Adding Custom Decoders and Sensor Definitions to BACnet

Reference Guide

www.multitech.com

Adding Custom Decoders and Sensor Definitions to BACnet

Part Number: S000825 Rev. 1.0

Copyright

Trademarks and Registered Trademarks

Legal Notices

Contacting MultiTech

Sales	Support
sales@multitech.com	support@multitech.com
+1 (763) 785-3500	+1 (763) 717-5863

Website

https://www.multitech.com

Support Portal

To create an account and submit a support case directly to our technical support team, visit: https://support.multitech.com.

Warranty

To read the warranty statement for your product, visit https://www.multitech.com/legal/warranty.

World Headquarters

Multi-Tech Systems, Inc.

2205 Woodale Drive, Mounds View, MN 55112

USA

Contents

1 – Adding Custom Decoders and Sensor Definitions to BACnet

Introduction

By default, the system supports MultiTech/Radio Bridge, Adeunis, and Elsys LoRaWAN sensors. The following are embedded into the firmware and cannot be modified or deleted by the user:

- Sensor definitions
- A sensor definition JSON file
- A sensor decoder file

This reference guide explains how to import sensor definitions, which allows adding new sensors or importing a custom sensor definition.

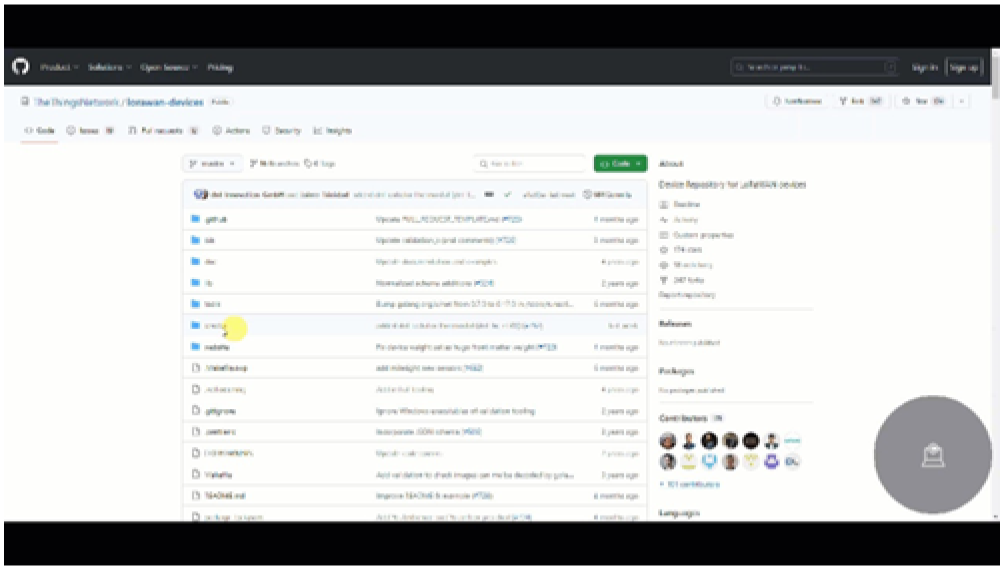
Requirements

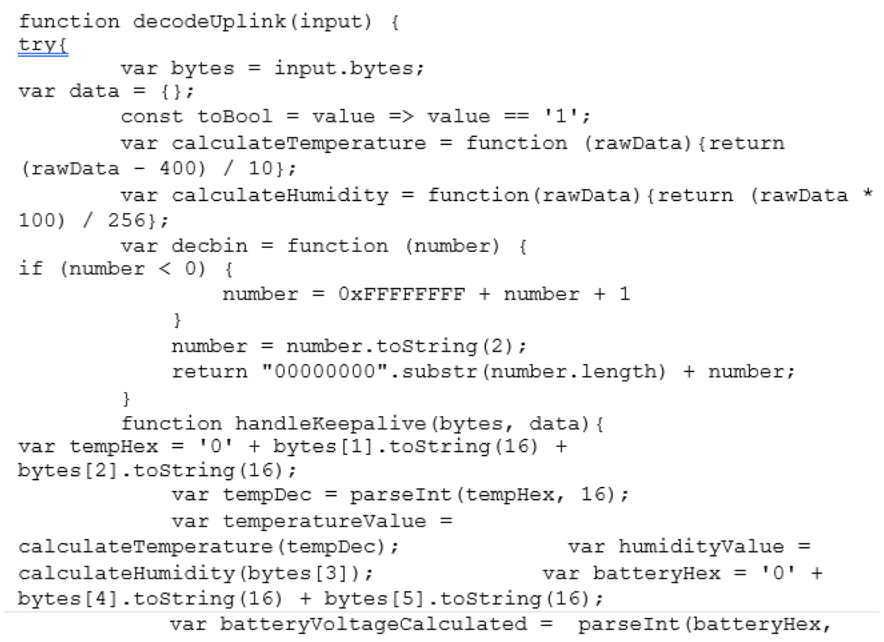
- MultiTech Gateway with Payload Management License with mPower 6.3.2 or higher.
- LoRaWAN sensor with JavaScript decoder.
- A Conduit gateway with mPower OS v6.3.2 or higher that has been optimized to use the standard of TTN JavaScript decoders from their device repository located in GitHub.

Downloading a JavaScript Decoder from the TTN Repository

If your sensor is included in the device repository you can download the decoder from the TTN GitHub and use that decoder to create a sensor definition file. Complete these steps:

1. Go to the device repository, find your sensor manufacturer and model, and download the file.
2. MultiTech/Radio Bridge Sensor decoders are pre-installed on our devices. You can use radio_bridge_packet_decoder.js if you need to modify it.


TheThingsNetwork / lorawan-devices
Code
About
Device Repository for LoRaWAN devices



```
function decodeUplink(input) {
try{
        var bytes = input.bytes;
var data = {};
        const toBool = value => value == '1';
        var calculateTemperature = function (rawData){return
(rawData - 400) / 10};
        var calculateHumidity = function(rawData){return (rawData *
100) / 256};
        var decbin = function (number) {
if (number < 0) {
                number = 0xFFFFFFFF + number + 1
            }
            number = number.toString(2);
            return "00000000".substr(number.length) + number;
        }
        function handleKeepalive(bytes, data){
var tempHex = '0' + bytes[1].toString(16) +
bytes[2].toString(16);
            var tempDec = parseInt(tempHex, 16);
            var temperatureValue =
calculateTemperature(tempDec);            var humidityValue =
calculateHumidity(bytes[3]);            var batteryHex = '0' +
bytes[4].toString(16) + bytes[5].toString(16);
            var batteryVoltageCalculated =  parseInt(batteryHex,
```

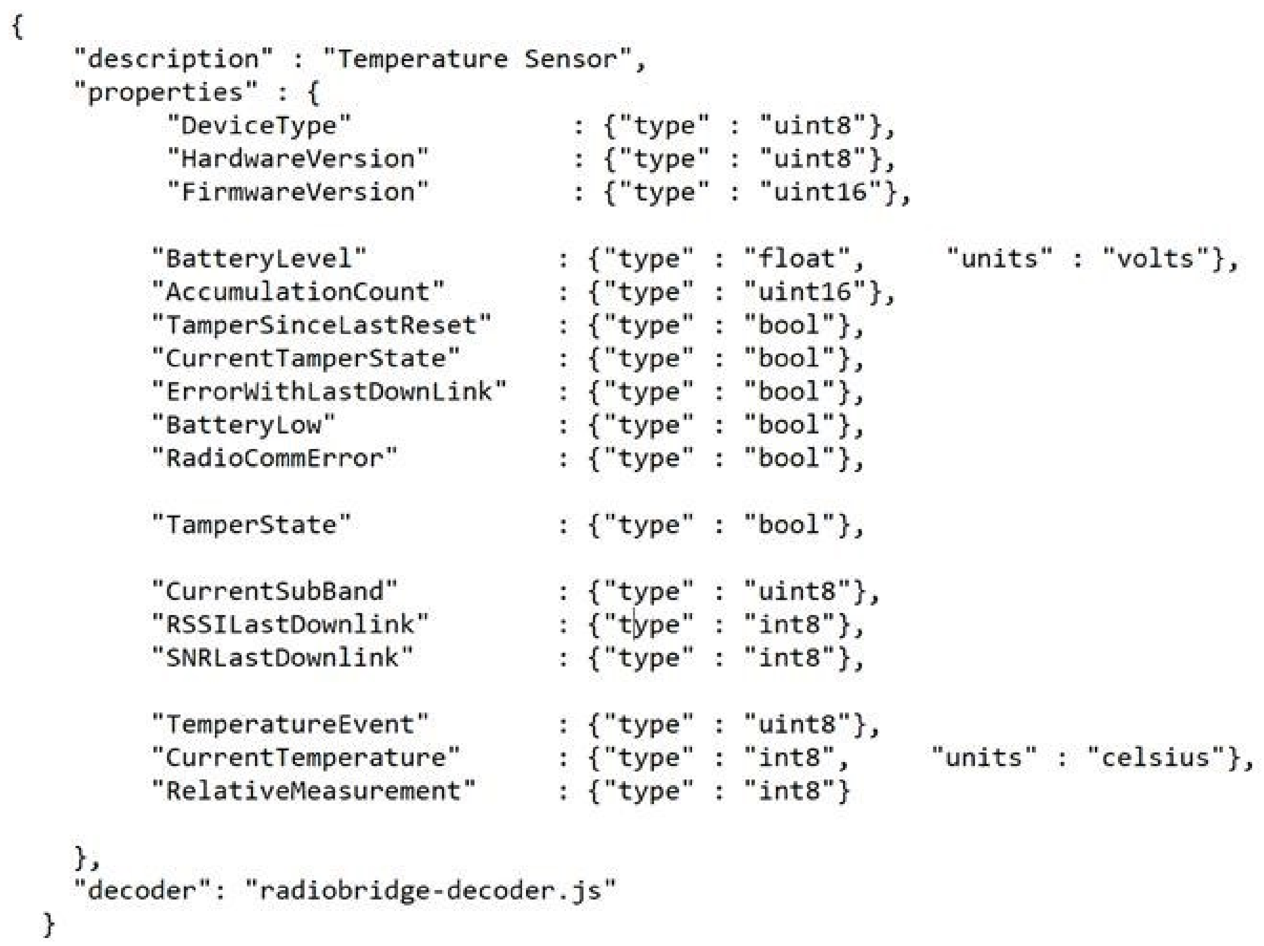
Creating a Sensor Definitions File

After downloading your JavaScript decoder either from the TTN repository or directly from the sensor manufacturer, create a sensor definitions file. This is a JSON file that defines all the data types that the decoder reads from the sensor.

This example uses the mClimate CO2 Display decoder downloaded from the Downloading a JavaScript Decoder from the TTN Repository section.

1. From the code snippet in the previous section, note the sections where the function returns **data**. These are the specific data points that are added to our sensor definitions file.

   This is the JSON sensor definitions file created for the CO2 Display sensor:



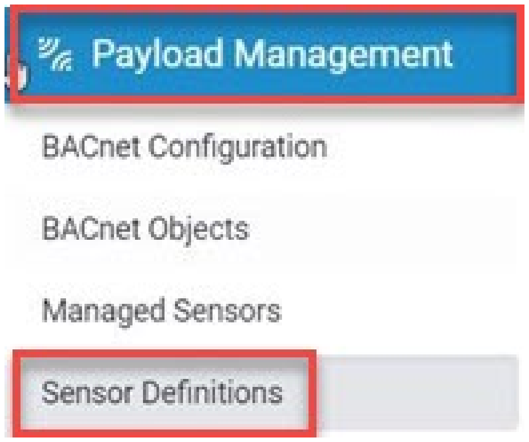
```
{
    "description" : "Temperature Sensor",
    "properties" : {
          "DeviceType"              : {"type" : "uint8"},
          "HardwareVersion"         : {"type" : "uint8"},
          "FirmwareVersion"         : {"type" : "uint16"},

        "BatteryLevel"            : {"type" : "float",     "units" : "volts"},
        "AccumulationCount"       : {"type" : "uint16"},
        "TamperSinceLastReset"    : {"type" : "bool"},
        "CurrentTamperState"      : {"type" : "bool"},
        "ErrorWithLastDownLink"   : {"type" : "bool"},
        "BatteryLow"              : {"type" : "bool"},
        "RadioCommError"          : {"type" : "bool"},

        "TamperState"             : {"type" : "bool"},

        "CurrentSubBand"          : {"type" : "uint8"},
        "RSSILastDownlink"        : {"type" : "int8"},
        "SNRLastDownlink"         : {"type" : "int8"},

        "TemperatureEvent"        : {"type" : "uint8"},
        "CurrentTemperature"      : {"type" : "int8",     "units" : "celsius"},
        "RelativeMeasurement"     : {"type" : "int8"}

    },
    "decoder": "radiobridge-decoder.js"
  }
```

2. Sensor definitions require these objects:
   - **description:** A description of the sensor.
   - **properties:** A list of all the properties the sensor returns including their name, type, and units if applicable.
   - **decoder:** The name of the JavaScript decoder file.
3. Ensure the following when creating the sensor definitions file:

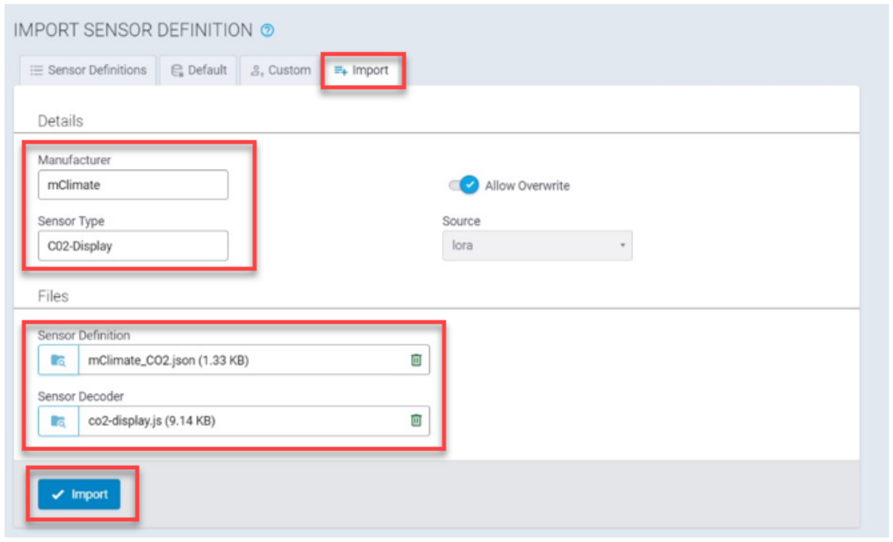
   a. The name of the properties must match the spelling and case of the name in the decoder.
   b. Ensure the data type returned corresponds to the decoder file. For example, if you list **data.hardwareVersion** as a float, make sure that in the decoder that value is a number. You may have to adjust your decoder.
   c. The decoder name must match the file name of the JavaScript decoder.

Uploading the Sensor Decoder and Definitions Files

1. Select **Payload Management** > **Sensor Definitions**.



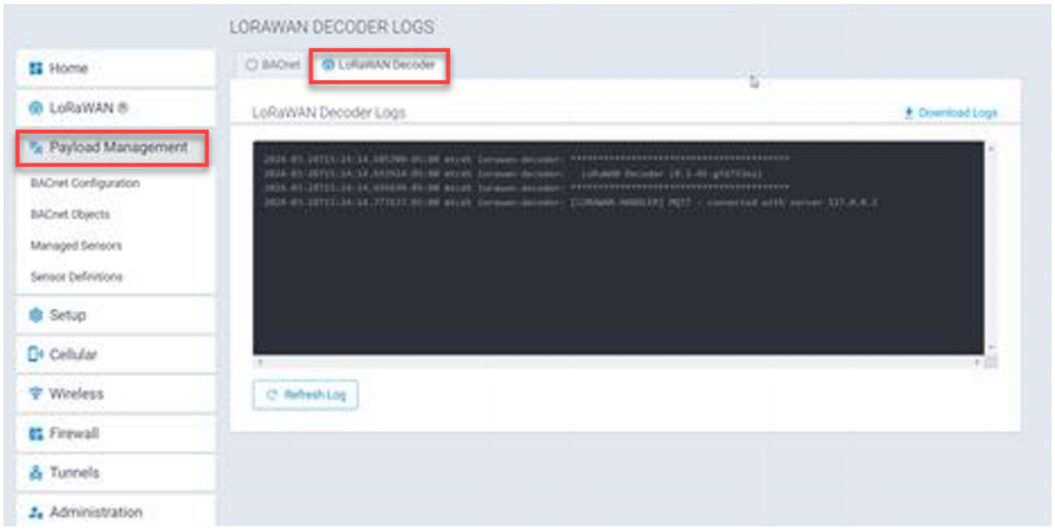
2. Go to the **Import** tab**.**
3. Enter the Manufacturer and Sensor Type.
4. Choose the **Sensor Definition** and **Sensor Decoder** files from your machine.
5. Click **Import**.



Manufacturer: mClimate
Sensor Type: C02-Display
Allow Overwrite
Source: lora
Sensor Definition: mClimate_CO2.json (1.33 KB)
Sensor Decoder: co2-display.js (9.14 KB)

Troubleshooting Sensor Definitions and Decoders From the UI

Error messages from the Payload Decoder can be found under **Status & Logs** > **Payload Management** under the LoRaWAN Decoder.



```
[illegible] lorawan-decoder: ****************************************
[illegible] lorawan-decoder:    LoRaWAN Decoder [illegible]
[illegible] lorawan-decoder: ****************************************
[illegible] lorawan-decoder: [LORAWAN HANDLER] MQTT - connected with server [illegible]
```

Testing Decoders Through SSH

If you have the programming experience, you can test the decoders directly on the operating system of the gateway:

1. Enable SSH by selecting **Administration** > **Access Configuration**.
2. SSH into the gateway.
3. Go to the /var/config/scada/sensors/ directory to access the custom decoder files.
4. Run the following command to test the decoder directly. Ensure that you test before uploading the decoder onto the device.

```
admin@mtcdt:~$ packet-decoder-cli \
      --js-uplink-decoder c02-didplay.js \
      --port 1 \ --packet 0102A33E0BE01A00000000 \
--stdout
```

2 – Revision History

Revision Number	Description
1.0	Initial release. June 2024