Driver offers input fields where users can enter their device number and if Celcius or Fahrenheit.

Instructions for Hubitat would be:

- create new driver in "<> **DRIVER CODE**" section and paste in this code
- go to DEVICES and "Add Vitual Device" and select "**pp-Code H&T Sensor**" from the driver list
- enter required fields
- add tile on dashboard selectinh *humidity* and/or *temperature* from the available templates

This drive should work with SmartThings as well with some minor changes (were not tested with their hub though)

```
import groovy.json.*

metadata {
definition (
name: "pp-Code H&T Sensor",
namespace: "",
author: "",
importUrl: ""
)
{
capability "TemperatureMeasurement"
capability "RelativeHumidityMeasurement"
   capability "Sensor"
capability "Refresh" // refresh command
capability "Polling"
}


preferences {
input("ip", "string", title:"IP", description:"Sensor IP Address", defaultValue:"" , required: false,
displayDuringSetup: true)
input("deviceSerial", "string", title:"Serial", description:"Device Serial Number", defaultValue:""
, required: false, displayDuringSetup: true)

   input "tempFormat", "enum", title: "Temperature Format", required: false, defaultValue: false,
//RK
options: [F:"Fahrenheit",C:"Celcius"] //RK

   input "refreshEvery", "enum", title: "Enable auto refresh every XX Minutes", required: false,
defaultValue: false, //RK
```

```
options: [5:"5 minutes",10:"10 minutes",15:"15 minutes",30:"30 minutes"] //RK
input "locale", "enum", title: "Choose refresh date format", required: true, defaultValue: true,
//RK
options: [US:"US MM/DD/YYYY",UK:"UK DD/MM/YYYY"] //RK
input name: "debugOutput", type: "bool", title: "Enable debug logging?", defaultValue: true
input name: "txtEnable", type: "bool", title: "Enable descriptionText logging", defaultValue: true
//RK
}
}

def installed() {
   log.debug "Installed"
   refresh()
}

def updated() {
   log.debug "Updated"
   log.info "Preferences updated..."
   log.warn "Debug logging is: ${debugOutput == true}"
   unschedule()
   // RK start
   if (refreshEvery != null) {
   "runEvery${refreshEvery}Minutes"(autorefresh)
   log.info "Refresh set for every ${refreshEvery} Minutes"
   } else {
   runEvery30Minutes (autorefresh)
   log.info "Refresh set for every 30 Minutes"
   }
   if (debugOutput) runIn(1800,logsOff)
   state.LastRefresh = new Date().format("YYYY/MM/dd \n HH:mm:ss", location.timeZone)
   // RK stop
   refresh()
}

def parse(description) {
   logDebug "Parsing result $description"

   def msg = parseLanMessage(description)
   def headersAsString = msg.header // => headers as a string
   def headerMap = msg.headers      // => headers as a Map
   def body = msg.body              // => request body as a string
   def status = msg.status          // => http status code of the response
   def data = msg.data    // => either JSON or XML in response body (whichever is specified by
content-type header in response)

   String myTemp = data.Stats.Temp.replaceAll("[F]","");
```

```
    String myHumi = data.Stats.Humi.replaceAll("[%]","")

    myTemptrimmed = myTemp.substring(0, myTemp.indexOf('.'));
    myHumitrimmed = myHumi.substring(0, myHumi.indexOf('.'));

    int myTempint = Integer.parseInt(myTemptrimmed);

    if (tempFormat == "C") TempResult = (myTempint - 32) * 5 / 9 else TempResult =
myTempint


      TempResult = Math.round(TempResult)

      if (tempFormat == "C") TempResult = TempResult + " C" else TempResult = TempResult  +
" F"


    HumiResult = myHumitrimmed + " %"

    sendEvent(name: "temperature", value: TempResult)
    sendEvent(name: "humidity", value: HumiResult)

}

def ping() {
logDebug "ping"
poll()
}

def initialize() {
log.info "initialize"
if (txtEnable) log.info "initialize" //RK
refresh()
}

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

def autorefresh() {
if (locale == "UK") {
if (debugOutput) log.info "Get last UK Date DD/MM/YYYY"
state.LastRefresh = new Date().format("d/MM/YYYY \n HH:mm:ss", location.timeZone)
sendEvent(name: "LastRefresh", value: state.LastRefresh, descriptionText: "Last refresh
performed")
```

```
}
if (locale == "US") {
if (debugOutput) log.info "Get last US Date MM/DD/YYYY"
state.LastRefresh = new Date().format("MM/d/YYYY \n HH:mm:ss", location.timeZone)
sendEvent(name: "LastRefresh", value: state.LastRefresh, descriptionText: "Last refresh
performed")
}
if (txtEnable) log.info "Executing 'auto refresh'" //RK
refresh()

}

def refresh() {
    logDebug "Refresh - Getting Status"
    def myPath = "/" + deviceSerial + "&Stats/json"
    sendHubCommand(new hubitat.device.HubAction(
      method: "GET",
      path: myPath,
      headers: [
        HOST: getSensorAddress(),
        "Content-Type": "application/x-www-form-urlencoded"
       ]
    ))
state.LastRefresh = new Date().format("MM/d/YYYY \n HH:mm:ss", location.timeZone)
state.tempFormat
}

private logDebug(msg) {
if (settings?.debugOutput || settings?.debugOutput == null) {
log.debug "$msg"
}
}

// handle commands
//RK Updated to include last refreshed
def poll() {
if (locale == "UK") {
if (debugOutput) log.info "Get last UK Date DD/MM/YYYY"
state.LastRefresh = new Date().format("d/MM/YYYY \n HH:mm:ss", location.timeZone)
sendEvent(name: "LastRefresh", value: state.LastRefresh, descriptionText: "Last refresh
performed")
}
if (locale == "US") {
if (debugOutput) log.info "Get last US Date MM/DD/YYYY"
state.LastRefresh = new Date().format("MM/d/YYYY \n HH:mm:ss", location.timeZone)
sendEvent(name: "LastRefresh", value: state.LastRefresh, descriptionText: "Last refresh
```

```
performed")
}
if (txtEnable) log.info "Executing 'poll'" //RK
refresh()
}

private getSensorAddress() {
    def port = 80
    def iphex = ip.tokenize( '.' ).collect { String.format( '%02x', it.toInteger() )
}.join().toUpperCase()
    def porthex = String.format('%04x', port.toInteger())
    def sensorAddress = iphex + ":" + porthex
    if (txtEnable) log.info "Using IP " + ip + ", PORT 80 and HEX ADDRESS " + sensorAddress
+ " for device: ${device.id}"
return sensorAddress
}

private dbCleanUp() {
unschedule()
}
```