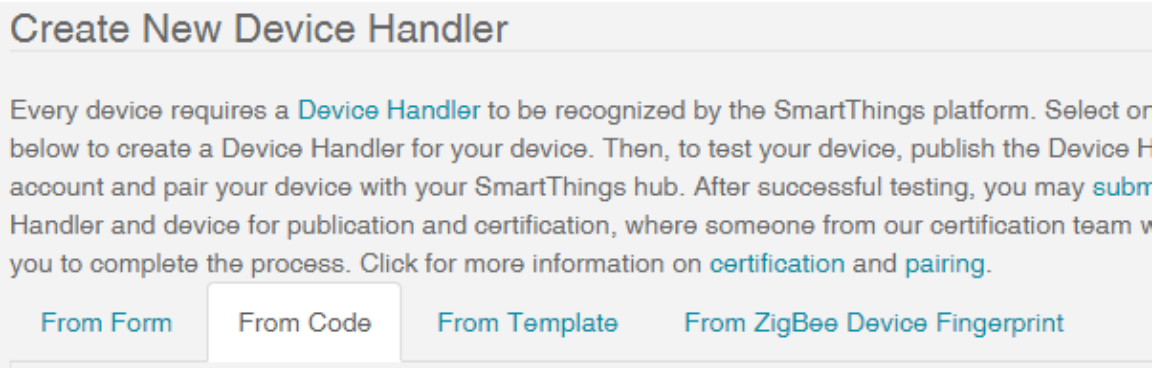
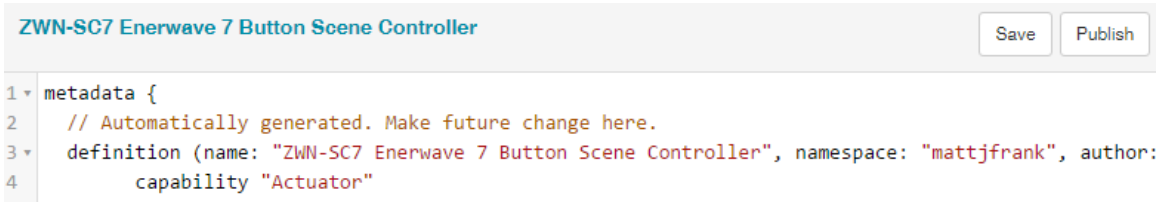
How to add and factory reset Enerwave ZWN-SC7 with SmartThings Hub

Follow the step by step instructions below, to successfully configure our ZWN-SC7 with SmartThings Hub:

1. Go to: **https://graph.api.smartthings.com** and log in with your SmartThings ID and password
2. Find "**My Device Handlers**" and "**Create a new device handler**"
   a. Go to **"From Code"**

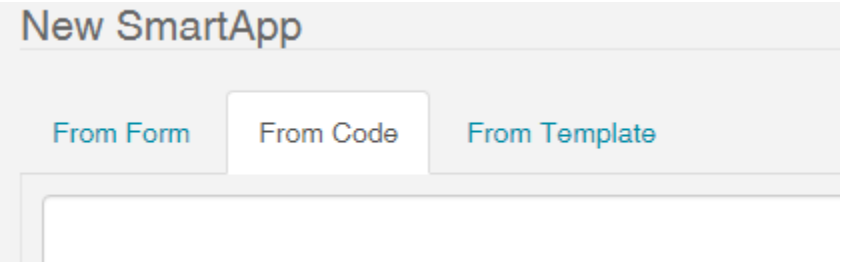


   b. Copy and Paste **ZWN-SC7 device type code below**
   c. Click "**Create**" at the bottom of the page
   d. Click "**Save**" and "**Publish**"



3. Find "**My Smart App**" and "**Create a new smart app**"

   a. Go to **"From Code"**



   b. Copy and Paste **ZWN-SC7 smart app code below**
   c. Click "**Create**" at the bottom of the page
   d. Click "**Save**" and "**Publish**"

4. Press "**Connect New Device**" under Marketplace and search a new device, then
    a. Press and hold the largest button of the scene controller for 3 seconds until the top two LED bulbs light-up.
    b. Press the top-left button, the LED indicator will drop down to the middle-left.
    c. Press the middle-left button, the LED indicator will drop down to the bottom-left.
    d. After you press the bottom-left button, it will run into pairing for about 5 seconds.
    e. All LED indicators will flash slowly, it means scene controller is ready to be included to/excluded from Z-Wave network.

    *If the scene controller has successfully included/excluded with your Smartthing Hub, all LEDs of ZWN-SC7 will flash 3 times slowly. If all LEDs of ZWN-SC7 flash 5 times quickly, please reset the circuit breaker and repeat the steps a-e.

5. When the device is discovered, follow the in-app prompts to configure the device and tap "done"
6. Go to "Marketplace," tap "SmartApps," find "My Apps" at the end and Click the ZWN-SC7 Scene Controller app
7. Set the controller to the ZWN-SC7 you want to control, and follow the in-app settings
8. Press "configuration" under ZWN-SC7 device page after programming
9. Test the scene controller

Factory reset devices
https://support.smartthings.com/hc/en-us/articles/200878314-How-to-exclude-Z-Wave-devices
Excluding or resetting a device that is not yet connected to SmartThings (General Device Exclusion)

Tap the menu
Tap My Locations
Tap the gear icon beside your Location
Tap your Hub
Tap Z-Wave Utilities
Tap General Device Exclusion
When you see the message "Please follow manufacturer's instructions to remove the Z-Wave device from My SmartThings," perform the required exclusion process (e.g., pressing a button on the device)
Enerwave:

a. Press and hold the biggest button of scene controller for 3 seconds until the top two LEDs light up.
b. Press the top-left button and the LED indicator will drop down to the middle-left.
c. Press the middle-left button and the LED indicator will drop down to the bottom-left.
d. After you press the bottom-left button, it will run into paring for about 5 seconds.
e. All LED indicators will flash slowly; it means the scene controller is ready to be removed from the Z-Wave network.
f. Reset at your circuit breaker and device to check the slow flashing LED lights of ZWN-SC7.

****
Copy the below code for ZWN-SC7 device type:
****

```
metadata {
 // Automatically generated. Make future change here.
 definition (name: "ZWN-SC7 Enerwave 7 Button Scene Controller", namespace: "mattjfrank", author: "Matt Frank")
{
    capability "Actuator"
    capability "Button"
    capability "Configuration"
    capability "Indicator"
    capability "Sensor"
```

```
        attribute "currentButton", "STRING"
        attribute "numButtons", "STRING"

    fingerprint deviceId: "0x0202", inClusters:"0x21, 0x2D, 0x85, 0x86, 0x72"
    fingerprint deviceId: "0x0202", inClusters:"0x2D, 0x85, 0x86, 0x72"
  }

  simulator {
    status "button 1 pushed":  "command: 2B01, payload: 01 FF"
    status "button 2 pushed":  "command: 2B01, payload: 02 FF"
    status "button 3 pushed":  "command: 2B01, payload: 03 FF"
    status "button 4 pushed":  "command: 2B01, payload: 04 FF"
    status "button 5 pushed":  "command: 2B01, payload: 05 FF"
    status "button 6 pushed":  "command: 2B01, payload: 06 FF"
    status "button 7 pushed":  "command: 2B01, payload: 07 FF"
    status "button released":  "command: 2C02, payload: 00"
  }

  tiles {
    standardTile("button", "device.button", width: 2, height: 2) {
      state "default", label: " ", icon: "st.unknown.zwave.remote-controller", backgroundColor: "#ffffff"
      state "button 1", label: "1", icon: "st.Weather.weather14", backgroundColor: "#79b821"
      state "button 2", label: "2", icon: "st.Weather.weather14", backgroundColor: "#79b821"
      state "button 3", label: "3", icon: "st.Weather.weather14", backgroundColor: "#79b821"
      state "button 4", label: "4", icon: "st.Weather.weather14", backgroundColor: "#79b821"
      state "button 5", label: "5", icon: "st.Weather.weather14", backgroundColor: "#79b821"
      state "button 6", label: "6", icon: "st.Weather.weather14", backgroundColor: "#79b821"
      state "button 7", label: "7", icon: "st.Weather.weather14", backgroundColor: "#79b821"

    }

        // Configure button.  Syncronize the device capabilities that the UI provides
    standardTile("configure", "device.configure", inactiveLabel: false, decoration: "flat") {
      state "configure", label:'', action:"configuration.configure", icon:"st.secondary.configure"


    }


    main "button"
    details (["button", "configure"])
  }
}

// parse events into attributes
def parse(String description) {
  log.debug "Parsing '${description}'"

    def result = null
  def cmd = zwave.parse(description)
  if (cmd) {
    result = zwaveEvent(cmd)
  }
  return result

}
```

```
// Handle a button being pressed
def buttonEvent(button) {
  button = button as Integer
  def result = []


    updateState("currentButton", "$button")

    if (button > 0) {
       // update the device state, recording the button press
       result << createEvent(name: "button", value: /*"pushed"*/ "button $button", data: [buttonNumber: button], descriptionText: "$device.displayName button $button was pushed", isStateChange: true)

       // turn off the button LED
       result << response(zwave.sceneActuatorConfV1.sceneActuatorConfReport(dimmingDuration: 255, level: 255, sceneId: 0))
  }
    else {
       // update the device state, recording the button press
       result << createEvent(name: "button", value: "default", descriptionText: "$device.displayName button was released", isStateChange: true)

       result << response(zwave.sceneActuatorConfV1.sceneActuatorConfReport(dimmingDuration: 255, level: 255, sceneId: 0))
    }

    result
}

// A zwave command for a button press was received
def zwaveEvent(physicalgraph.zwave.commands.sceneactivationv1.SceneActivationSet cmd) {

  // The controller likes to repeat the command... ignore repeats
  if (state.lastScene == cmd.sceneId && (state.repeatCount < 4) && (now() - state.repeatStart < 2000)) {
     log.debug "Button ${cmd.sceneId} repeat ${state.repeatCount}x ${now()}"
       state.repeatCount = state.repeatCount + 1
       createEvent([:])
    }
    else {
     // If the button was really pressed, store the new scene and handle the button press
       state.lastScene = cmd.sceneId
       state.lastLevel = 0
       state.repeatCount = 0
       state.repeatStart = now()

       buttonEvent(cmd.sceneId)
    }
}

// A scene command was received -- it's probably scene 0, so treat it like a button release
def zwaveEvent(physicalgraph.zwave.commands.sceneactuatorconfv1.SceneActuatorConfGet cmd) {

  buttonEvent(cmd.sceneId)

}

// The controller sent a scene activation report.  Log it, but it really shouldn't happen.
```

```
def zwaveEvent(physicalgraph.zwave.commands.sceneactuatorconfv1.SceneActuatorConfReport cmd) {
   log.debug "Scene activation report"
  log.debug "Scene ${cmd.sceneId} set to ${cmd.level}"

   createEvent([:])
}


// Configuration Reports are replys to configuration value requests... If we knew what configuration parameters
// to request this could be very helpful.
def zwaveEvent(physicalgraph.zwave.commands.configurationv1.ConfigurationReport cmd) {
  createEvent([:])
}

// The VRC supports hail commands, but I haven't seen them.
def zwaveEvent(physicalgraph.zwave.commands.hailv1.Hail cmd) {
   createEvent([name: "hail", value: "hail", descriptionText: "Switch button was pressed", displayed: false])
}

// Update manufacturer information when it is reported
def zwaveEvent(physicalgraph.zwave.commands.manufacturerspecificv2.ManufacturerSpecificReport cmd) {
  if (state.manufacturer != cmd.manufacturerName) {
   updateDataValue("manufacturer", cmd.manufacturerName)
  }

   createEvent([:])
}

// Association Groupings Reports tell us how many groupings the device supports.  This equates to the number of
// buttons/scenes in the VRCS
def zwaveEvent(physicalgraph.zwave.commands.associationv2.AssociationGroupingsReport cmd) {
  def response = []

   log.debug "${getDataByName("numButtons")} buttons stored"
  if (getDataByName("numButtons") != "$cmd.supportedGroupings") {
   updateState("numButtons", "$cmd.supportedGroupings")
      log.debug "${cmd.supportedGroupings} groups available"
      response << createEvent(name: "numButtons", value: cmd.supportedGroupings, displayed: false)

      response << associateHub()
  }
   else {
    response << createEvent(name: "numButtons", value: cmd.supportedGroupings, displayed: false)
   }
   return response
}


// Handles all Z-Wave commands we don't know we are interested in
def zwaveEvent(physicalgraph.zwave.Command cmd) {
   createEvent([:])
}

// handle commands

// Create a list of the configuration commands to send to the device
```

```
def configurationCmds() {
 // Always check the manufacturer and the number of groupings allowed
 def commands = [
  zwave.manufacturerSpecificV1.manufacturerSpecificGet().format(),
  zwave.associationV1.associationGroupingsGet().format(),
  zwave.sceneControllerConfV1.sceneControllerConfSet(groupId:1, sceneId:1).format(),
  zwave.sceneControllerConfV1.sceneControllerConfSet(groupId:2, sceneId:2).format(),
  zwave.sceneControllerConfV1.sceneControllerConfSet(groupId:3, sceneId:3).format(),
  zwave.sceneControllerConfV1.sceneControllerConfSet(groupId:4, sceneId:4).format(),
  zwave.sceneControllerConfV1.sceneControllerConfSet(groupId:5, sceneId:5).format(),
  zwave.sceneControllerConfV1.sceneControllerConfSet(groupId:6, sceneId:6).format(),
  zwave.sceneControllerConfV1.sceneControllerConfSet(groupId:7, sceneId:7).format()

  ]

  commands << associateHub()

  delayBetween(commands)
}

// Configure the device
def configure() {
 def cmd=configurationCmds()
  log.debug("Sending configuration: ${cmd}")
  return cmd
}


//
// Associate the hub with the buttons on the device, so we will get status updates
def associateHub() {
  def commands = []

  // Loop through all the buttons on the controller
  for (def buttonNum = 1; buttonNum <= integer(getDataByName("numButtons")); buttonNum++) {

      // Associate the hub with the button so we will get status updates
      commands << zwave.associationV1.associationSet(groupingIdentifier: buttonNum, nodeId: zwaveHubNodeId).format()

 }

  return commands
}

// Update State
// Store mode and settings
def updateState(String name, String value) {
 state[name] = value
 device.updateDataValue(name, value)
}

// Get Data By Name
// Given the name of a setting/attribute, lookup the setting's value
def getDataByName(String name) {
 state[name] ?: device.getDataValue(name)
}
```

```
//Stupid conversions

// convert a double to an integer
def integer(double v) {
 return v.toInteger()
}

// convert a hex string to integer
def integerhex(String v) {
 if (v == null) {
    return 0
  }

 return Integer.parseInt(v, 16)
}

// convert a hex string to integer
def integer(String v) {
 if (v == null) {
    return 0
  }

 return Integer.parseInt(v)
}

*******************************************************************************************
Copy the below code for ZWN-SC7 smart app
*******************************************************************************************
/**
 *     Button Controller App for ZWN-SC7
 *
 *     Author: Matt Frank based on VRCS Button Controller by Brian Dahlem, based on SmartThings Button
Controller
 *     Date Created: 2014-12-18
 *  Last Updated: 2015-10-05
 *
 */
definition(
    name: "ZWN-SC7     Button Controller",
    namespace: "mattjfrank",
    author: "Matt Frank, using code from Brian Dahlem",
    description: "ZWN-SC7     7-Button      Scene   Controller    Button Assignment App",
    category: "Convenience",
    iconUrl: "https://s3.amazonaws.com/smartapp-icons/MyApps/Cat-MyApps.png",
    iconX2Url: "https://s3.amazonaws.com/smartapp-icons/MyApps/Cat-MyApps@2x.png"
)

preferences {
  page(name: "selectButton")
  page(name: "configureButton1")
  page(name: "configureButton2")
  page(name: "configureButton3")
  page(name: "configureButton4")
  page(name: "configureButton5")
  page(name: "configureButton6")
  page(name: "configureButton7")
}

def selectButton() {
  dynamicPage(name:  "selectButton",  title:  "First,  select  which  ZWN-SC7",  nextPage:  "configureButton1",
uninstall: configured()) {
```

```
    section {
      input "buttonDevice", "capability.button", title: "Controller", multiple: false, required: true
    }


  }
}

def configureButton1() {
  dynamicPage(name: "configureButton1", title: "1st button, what do you want it to do?",
    nextPage: "configureButton2", uninstall: configured(), getButtonSections(1))

}

def configureButton2() {
  dynamicPage(name: "configureButton2", title: "2nd button, what do you want it to do?",
    nextPage: "configureButton3", uninstall: configured(), getButtonSections(2))
}

def configureButton3() {
  dynamicPage(name: "configureButton3", title: "3rd button, what do you want it to do?",
    nextPage: "configureButton4", uninstall: configured(), getButtonSections(3))
}
def configureButton4() {
  dynamicPage(name: "configureButton4", title: "4th button, what do you want it to do?",
    nextPage: "configureButton5", uninstall: configured(), getButtonSections(4))
}
def configureButton5() {
  dynamicPage(name: "configureButton5", title: "5th button, what do you want it to do?",
    nextPage: "configureButton6", uninstall: configured(), getButtonSections(5))
}
def configureButton6() {
  dynamicPage(name: "configureButton6", title: "6th button, what do you want it to do?",
    nextPage: "configureButton7", uninstall: configured(), getButtonSections(6))
}
def configureButton7() {
  dynamicPage(name: "configureButton7", title: "7th  button, what do you want it to do?",
    install: true, uninstall: true, getButtonSections(7))
}

def getButtonSections(buttonNumber) {
  return {
      section(title: "Toggle these...", hidden: hideSection(buttonNumber, "toggle"), hideable: true) {
      input  "lights_${buttonNumber}_toggle",  "capability.switch",  title:  "switches:",  multiple:  true,
required: false
      input  "locks_${buttonNumber}_toggle",  "capability.lock",  title:  "locks:",  multiple:  true,  required:
false
      input "sonos_${buttonNumber}_toggle", "capability.musicPlayer", title: "music players:", multiple: true,
required: false
    }
      section(title: "Turn on these...", hidden: hideSection(buttonNumber, "on"), hideable: true) {
      input "lights_${buttonNumber}_on", "capability.switch", title: "switches:", multiple: true, required:
false
      input  "sonos_${buttonNumber}_on",  "capability.musicPlayer",  title:  "music  players:",  multiple:  true,
required: false
    }
      section(title: "Turn off these...", hidden: hideSection(buttonNumber, "off"), hideable: true) {
      input "lights_${buttonNumber}_off", "capability.switch", title: "switches:", multiple: true, required:
false
      input "sonos_${buttonNumber}_off", "capability.musicPlayer", title: "music players:", multiple: true,
required: false
    }
      section(title: "Locks:", hidden: hideLocksSection(buttonNumber), hideable: true) {
      input "locks_${buttonNumber}_unlock", "capability.lock", title: "Unlock these locks:", multiple: true,
required: false
      input  "locks_${buttonNumber}_lock",  "capability.lock",  title:  "Lock  these  locks:",  multiple:  true,
required: false
    }
```

```
    section("Modes") {
      input "mode_${buttonNumber}_on", "mode", title: "Activate these modes:", required: false
    }
    def phrases = location.helloHome?.getPhrases()*.label
    if (phrases) {
      section("Hello Home Actions") {
        log.trace phrases
        input "phrase_${buttonNumber}_on", "enum", title: "Activate these phrases:", required: false, options:
phrases
      }
    }
  }
}

def installed() {
  initialize()
}

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

def initialize() {

  subscribe(buttonDevice, "button", buttonEvent)

    if (relayDevice) {
        log.debug "Associating ${relayDevice.deviceNetworkId}"
        if (relayAssociate == true) {
            buttonDevice.associateLoad(relayDevice.deviceNetworkId)
        }
        else {
            buttonDevice.associateLoad(0)
        }
    }
}

def configured() {
  return     buttonDevice   ||   buttonConfigured(1)   ||   buttonConfigured(2)   ||   buttonConfigured(3)   ||
buttonConfigured(4) || buttonConfigured(5) || buttonConfigured(6) || buttonConfigured(7)
}

def buttonConfigured(idx) {
  return settings["lights_$idx_toggle"] ||
    settings["locks_$idx_toggle"] ||
    settings["sonos_$idx_toggle"] ||
    settings["mode_$idx_on"] ||
        settings["lights_$idx_on"] ||
    settings["locks_$idx_on"] ||
    settings["sonos_$idx_on"] ||
        settings["lights_$idx_off"] ||
    settings["locks_$idx_off"] ||
    settings["sonos_$idx_off"]
}

def buttonEvent(evt){
  log.debug "buttonEvent"
  if(allOk) {
    def buttonNumber = evt.jsonData.buttonNumber
    log.debug "buttonEvent: $evt.name - ($evt.data)"
    log.debug "button: $buttonNumber"


    def recentEvents = buttonDevice.eventsSince(new Date(now() - 1000)).findAll{it.value == evt.value &&
it.data == evt.data}
    log.debug "Found ${recentEvents.size()?:0} events in past 1 seconds"
```

```
    if(recentEvents.size <= 3){
      switch(buttonNumber) {
        case ~/.*1.*/:
          executeHandlers(1)
          break
        case ~/.*2.*/:
          executeHandlers(2)
          break
        case ~/.*3.*/:
          executeHandlers(3)
          break
        case ~/.*4.*/:
          executeHandlers(4)
          break
        case ~/.*5.*/:
          executeHandlers(5)
          break
        case ~/.*6.*/:
          executeHandlers(6)
          break
        case ~/.*7.*/:
          executeHandlers(7)
          break
      }
    } else {
      log.debug "Found recent button press events for $buttonNumber with value $value"
    }
  }
    else {
      log.debug "NotOK"
    }
}

def executeHandlers(buttonNumber) {
  log.debug "executeHandlers: $buttonNumber"

  def lights = find('lights', buttonNumber, "toggle")
  if (lights != null) toggle(lights)

  def locks = find('locks', buttonNumber, "toggle")
  if (locks != null) toggle(locks)

  def sonos = find('sonos', buttonNumber, "toggle")
  if (sonos != null) toggle(sonos)

  lights = find('lights', buttonNumber, "on")
  if (lights != null) flip(lights, "on")

  locks = find('locks', buttonNumber, "unlock")
  if (locks != null) flip(locks, "unlock")

  sonos = find('sonos', buttonNumber, "on")
  if (sonos != null) flip(sonos, "on")

  lights = find('lights', buttonNumber, "off")
  if (lights != null) flip(lights, "off")

  locks = find('locks', buttonNumber, "lock")
  if (locks != null) flip(locks, "lock")

  sonos = find('sonos', buttonNumber, "off")
  if (sonos != null) flip(sonos, "off")

  def mode = find('mode', buttonNumber, "on")
  if (mode != null) changeMode(mode)

  def phrase = find('phrase', buttonNumber, "on")
```

```
  if (phrase != null) location.helloHome.execute(phrase)
}

def find(type, buttonNumber, value) {
  def preferenceName = type + "_" + buttonNumber + "_" + value
  def pref = settings[preferenceName]
  if(pref != null) {
    log.debug "Found: $pref for $preferenceName"
  }

  return pref
}

def flip(devices, newState) {
  log.debug "flip: $devices = ${devices*.currentValue('switch')}"


  if (newState == "off") {
    devices.off()
  }
  else if (newState == "on") {
    devices.on()
  }
  else if (newState == "unlock") {
    devices.unlock()
  }
  else if (newState == "lock") {
    devices.lock()
  }
}

def toggle(devices) {
  log.debug "toggle: $devices = ${devices*.currentValue('switch')}"

  if (devices*.currentValue('switch').contains('on')) {
    devices.off()
  }
  else if (devices*.currentValue('switch').contains('off')) {
    devices.on()
  }
  else if (devices*.currentValue('lock').contains('locked')) {
    devices.unlock()
  }
  else if (devices*.currentValue('lock').contains('unlocked')) {
    devices.lock()
  }
  else {
    devices.on()
  }
}

def changeMode(mode) {
  log.debug "changeMode: $mode, location.mode = $location.mode, location.modes = $location.modes"

  if (location.mode != mode && location.modes?.find { it.name == mode }) {
    setLocationMode(mode)
  }
}

// execution filter methods
private getAllOk() {
  modeOk && daysOk && timeOk
}

private getModeOk() {
  def result = !modes || modes.contains(location.mode)
  log.trace "modeOk = $result"
  result
```

```
}

private getDaysOk() {
  def result = true
  if (days) {
    def df = new java.text.SimpleDateFormat("EEEE")
    if (location.timeZone) {
      df.setTimeZone(location.timeZone)
    }
    else {
      df.setTimeZone(TimeZone.getTimeZone("America/New_York"))
    }
    def day = df.format(new Date())
    result = days.contains(day)
  }
  log.trace "daysOk = $result"
  result
}

private getTimeOk() {
  def result = true
  if (starting && ending) {
    def currTime = now()
    def start = timeToday(starting).time
    def stop = timeToday(ending).time
    result = start < stop ? currTime >= start && currTime <= stop : currTime <= stop || currTime >= start
  }
  log.trace "timeOk = $result"
  result
}

private hhmm(time, fmt = "h:mm a")
{
  def t = timeToday(time, location.timeZone)
  def f = new java.text.SimpleDateFormat(fmt)
  f.setTimeZone(location.timeZone ?: timeZone(time))
  f.format(t)
}

private hideOptionsSection() {
  (starting || ending || days || modes) ? false : true
}

private hideSection(buttonNumber, action) {
  (find("lights", buttonNumber, action) || find("locks", buttonNumber, action) || find("sonos", buttonNumber,
action)) ? false : true
}

private hideLocksSection(buttonNumber) {
  (find("lights", buttonNumber, "lock") || find("locks", buttonNumber, "unlock")) ? false : true
}

private timeIntervalLabel() {
  (starting && ending) ? hhmm(starting) + "-" + hhmm(ending, "h:mm a z") : ""
}

private integer(String s) {
  return Integer.parseInt(s)
}
```
**********End

Enerwave Home Automation 02/17/2016

Fix citation:

Acknowledge by Matt Fank's hard work and contribution from community