# How To Add HS-FLS100+ GEN1 to SmartThings Hub

Information and code on adding HS-FLS100+ (GEN 1) to SmartThings

## Purpose

This device handler provides extended feature support of this product to SmartThings new app users. By default, if you add the sensor to ST without the device handler, you will not be able to trigger on motion, you can only turn the load on and off. The device handler adds the following:

- Button to turn the load on/off and display the status of the light
- Motion tile that tells you if motion is detected
- LUX tile that shows the current light level

For triggering actions, you can use the built-in automation to trigger on motion and you can set an action to control the load.
If you need to trigger on the light level, install the SmartApp named "Smart Lighting". This will allow you to trigger on the device LUX value.

## To add this Device Handler:

1. Navigate to this URL and log in with your SmartThings account: https://graph-na04-useast2.api.smartthings.com
2. Click on the "My Device Handlers" menu
3. Click "Create New Device Handler" on the right
4. Click "From Code"
5. Click here to view the code or see the snippet below
6. Click "Create" at the bottom
7. Click "Publish" and select "For Me"

## Device Handler Code

**HS-FLS100+**

```
/**
 *  HomeSeer HS-FLS100+
 *
 *  Copyright 2018 HomeSeer
 *
 *
 *
 *
 *  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.
 *
 *        Author: HomeSeer
 *        Date: 7/24/2018
 *
 *        Changelog:
 *
 *        1.0        Initial Version
 *  1.1 Fixed error setting parameters
 *  1.2 Adds option to disable motion activation for lights
 *
 *
 *
 */

metadata {
        definition (name: "FLS100+ Motion Sensor", namespace: "homeseer", author: "support@homeseer.com") {
                capability "Switch"
                capability "Motion Sensor"
                capability "Sensor"
                capability "Polling"
```

```
                capability "Refresh"
        capability "Configuration"
        capability "Illuminance Measurement"



        fingerprint mfr: "000C", prod: "0201", model: "000B"
}


        simulator {
                status "on":  "command: 2003, payload: FF"
                status "off": "command: 2003, payload: 00"


                // reply messages
                reply "2001FF,delay 5000,2602": "command: 2603, payload: FF"
                reply "200100,delay 5000,2602": "command: 2603, payload: 00"
        }


    preferences {
       input ( "onTime", "number", title: "Press Configuration button after changing preferences\n\nOn Time:
Duration (8-720 seconds) [default: 15]", defaultValue: 15,range: "8..720", required: false)
       input ( "luxDisableValue", "number", title: "Lux Value to Disable Sensor: (30-200 lux) [default:
50]", defaultValue: 50, range: "0..200", required: false)
       input ( "luxReportInterval", "number", title: "Lux Report Interval: (0-1440 minutes) [default 10]",
defaultValue: 10, range: "0..1440", required: false)
    }

        tiles(scale: 2) {
                multiAttributeTile(name:"switch", type: "lighting", width: 6, height: 4, canChangeIcon: true)
{
                        tileAttribute ("device.switch", key: "PRIMARY_CONTROL") {
                                attributeState "on", label:'${name}', action:"switch.off", icon:"st.Home.
home30", backgroundColor:"#79b821", nextState:"turningOff"
                                attributeState "off", label:'${name}', action:"switch.on", icon:"st.Home.
home30", backgroundColor:"#ffffff", nextState:"turningOn"
                                attributeState "turningOn", label:'${name}', action:"switch.off", icon:"st.
Home.home30", backgroundColor:"#79b821", nextState:"turningOff"
                                attributeState "turningOff", label:'${name}', action:"switch.on", icon:"st.
Home.home30", backgroundColor:"#ffffff", nextState:"turningOn"
                        }
            tileAttribute("device.status", key: "SECONDARY_CONTROL") {
                attributeState("default", label:'${currentValue}', unit:"")
            }
                }
        /*
        multiAttributeTile(name:"motion", type: "generic", width: 6, height: 4, canChangeIcon: false){
                        tileAttribute ("device.motion", key: "PRIMARY_CONTROL") {
                                attributeState "inactive",
                                        label:'No Motion',
                                        icon:"st.motion.motion.inactive",
                                        backgroundColor:"#cccccc"
                                attributeState "active",
                                        label:'Motion',
                                        icon:"st.motion.motion.active",
                                        backgroundColor:"#00a0dc"
                        }
                }
        */
                /*
                valueTile("motion", "device.motion", inactiveLabel: false, width: 2, height: 2) {
                        state "motion", label:'${currentValue}'
                }
        */

        standardTile("motion", "device.motion", inactiveLabel: true, decoration: "flat", width: 2, height:
2) {
                        state "inactive",  icon: "st.motion.motion.inactive", label: 'No Motion'
                        state "active",  icon: "st.motion.motion.active", label: 'Motion'
```

```
                }


                valueTile("illuminance", "device.illuminance", inactiveLabel: false, width: 2, height: 2) {
                        state "illuminance", label:'${currentValue} lux',unit:"lux"

                }

                standardTile("refresh", "device.switch", width: 2, height: 2, inactiveLabel: false,
decoration: "flat") {
                        state "default", label:'', action:"refresh.refresh", icon:"st.secondary.configure"
                }

        valueTile("firmwareVersion", "device.firmwareVersion", width:2, height: 2, decoration: "flat",
inactiveLabel: false) {
                        state "default", label: '${currentValue}'
                }


                main(["switch"])

                details(["switch","motion","illuminance","firmwareVersion", "refresh"])
        }
}


def parse(String description) {
        def result = null
    log.debug (description)
    if (description != "updated") {
             def cmd = zwave.parse(description, [0x20: 1, 0x26: 1, 0x70: 1])
        if (cmd) {
                    result = zwaveEvent(cmd)
            }
    }
    if (!result){
        log.debug "Parse returned ${result} for command ${cmd}"
    }
    else {
                log.debug "Parse returned ${result}"
    }
        return result
}




// Creates motion events.
def zwaveEvent(physicalgraph.zwave.commands.notificationv3.NotificationReport cmd) {
        log.debug "NotificationReport: ${cmd}"

        if (cmd.notificationType == 0x07) {
                switch (cmd.event) {
                        case 0:
                                log.debug "NO MOTION"
                createEvent(name:"motion", value: "inactive", isStateChange: true)
                                break
                        case 8:
                                log.debug "MOTION"
                                createEvent(name:"motion", value: "active", isStateChange: true)
                                break
                        default:
                                logDebug "Sensor is ${cmd.event}"
                }
        }
}


def zwaveEvent(physicalgraph.zwave.commands.sensormultilevelv5.SensorMultilevelReport cmd) {
        //log.debug("sensor multilevel report")
    //log.debug "cmd:  ${cmd}"
```

```
    def lval = cmd.scaledSensorValue

    createEvent(name:"illuminance", value: lval)
}


def zwaveEvent(physicalgraph.zwave.commands.manufacturerspecificv2.ManufacturerSpecificReport cmd) {
        log.debug "manufacturerId:   ${cmd.manufacturerId}"
        log.debug "manufacturerName: ${cmd.manufacturerName}"
    state.manufacturer=cmd.manufacturerName
        log.debug "productId:        ${cmd.productId}"
        log.debug "productTypeId:    ${cmd.productTypeId}"
        def msr = String.format("%04X-%04X-%04X", cmd.manufacturerId, cmd.productTypeId, cmd.productId)
        updateDataValue("MSR", msr)
    setFirmwareVersion()
    createEvent([descriptionText: "$device.displayName MSR: $msr", isStateChange: false])
}


def zwaveEvent(physicalgraph.zwave.commands.versionv1.VersionReport cmd) {
    //updateDataValue("applicationVersion", "${cmd.applicationVersion}")
    log.debug ("received Version Report")
    log.debug "applicationVersion:      ${cmd.applicationVersion}"
    log.debug "applicationSubVersion:   ${cmd.applicationSubVersion}"
    state.firmwareVersion=cmd.applicationVersion+'.'+cmd.applicationSubVersion
    log.debug "zWaveLibraryType:        ${cmd.zWaveLibraryType}"
    log.debug "zWaveProtocolVersion:    ${cmd.zWaveProtocolVersion}"
    log.debug "zWaveProtocolSubVersion: ${cmd.zWaveProtocolSubVersion}"
    setFirmwareVersion()
    createEvent([descriptionText: "Firmware V"+state.firmwareVersion, isStateChange: false])
}


def zwaveEvent(physicalgraph.zwave.commands.firmwareupdatemdv2.FirmwareMdReport cmd) {
    log.debug ("received Firmware Report")
    log.debug "checksum:       ${cmd.checksum}"
    log.debug "firmwareId:     ${cmd.firmwareId}"
    log.debug "manufacturerId: ${cmd.manufacturerId}"
    [:]
}


def zwaveEvent(physicalgraph.zwave.commands.switchbinaryv1.SwitchBinaryReport cmd) {
        log.debug ("received switch binary Report")
    createEvent(name:"switch", value: cmd.value ? "on" : "off")
}


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


def on() {

        delayBetween([
                        zwave.basicV1.basicSet(value: 0xFF).format(),
                        zwave.switchMultilevelV1.switchMultilevelGet().format()
        ],5000)
}


def off() {

        delayBetween([
                        zwave.basicV1.basicSet(value: 0x00).format(),
                        zwave.switchMultilevelV1.switchMultilevelGet().format()
        ],5000)
}
```

```
def poll() {
/*
zwave.commands.switchbinaryv1.SwitchBinaryGet
        zwave.switchMultilevelV1.switchMultilevelGet().format()
*/
}


def refresh() {
        log.debug "refresh() called"
    configure()
}




def setFirmwareVersion() {
   def versionInfo = ''
   if (state.manufacturer)
   {
      versionInfo=state.manufacturer+' '
   }
   if (state.firmwareVersion)
   {
      versionInfo=versionInfo+"Firmware V"+state.firmwareVersion
   }
   else
   {
     versionInfo=versionInfo+"Firmware unknown"
   }
   sendEvent(name: "firmwareVersion",  value: versionInfo, isStateChange: true, displayed: false)
}


def configure() {
   log.debug ("configure() called")

   sendEvent(name: "numberOfButtons", value: 12, displayed: false)
   def commands = []
   commands << setPrefs()
   commands << zwave.manufacturerSpecificV1.manufacturerSpecificGet().format()
   commands << zwave.versionV1.versionGet().format()
   delayBetween(commands,500)
}


def setPrefs()
{
   log.debug ("set prefs")
   def cmds = []


        if (onTime)
        {
            //def onTime = Math.max(Math.min(onTime, 720), 8)
                cmds << zwave.configurationV1.configurationSet(parameterNumber:1, size:2,
scaledConfigurationValue: onTime ).format()
        }
    if (luxDisableValue)
        {
            //def luxDisableValue = Math.max(Math.min(luxDisableValue, 200), 30)
                cmds << zwave.configurationV1.configurationSet(parameterNumber:2, size:2,
scaledConfigurationValue: luxDisableValue ).format()
        }
    if (luxReportInterval)
```

```
        {
            //def luxReportInterval = Math.max(Math.min(luxReportInterval, 1440), 0)
                cmds << zwave.configurationV1.configurationSet(parameterNumber:3, size:2,
scaledConfigurationValue: luxReportInterval).format()
        }



   //Enable the following configuration gets to verify configuration in the logs
   //cmds << zwave.configurationV1.configurationGet(parameterNumber: 7).format()
   //cmds << zwave.configurationV1.configurationGet(parameterNumber: 8).format()
   //cmds << zwave.configurationV1.configurationGet(parameterNumber: 9).format()
   //cmds << zwave.configurationV1.configurationGet(parameterNumber: 10).format()

   return cmds
}

def updated()
{
 def cmds= []
 cmds << setPrefs
 delayBetween(cmds, 500)
}
```