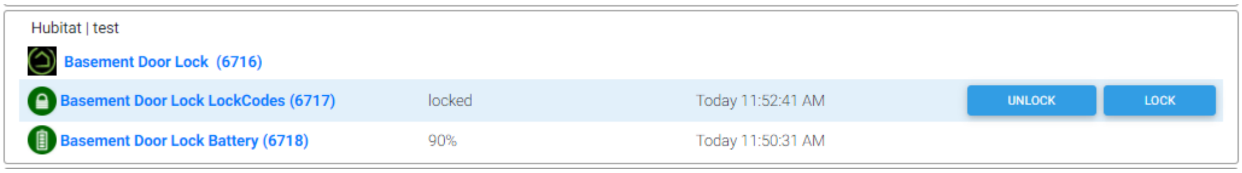
ReliableBasementDoorLock (6713)

ReliableBasementDoorLock Lock (6714)	unlocked with timeout	Today 11:52:41 AM
ReliableBasementDoorLock Battery (6715)	90%	Today 11:50:31 AM

Figure 36 Reliable Lock HS Device and Features

4.21 Generic Z-Wave Lock

The Z-Wave lock support management of lock codes within Hubitat Elevation. This support was not enabled in the HS integration.



Hubitat | test

Basement Door Lock (6716)				
Basement Door Lock LockCodes (6717)	locked	Today 11:52:41 AM	UNLOCK	LOCK
Basement Door Lock Battery (6718)	90%	Today 11:50:31 AM		

Figure 37 Generic Z-Wave Lock HS Device and Features