/\*\*

\* Monoprice Motion Sensor

\*

\* 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.

\*

\* Updates:

\* -------

\* 02-18-2016 : Initial commit

\* 04-08-2016 : Added fingerprint info

\* 04-10-2016 : Added Refesh tile

\* 08-27-2016 : Modified the device handler for my liking, primarly for looks and feel.

\*

\*/

preferences {

input description: "Number of minutes after movement is gone before its reported inactive by the sensor.", displayDuringSetup: false, type: "paragraph", element: "paragraph"

input "inactivityTimeout", "number", title: "Inactivity Timeout", displayDuringSetup: true, default: 3

input description: "This feature allows you to correct any temperature variations by selecting an offset. Ex: If your sensor consistently reports a temp that's 5 degrees too warm, you'd enter \"-5\". If 3 degrees too cold, enter \"+3\".", displayDuringSetup: false, type: "paragraph", element: "paragraph"

input "tempOffset", "number", title: "Temperature Offset", description: "Adjust temperature by this many degrees", range: "\*..\*", displayDuringSetup: false

input description: "This feature allows you to change the temperature Unit. If left blank or anything else is typed the default is F.", displayDuringSetup: false, type: "paragraph", element: "paragraph"

input "tempUnit", "string", title: "Celsius or Fahrenheit", description: "Temperature Unit (Type C or F)", displayDuringSetup: false

}

metadata {

definition (name:"My Monoprice Motion Sensor", namespace:"st340", author: "wsf52") {

capability "Battery"

capability "Motion Sensor"

capability "Temperature Measurement"

capability "Sensor"

capability "Polling"

capability "Refresh"

command "refresh"

fingerprint deviceId: "0x2001", inClusters: "0x71,0x85,0x80,0x72,0x30,0x86,0x31,0x70,0x84"

}

tiles(scale: 2) {

multiAttributeTile(name:"motion", type: "generic", width: 6, height: 4){

tileAttribute ("device.motion", key: "PRIMARY\_CONTROL") {

attributeState "active", label:'motion', icon:"st.motion.motion.active", backgroundColor:"#53a7c0"

attributeState "inactive", label:'no motion', icon:"st.motion.motion.inactive", backgroundColor:"#ffffff"

}

}

valueTile("temperature", "device.temperature", width: 2, height: 2) {

state("temperature", label:'${currentValue}°', unit:"F",

backgroundColors:[

[value: 31, color: "#153591"],

[value: 44, color: "#1e9cbb"],

[value: 59, color: "#90d2a7"],

[value: 74, color: "#44b621"],

[value: 84, color: "#f1d801"],

[value: 95, color: "#d04e00"],

[value: 96, color: "#bc2323"]

]

)

}

valueTile("battery", "device.battery", decoration: "flat", inactiveLabel: false, width: 2, height: 2) {

state "battery", label:'${currentValue}% battery', unit:""

}

standardTile("refresh", "device.motion", width: 2, height: 2, inactiveLabel: false, decoration: "flat") {

state "default", label:'Refresh', action:"refresh.refresh", icon:"st.secondary.refresh-icon"

}

main(["motion", "temperature"])

details(["motion", "temperature", "battery", "refresh"])

}

}

def parse(String description) {

log.trace "Parse Raw: ${description}"

def result = []

// Using reference in: <http://www.pepper1.net/zwavedb/device/197>

def cmd = zwave.parse(description, [0x20: 1, 0x80: 1, 0x31: 2, 0x84: 2, 0x71: 1, 0x30: 1])

if (cmd) {

if (cmd instanceof physicalgraph.zwave.commands.wakeupv2.WakeUpNotification) {

result.addAll(sendSettingsUpdate(cmd))

}

result << createEvent(zwaveEvent(cmd))

if (cmd.CMD == "8407") {

result << new physicalgraph.device.HubAction(zwave.wakeUpV1.wakeUpNoMoreInformation().format())

}

}

if (inactivityTimeout) {

log.debug "Applying preferences for Monoprice Motion Sensor: ${inactivityTimeout}"

zwave.configurationV1.configurationSet(configurationValue: [inactivityTimeout], parameterNumber: 1, size: 1).format()

log.debug "zwaveEvent ConfigurationReport: '${cmd}'"

}

return result

}

def zwaveEvent(physicalgraph.zwave.commands.wakeupv2.WakeUpNotification cmd) {

//log.trace "Woke Up!"

def map = [:]

map.value = ""

map.descriptionText = "${device.displayName} woke up."

return map

}

def sendSettingsUpdate(physicalgraph.zwave.commands.wakeupv2.WakeUpNotification cmd) {

/\* def inactivityTimeout = (settings.inactivityTimeout == null ?

1 : Integer.parseInt(settings.inactivityTimeout))

def inactivityTimeoutStr = Integer.toString(inactivityTimeout) \*/

def actions = []

def lastBatteryUpdate = state.lastBatteryUpdate == null ? 0 : state.lastBatteryUpdate

if ((new Date().time - lastBatteryUpdate) > 1000 \* 60 \* 60 \* 24) {

actions.addAll([

response(zwave.batteryV1.batteryGet().format()),

[ descriptionText: "Requested battery update from ${device.displayName}.", value: "" ],

response("delay 600"),

])

}

actions.addAll([

response(zwave.configurationV1.configurationSet(

configurationValue: [inactivityTimeout], defaultValue: False, parameterNumber: 1, size: 1).format()),

response("delay 600"),

[ descriptionText: "${device.displayName} was sent inactivity timeout of ${inactivityTimeoutStr}.", value: "" ]

])

actions

}

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

def map = [:]

map.name = "motion"

map.value = cmd.value ? "active" : "inactive"

map.handlerName = map.value

map.descriptionText = cmd.value ? "${device.displayName} detected motion" : "${device.displayName} motion has stopped."

return map

}

def zwaveEvent(physicalgraph.zwave.commands.sensormultilevelv2.SensorMultilevelReport cmd) {

def map = [:]

if (cmd.sensorType == 1) {

def cmdScale = cmd.scale == 1 ? "F" : "C"

def preValue = convertTemperatureIfNeeded(cmd.scaledSensorValue, cmdScale, cmd.precision)

def value = preValue as float

map.unit = tempUnit

map.name = "temperature"

switch(tempUnit) {

case ["C","c"]:

if (tempOffset) {

def offset = tempOffset as float

map.value = value + offset as float

}

else {

map.value = value as float

}

map.value = map.value.round()

map.descriptionText = "${device.displayName} temperature is ${map.value}°C."

break

case ["F","f"]:

if (tempOffset) {

def offset = tempOffset as float

map.value = value + offset as float

}

else {

map.value = value as float

}

map.value = map.value.round()

map.descriptionText = "${device.displayName} temperature is ${map.value}°F."

break

default:

if (tempOffset) {

def offset = tempOffset as float

map.value = value + offset as float

}

else {

map.value = value as float

}

map.value = map.value.round()

map.descriptionText = "${device.displayName} temperature is ${map.value}°."

break

}

}

map

}

def zwaveEvent(physicalgraph.zwave.commands.batteryv1.BatteryReport cmd) {

state.lastBatteryUpdate = new Date().time

def map = [ name: "battery", unit: "%" ]

if (cmd.batteryLevel == 0xFF || cmd.batteryLevel == 0 ) {

map.value = 1

map.descriptionText = "${device.displayName} battery is almost dead!"

} else if (cmd.batteryLevel < 15 ) {

map.value = cmd.batteryLevel

map.descriptionText = "${device.displayName} battery is low!"

} else {

map.value = cmd.batteryLevel

}

map

}

def zwaveEvent(physicalgraph.zwave.Command cmd) {

// Catch-all handler. The sensor does return some alarm values, which

// could be useful if handled correctly (tamper alarm, etc.)

[descriptionText: "Unhandled: ${device.displayName}: ${cmd}", displayed: false]

}

def poll() {

refresh()

}

def refresh() {

log.info "Executing Refresh/Poll per user request"

// sendEvent(name: "motion", value: $device.currentState('motionState')) testing

}