How to add Custom Decoders and Sensor Definitions

Introduction

By default, the system supports MultiTech/Radio Bridge, ADEUNIS and ELSYS LoRaWAN sensors. The sensor definitions, a **sensor definition JSON file,** and a **sensor decoder file**, are embedded into the firmware and cannot be modified or deleted by the user.

The system also allows importing sensor definitions, which allows adding new sensors or importing a custom sensor definition. This guide will instruct you on the process.

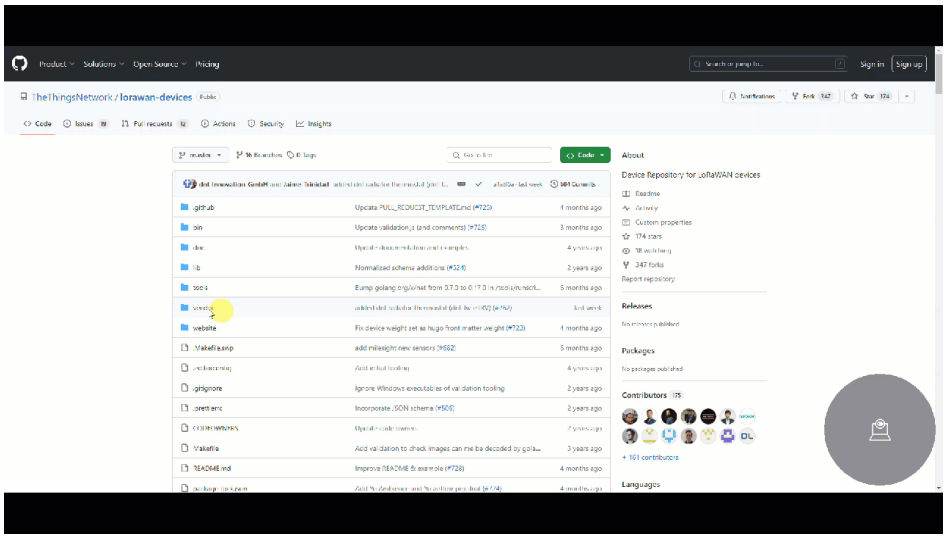
Requirement

- MultiTech Gateway with Payload Management License with mPower 6.3.2+
- 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. You can check out their sensor library here https://github.com/TheThingsNetwork/lorawan-devices

Downloading Decoder from TTN Repository

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

1. In the link above, navigate the repository to find your sensor manufacturer and model and download the file.
2. MultiTech/Radio Bridge Sensor decoders are pre-installed on our devices, if you'd like to modify it, they can be found here: lorawan-devices/vendor/radio-bridge/radio_bridge_packet_decoder.js at master · TheThingsNetwork/lorawan-devices · GitHub



 •

```
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);
```

Create Sensor Definitions File

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

In this example, we are using the mClimate CO2 Display decoder we downloaded from the step above. See the code snippet above.

Make note of the sections where the function returns '**data.**' These are the specific data points that are added to our sensor definitions file. Below is the JSON sensor definitions file created for the CO2 Display sensor above.

Sensor definitions require the following 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.

A few things to note when creating the sensor definitions file.

- The name of the properties must match the spelling and case of the name in the decoder.
- Make sure the data type returned corresponds to the decoder file. For ex., 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.
- The decoder name must match the file name of the JavaScript decoder.

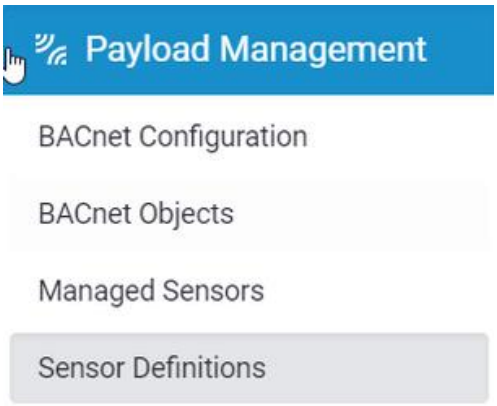
Below is a sample sensor definitions file for a Radio Bridge Temperature Sensor.

```
{
    "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"
  }
```

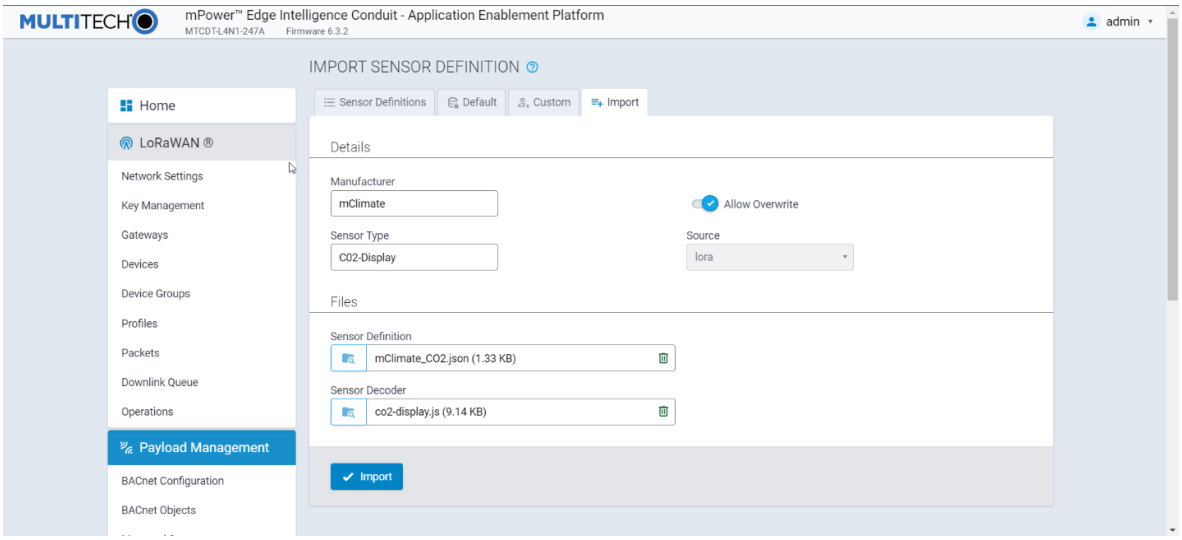
Upload Sensor Definitions Files

Now it's time to upload both files to the Gateway.

1. Navigate to **Payload Management**-->**Sensor Definitions**



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

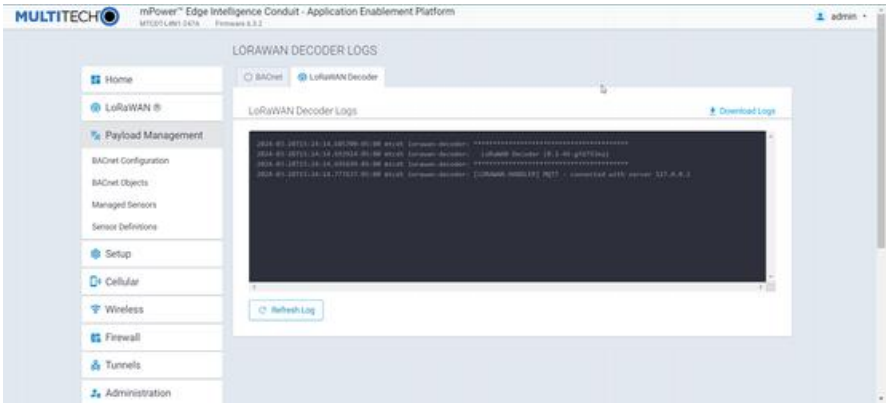


5. Click **Import**

Troubleshooting Sensor Definitions & Decoders

From the UI

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



Through SSH

For those with more programming experience, you can test the decoders directly on the operating system of the gateway. Enable SSH through **Administration → Access Configuration**

SSH into the gateway. You can find custom decoder files in the /var/config/scada/sensors/ directory.

 •

There you can run the following command below to test the decoder directly. We recommend testing before uploading the decoder onto the device.

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

