
Application Guide: Remote Setpoints

Version 42.1 authored by  Wim Verheirstraeten on 2024/04/29 15:25

Content

- Content
- Introduction
- Getting the device nodeId
- Sending a schedule
 - Datapoint structure & schedule properties
 - Limitations
 - Using the Eniris Python package (recommended)
 - API call (for other languages)
 - Reference schema for the data structure
- Configuring a device to use the signal

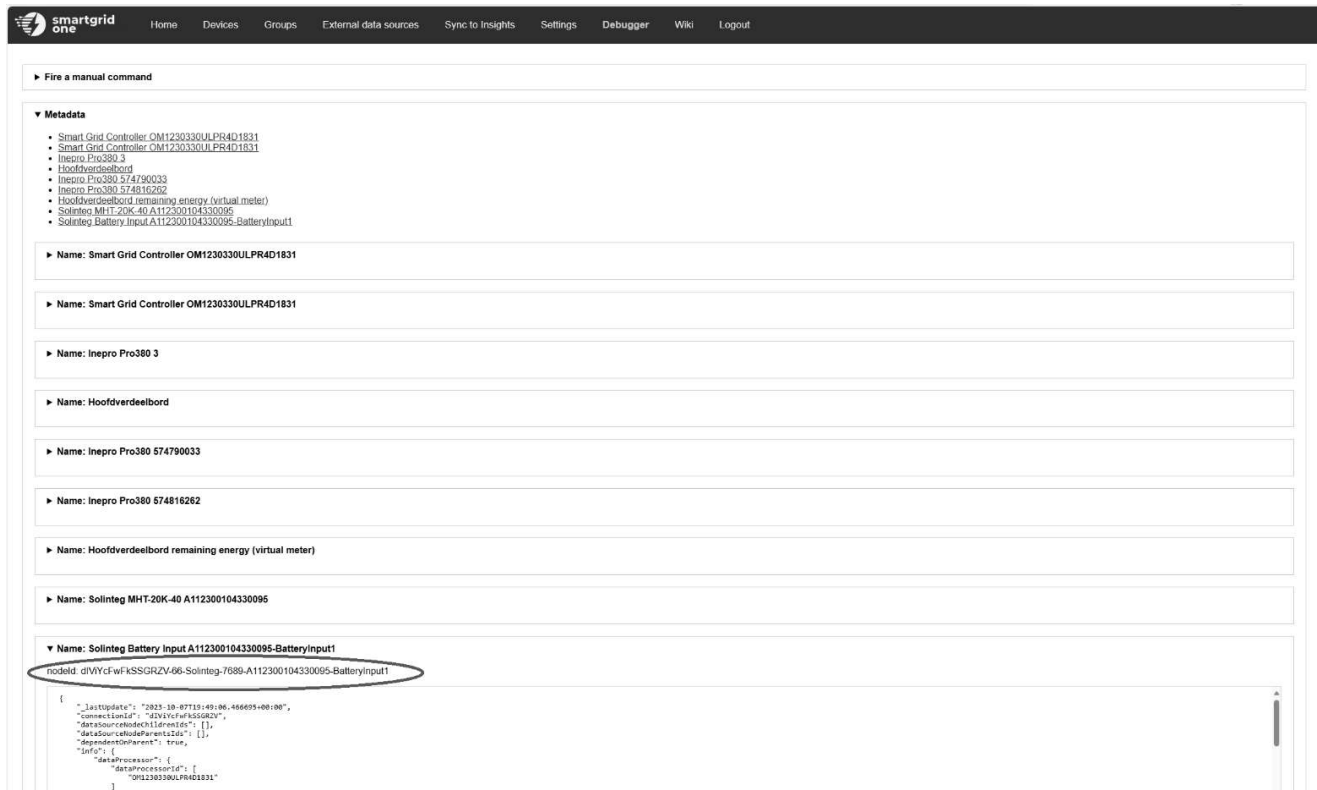
Introduction

Devices connected to the controller can be remotely controlled via the Eniris Insights API. The complete documentation of all the available API calls can be found at <https://api.eniris.be/docs/>. We recommend using Python and the Eniris pip package for handling of the authentication flow with the API. See the installation instructions and example at <https://pypi.org/project/eniris/>.

Getting the device nodeId

Each device in Insights has a nodeId associated that is used as an identifier for the device. The nodeId can be obtained from the controller commissioning interface from the debugger page (hidden page [http://\[IP ADDRESS CONTROLLER\]/debugger](http://[IP ADDRESS CONTROLLER]/debugger)), where all the device metadata is listed. See below for an example.

Make sure to use the nodeId of the device that will actually be controlled. See also the paragraph "Configuring a device to use the signal" below. This is important, because e.g. hybrid inverters may have different subdevices (storage and PV) that each can be controlled separately.



Alternatively, you can use the Insights API to retrieve all devices and filter on the device that you want to control.

Sending a schedule

The control signal is sent as a schedule (a sequence of datapoints) that indicates for each moment in the future which policy and/or setpoint to use. After the last datapoint, the control will go by default to local control, i.e. control on its own according to the local settings. See the description below and the Python example for how this is implemented.

Datapoint structure & schedule properties

Time

The time property of each datapoint follows the following convention:

- It must be in the ISO 8601 UTC format.
- Always in UTC, never in local time
- Sending a new datapoint with the same time as before will overwrite the previous datapoint. Use this property to update existing schedules.
- A datapoint is valid until the time of the datapoint; and from the time of the previous datapoint in the database.

It is highly recommended to use a regular interval in datapoint times. E.g. send datapoints that each are one minute, five seconds etc. spaced apart. Writing with irregular intervals or in between datapoints can lead to unexpected behaviour!

Fields

There are two fields in each datapoint:

- **policy**: An integer that indicates which policy the device should follow. The following values are allowed (see also the table below, as the allowed policies depend on the type of device):
 - 0: Local control by the controller; the remote control signal is to be ignored.
 - 1: Self-consumption; the end device will be in self consumption mode.
 - 2: Follow setpoint; the end device will follow the setpoint of the powerSetpoint_W field.
 - 3: Maximum injection limit on grid connection point (experimental). The smart grid controller curtails the solar installation to remain below this injection limit.
 - 10: On/off loads, heat pumps & boilers: Switch on. **WARNING**: This disregards any cooldown/min runtime/max runtime timing constraints.
 - 11: On/off loads, heat pumps & boilers: Switch off. **WARNING**: This disregards any cooldown/min runtime/max runtime timing constraints.
- **powerSetpoint_W**: A float that should be present in case the policy is 2 (Follow setpoint) and that indicates the setpoint for the device. Note, you **must** send data as **floats**, so e.g. 100 must be send as 100.0, 0 as 0.0 etc.

Allowed policies per type of device

| Policy | Storage (batteries) | EV chargers | On/off loads, heat pumps & boilers | PV production |
|--------|---------------------|-------------|------------------------------------|---------------|
| 0 | X | X | X | X |
| 1 | X | X | | X |
| 2 | X | X | | X |
| 3 | | | | X |
| 10 | | | X | |
| 11 | | | X | |

Measurement

Must always be remoteControlSignals.

Tags

Must always contain serialNr and nodeId as in the example above.

Limitations

Rate limits

- There may be at most 60 API calls per minute. With more API calls you may experience a backoff. Please bundle your datapoints in a single API call!
- Datapoints with a time longer than three days in the past are removed from the server.

Power/current limits

The controller will (in most cases) override the power setpoints if the installation's current or power limits would be exceeded.

Using the Eniris Python package (recommended)

See the installation instructions for the Eniris package at <https://pypi.org/project/eniris/>

```
from eniris import ApiDriver
from eniris.point import Point
from eniris.point.writer import (
    PointDuplicateFilter,
    BufferedPointToTelemessageWriter,
    DirectPointToTelemessageWriter
)
from eniris.telemessage.writer import DirectTelemessageWriter
from datetime import datetime, timezone

def example():
    apiUsername = "username" # Your Insights username
    apiPassword = "password" # Your Insights password
    collectorToken = "token" # Collector Token; request to Eniris together with the Insights accounts
    that must have the right to use it.

    # Create an API drivers and a point writer with the desired functionality
    driver = ApiDriver(apiUsername, apiPassword)
    writer = PointDuplicateFilter(
        BufferedPointToTelemessageWriter(
            DirectTelemessageWriter(
                authorizationHeaderFunction=driver.accesstoken,
                params={"u": collectorToken},
            ),
            lingerTimeS=1
        )
    )

    # Use the writer!
    namespace = {'database': 'SGC', 'retentionPolicy': 'rp_one_s'}
    tags = {
        "serialNr": "Serial number of the controller",
        "nodeId": "nodeId of the device or site you're controlling"}

    dt1 = datetime(year=2023, month=10, day=6, hour=12, minute=0, second=0, tzinfo=timezone.utc)
    fields1 = {'policy': 2, 'powerSetpoint_W': 2000.0}
    datapoint1 = Point(namespace, 'remoteControlSignals', dt1, tags, fields1)

    dt2 = datetime(year=2023, month=10, day=6, hour=12, minute=1, second=0, tzinfo=timezone.utc)
    fields2 = {'policy': 2, 'powerSetpoint_W': 3000.0}
    datapoint2 = Point(namespace, 'remoteControlSignals', dt2, tags, fields2)

    writer.writePoints([datapoint1, datapoint2])

    # Do not forget to flush before the program terminates!
    writer.flush()

example()
```

API call (for other languages)

We strongly recommend using the Eniris Python package instead of implementing your own API interface. Many problems you will encounter have been solved for you in this package so you can enjoy spending time on more fun things. 😊

Controlling a device is done by sending a schedule of future datapoints using the `/v1/telemetry` post call. A schedule consists of multiple datapoints that indicate which policy and/or setpoint the device should follow until a given time.

Endpoint: POST <https://neodata-ingress.eniris.be/v1/telemetry>

Special required headers:

- Authorization: A "Bearer" token, as documented via: <https://authentication.eniris.be/docs/>

Query parameters:

- `u`: Collector Token; request to Eniris together with the Insights accounts that must have the right to use the token.
- `db`: SGC
- `rp`: `rp_one_s`

The **body must** be sent in **Influx line**

protocol: https://docs.influxdata.com/influxdb/v1/write_protocols/line_protocol_reference/ and have content type **text**. Sending the body as a json is possible in some cases when there are no integers but it is not well supported due to the fact that json makes no difference between integers and floats, while the timeseries database does.

Reference schema for the data structure

The structure below is intended only as an example for visualising the datapoints textually. Avoid sending this as a json body to the API.

```
[
  {
    "measurement": "remoteControlSignals",
    "time": "2023-08-02T10:00:00Z",
    "tags": {
      "serialNr": "Serial number of the controller",
      "nodeId": "nodeId of the device you're controlling"
    },
    "fields": {
      "powerSetpoint_W": 2000.0,
      "policy": 1
    }
  },
  {
    "measurement": "remoteControlSignals",
    "time": "2023-08-02T11:00:00Z",
```

```

{
  "tags": [{
    "serialNr": "Serial number of the controller",
    "nodeId": "nodeId of the device you're controlling"
  }],
  "fields": {
    "powerSetpoint_W": 3000.0,
    "policy": 1
  }
}
]

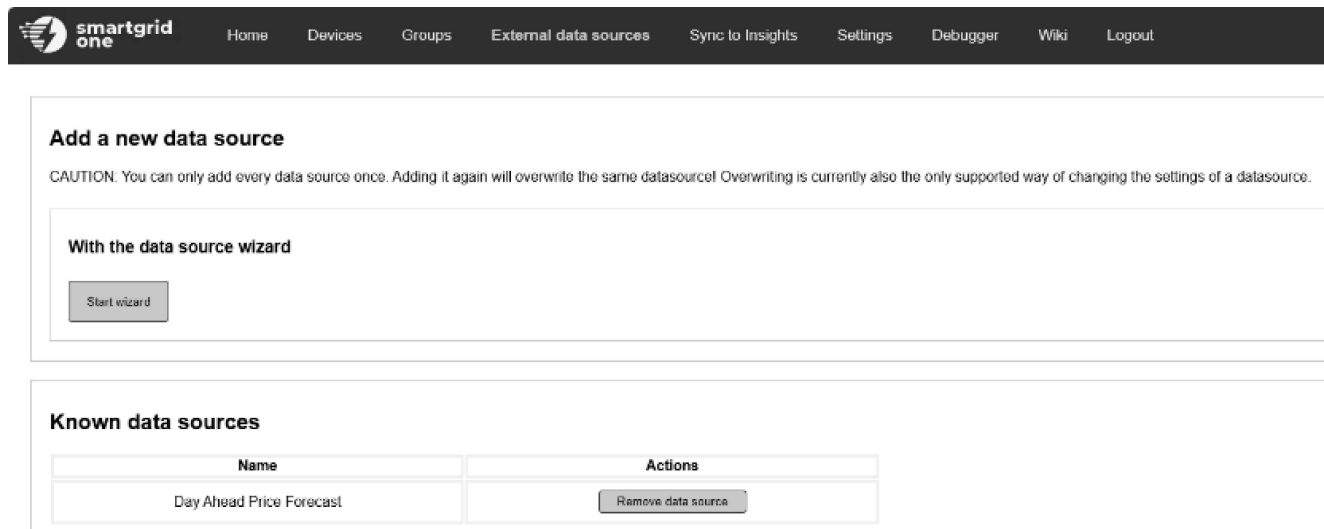
```

Configuring a device to use the signal

By default the controller does not check for external signals. You must enable the usage of the external signal in the controller and select the devices to which it applies.

If you don't see the data sources as mentioned below, you may need to purchase an additional license for activating remote control.

Go to the tab "External data sources", and click "Start wizard".



smartgrid one Home Devices Groups External data sources Sync to Insights Settings Debugger Wiki Logout

Add a new data source

CAUTION: You can only add every data source once. Adding it again will overwrite the same datasource! Overwriting is currently also the only supported way of changing the settings of a datasource.


With the data source wizard

Start wizard

Known data sources


| Name | Actions |
|--------------------------|--------------------|
| Day Ahead Price Forecast | Remove data source |

Select "External control signal"

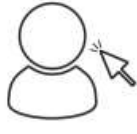


Home Devices Groups External data sources Sync to Insights Settings Debugger Wiki Logout

What kind of data source do you want to add?




Price data



External control signal


Select Eniris as provider.




Home Devices Groups External data sources Sync to Insights Settings Debugger Wiki Logout

Choose one of the supported providers

If your provider is not listed, contact info@eniris.be to request a new integration.



Select "Eniris API Device Control Signal".




Home Devices Groups External data sources Sync to Insights Settings Debugger Wiki Logout

Which data source do you want to use?

☒ Eniris API Device Control Signal

Next

Enter the collector token; this is the same collector token as used in the Python code to push schedules.



Home Devices Groups External data sources Sync to Insights Settings Debugger Wiki Logout

The data source requires some additional input


Enter the collector token (request to your distributor / device manufacturer):

Example: COMPANY_abcd1234

Submit

Select to which devices the external input applies.

NOTE: In the Python code to push schedules, use the nodeId of the device that corresponds with the selected device here!




Home Devices Groups External data sources Sync to Insights Settings Debugger Wiki Logout

To which devices does the external signal apply?

| Name | Manufacturer | Select |
|--|--------------|-------------------------------------|
| Solinteg Battery Input A112300104330095-BatteryInput1 | Solinteg | <input checked="" type="checkbox"/> |

Apply signal to selected devices

The data source has now been added. Reboot the controller to make it effective.



Home Devices Groups External data sources Sync to Insights Settings Debugger Wiki Logout

Data source was added.

Proceed