```
/**
 *  Copyright 2015 SmartThings
 *
 *  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.
 *
 *  Z-Wave LED Strip Improved
 *
 *  Author: Chris Cheng
 *  Date: 2017-8-3
 */

metadata {
        definition (name: "Aeon Labs LED Strip (Improved)", namespace: "Aeotec by Aeon Labs",
author: "Chris Cheng") {
                capability "Switch Level"
                capability "Color Control"
                capability "Color Temperature"
                capability "Switch"
                capability "Refresh"
                capability "Actuator"
                capability "Sensor"

                command "reset"
                command "refresh"
        command "config1"

                fingerprint inClusters: "0x26,0x33"
                fingerprint inClusters: "0x33"
        }

        simulator {
        }

        standardTile("switch", "device.switch", width: 1, height: 1, canChangeIcon: true) {
                state "on", label:'${name}', action:"switch.off", icon:"st.lights.philips.hue-
single", backgroundColor:"#79b821", nextState:"turningOff"
                state "off", label:'${name}', action:"switch.on", icon:"st.lights.philips.hue-
single", backgroundColor:"#ffffff", nextState:"turningOn"
                state "turningOn", label:'${name}', action:"switch.off",
icon:"st.lights.philips.hue-single", backgroundColor:"#79b821", nextState:"turningOff"
                state "turningOff", label:'${name}', action:"switch.on",
icon:"st.lights.philips.hue-single", backgroundColor:"#ffffff", nextState:"turningOn"
        }
        standardTile("reset", "device.reset", inactiveLabel: false, decoration: "flat") {
                state "default", label:"Reset Color", action:"reset", icon:"st.lights.philips.hue-
single"
        }
        standardTile("refresh", "device.switch", inactiveLabel: false, decoration: "flat") {
                state "default", label:"", action:"refresh.refresh", icon:"st.secondary.refresh"
        }
        controlTile("levelSliderControl", "device.level", "slider", height: 1, width: 2,
inactiveLabel: false, range:"(0..100)") {
                state "level", action:"switch level.setLevel"
        }
        controlTile("rgbSelector", "device.color", "color", height: 3, width: 3, inactiveLabel:
false) {
                state "color", action:"setColor"
        }
        valueTile("level", "device.level", inactiveLabel: false, decoration: "flat") {
```

```
                state "level", label: 'Level ${currentValue}%'
        }
        controlTile("colorTempControl", "device.colorTemperature", "slider", height: 1, width: 2,
inactiveLabel: false) {
                state "colorTemperature", action:"setColorTemperature"
        }
        valueTile("hue", "device.hue", inactiveLabel: false, decoration: "flat") {
                state "hue", label: 'Hue ${currentValue}   '
        }

        standardTile("config1", "device.button", inactiveLabel: false, decoration: "flat") {
                state "default", label:"Rainbow", action:"config1"
        }

        main(["switch"])
        //details(["switch", "levelSliderControl", "rgbSelector", "reset", "colorTempControl",
"refresh"])
    details(["switch", "levelSliderControl", "rgbSelector", "reset", "colorTempControl",
"refresh", "config1"])
}

def updated() {
        response(refresh())
}

def parse(description) {
        def result = null
        if (description != "updated") {
                def cmd = zwave.parse(description, [0x20: 1, 0x26: 3, 0x70: 1, 0x33:3])
                if (cmd) {
                        result = zwaveEvent(cmd)
                        log.debug("'$description' parsed to $result")
                } else {
                        log.debug("Couldn't zwave.parse '$description'")
                }
        }
        result
}

def zwaveEvent(physicalgraph.zwave.commands.basicv1.BasicReport cmd) {
        dimmerEvents(cmd)
}

def zwaveEvent(physicalgraph.zwave.commands.basicv1.BasicSet cmd) {
        dimmerEvents(cmd)
}

def zwaveEvent(physicalgraph.zwave.commands.switchmultilevelv3.SwitchMultilevelReport cmd) {
        dimmerEvents(cmd)
}

private dimmerEvents(physicalgraph.zwave.Command cmd) {
        def value = (cmd.value ? "on" : "off")
        def result = [createEvent(name: "switch", value: value, descriptionText:
"$device.displayName was turned $value")]
        if (cmd.value) {
                result << createEvent(name: "level", value: cmd.value, unit: "%")
        }
        return result
}

def zwaveEvent(physicalgraph.zwave.commands.hailv1.Hail cmd) {
        response(command(zwave.switchMultilevelV1.switchMultilevelGet()))
}

def zwaveEvent(physicalgraph.zwave.commands.securityv1.SecurityMessageEncapsulation cmd) {
        def encapsulatedCommand = cmd.encapsulatedCommand([0x20: 1, 0x84: 1])
        if (encapsulatedCommand) {
                state.sec = 1
```

```
                def result = zwaveEvent(encapsulatedCommand)
                result = result.collect {
                        if (it instanceof physicalgraph.device.HubAction &&
!it.toString().startsWith("9881")) {
                                response(cmd.CMD + "00" + it.toString())
                        } else {
                                it
                        }
                }
                result
        }
}


def zwaveEvent(physicalgraph.zwave.Command cmd) {
        def linkText = device.label ?: device.name
        [linkText: linkText, descriptionText: "$linkText: $cmd", displayed: false]
}

def on() {
        commands([
                zwave.basicV1.basicSet(value: 0xFF),
                zwave.switchMultilevelV3.switchMultilevelGet(),
        ], 3500)
}

def off() {
        commands([
                zwave.basicV1.basicSet(value: 0x00),
                zwave.switchMultilevelV3.switchMultilevelGet(),
        ], 3500)
}

def setLevel(level) {
        setLevel(level, 1)
}

def setLevel(level, duration) {
        if(level > 99) level = 99
        commands([
                zwave.switchMultilevelV3.switchMultilevelSet(value: level, dimmingDuration:
duration),
                zwave.switchMultilevelV3.switchMultilevelGet(),
        ], (duration && duration < 12) ? (duration * 1000) : 3500)
}

def refresh() {
        commands([
                zwave.switchMultilevelV3.switchMultilevelGet(),
        ], 1000)
}

def setSaturation(percent) {
        log.debug "setSaturation($percent)"
        setColor(saturation: percent)
}

def setHue(value) {
        log.debug "setHue($value)"
        setColor(hue: value)
}

def setColor(value) {
        def result = []
        log.debug "setColor: ${value}"
        if (value.hex) {
                def c = value.hex.findAll(/[0-9a-fA-F]{2}/).collect { Integer.parseInt(it, 16) }
                result << zwave.switchColorV3.switchColorSet(red:c[0], green:c[1], blue:c[2],
warmWhite:0, coldWhite:0)
```

```
        } else {
                def hue = value.hue ?: device.currentValue("hue")
                def saturation = value.saturation ?: device.currentValue("saturation")
                if(hue == null) hue = 13
                if(saturation == null) saturation = 13
                def rgb = huesatToRGB(hue, saturation)
                result << zwave.switchColorV3.switchColorSet(red: rgb[0], green: rgb[1], blue:
rgb[2], warmWhite:0, coldWhite:0)
        }

        if(value.hue) sendEvent(name: "hue", value: value.hue)
        if(value.hex) sendEvent(name: "color", value: value.hex)
        if(value.switch) sendEvent(name: "switch", value: value.switch)
        if(value.saturation) sendEvent(name: "saturation", value: value.saturation)

        commands(result)
}

def setColorTemperature(percent) {
        if(percent > 99) percent = 99
        int warmValue = percent * 255 / 99
        command(zwave.switchColorV3.switchColorSet(red:0, green:0, blue:0, warmWhite:warmValue,
coldWhite:(255 - warmValue)))
}

def reset() {
        log.debug "reset()"
        command(zwave.switchColorV3.switchColorSet(red:0, green:0, blue:0, warmWhite:255,
coldWhite:255))
    //sendEvent(name: "color", value: "#ffffff")
        //setColorTemperature(99)
}

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

private commands(commands, delay=200) {
        delayBetween(commands.collect{ command(it) }, delay)
}

def rgbToHSV(red, green, blue) {
        float r = red / 255f
        float g = green / 255f
        float b = blue / 255f
        float max = [r, g, b].max()
        float delta = max - [r, g, b].min()
        def hue = 13
        def saturation = 0
        if (max && delta) {
                saturation = 100 * delta / max
                if (r == max) {
                        hue = ((g - b) / delta) * 100 / 6
                } else if (g == max) {
                        hue = (2 + (b - r) / delta) * 100 / 6
                } else {
                        hue = (4 + (r - g) / delta) * 100 / 6
                }
        }
        [hue: hue, saturation: saturation, value: max * 100]
}

def huesatToRGB(float hue, float sat) {
        while(hue >= 100) hue -= 100
        int h = (int)(hue / 100 * 6)
```

```
        float f = hue / 100 * 6 - h
        int p = Math.round(255 * (1 - (sat / 100)))
        int q = Math.round(255 * (1 - (sat / 100) * f))
        int t = Math.round(255 * (1 - (sat / 100) * (1 - f)))
        switch (h) {
                case 0: return [255, t, p]
                case 1: return [q, 255, p]
                case 2: return [p, 255, t]
                case 3: return [p, q, 255]
                case 4: return [t, p, 255]
                case 5: return [255, p, q]
        }
}

def config1() {
        //log.debug "config1()"
    def cmds = []
    cmds << zwave.configurationV1.configurationSet(scaledConfigurationValue: 0, parameterNumber:
252, size: 4)
    cmds << zwave.configurationV1.configurationSet(scaledConfigurationValue: 73597278,
parameterNumber: 37, size: 4)
    cmds << zwave.configurationV1.configurationSet(scaledConfigurationValue: 1, parameterNumber:
40, size: 1)
    cmds << zwave.basicV1.basicGet()
        commands(cmds)
}

//def config2(){}
```