Invisible Systems

Realtime Online Core API V3

Overview

The Realtime Online API allows the automated retrieval of readings from devices which have been configured in Realtime Online, as well as providing means to query meta data about the sensors which are available.

Version 3 of the API adds full support for LoRaWAN and NBIoT devices.

To upgrade from API v2.x

- Upgrade your sensor_id storage to store Sensor IDs as strings instead of integers.
- Handle new error status response codes and messages.
- Handle 429 Usage Limit Exceeded error code.
- (optional) Implement Get Latest Record Date action.

Configuration Of Devices

A single transmitter has a single ID associated with it, but it may transmit data from more than one data point (such as temperature, humidity or pulse count) or from more than one source (such as multi-channel meters each taking a separate pulse count). In order to make use of a transmitter in Realtime Online, its ID must be entered in the *sensor setup* page, and one or more data points and/or channels must be selected. Only the selected channels will be available in Realtime Online and its API.

In Realtime Online, configured transmitters belong to a *system*, and systems are grouped in *organisations*.

Additional meta data can be associated with sensors within an organisation, using the *Sensor Export Fields* page

Anatomy Of A Request

Requests to the API are made over the HTTP protocol. An API request is encoded as JSON placed in the body of a HTTP POST request, and the HTTP response body will contain the JSON encoded API response.

URL

The API can be accessed through the following URL:

https://www.realtime-online.com/api/v3/json/

The second part of the path indicates the version of the API. This is treated as the MAJOR version as described by Semantic Versioning,
meaning that for a given version, changes will never be made such that a valid request or response becomes invalid.

Authentication

The API uses a token to uniquely identify the system or organisation being operated upon, and a secret shared key to authenticaterequests and responses. These are configured through the Realtime Online web interface.

The JSON body of a request might look something like this:

```
{
    "action": "getSensors",
    "request_date": "2020-02-04T11:59:28+00:00",
    "systems": [
        2571
    ]
}
```

You should have been informed when you asked for API credentials whether or not replay protection was enabled for them. Please contact our support team if you are unsure.

If replay protection is **enabled** for your credentials, authenticate the request using the following procedure:

1. The current date/time is included as **request_date** in the body of the request.
2. The body of the request is taken as a string.
3. The secret shared key is appended to this string.
4. The SHA-256 hash of the string is taken.
5. The hash is placed in the **X-RT2-API-Hash** header of the request, along with the token, as in the example below.

```
Content-Type: application/json
X-RT2-API-Token: db30b7e74e13
X-RT2-API-Hash:
f8119ac69dbe780e18f0d6ba31ab88a25a2b351025912508c2a91c8d91e69617
```

Note that:

- For hashing, the body of the request is treated as a string, i.e. the fact that it is JSON is ignored and no changes are made to its formatting.
- The current time must be included in the request to protect against replay attacks. The request will be rejected if the time in the request differs from the time on the server at the time the request is received, save for a small error margin.

If replay protection is **disabled** for your credentials you do not need to include the **X-RT2-API-Hash** header or the **request_date** in the body.

Responses

The API uses HTTP status codes in the response to indicate success or failure fulfilling the request. The status code is mirrored in the response body, and may be accompanied by a message detailing the reason for failure.

An example failure response:

```
{
    "status": 500,
    "message": "An unexpected error occurred processing the request.",
    "request_date": "2020-02-04T12:08:12+00:00"
}
```

The **request_payload** field may be present, containing the original request either as a string or as a JSON object.

An example success response:

```
{
    "status": 200,
    "request_date": "2020-02-04T12:08:12+00:00"
}
```

A successful response will also have fields containing the data that was requested. The fields required for specific requests, and those given in their responses, are detailed later in this document.

Types Of Request

The **action** field of the request object determines the type of the request, and what fields are required to be present in the request. All requests require this field.

The **request_date** field is only mandatory if replay protection is **enabled**.

Get Available Systems

```
{
    "action": "getSystems",
    "request_date": "2020-02-04T13:39:26+00:00"
}
```

The **getSystems** action requires no additional parameters.

```
{
    "action": "getSystems",
    "status": 200,
    "message": "OK",
    "request_date": "2020-02-04T13:39:26+00:00",
    "systems": [
        {
            "system_id": 2571,
            "name": "~ Test System ~",
            "timezone": "Europe/London",
            "sensors_count": 21
        }
    ]
}
```

The response includes the **systems** field, which is an array of objects, one for each system, with the following fields:

Field	**Meaning**
system_id	Uniquely identifies the system to Realtime Online.
name	The name as configured in Realtime Online.
timezone	The configured timezone of the system. This does not affect dates; they are all required to be in ISO

	8601 format and include a time offset.
sensors_count	The number of sensors which have been configured on the system.

Get Sensors

```
{
    "action": "getSensors",
    "request_date": "2020-02-04T13:50:13+00:00",
    "systems": [ 2571 ]
}
```

The **getSensors** action requires a single extra parameter: **systems**, an array of integers, which specifies the IDs of systems whose sensors should be retrieved.

```
{
    "action": "getSensors",
    "status": 200,
    "message": "OK",
    "request_date": "2020-07-22T15:44:30+02:00",
    "sensors": [
        {
            "sensor_id": "6110551",
            "system_id": 2571,
            "type_id": 5,
            "names": {
                "Battery": "6110551"
            },
            "units": {
                "Battery": "V"
            },
            "meta_data": {
                "Battery": {}
            }
        },
        {
            "sensor_id": "6322990",
            "system_id": 2571,
            "type_id": 108,
            "names": {
                "temperature": "New Sensor"
            },
            "units": {
                "temperature": ""
            },
            "meta_data": {
                "temperature": {}
            }
        },
        {
            "sensor_id": "6321402",
            "system_id": 2571,
            "type_id": 116,
            "names": {
                "channel_1_energy": "Export Test Threephase E1",
                "channel_1_current": "Export Test Threephase C1",
                "channel_2_energy": "Export Test Threephase E2",
                "channel_2_current": "Export Test Threephase C2",
                "channel_3_energy": "Export Test Threephase E3",
                "channel_3_current": "Export Test Threephase C3"
            },
            "units": {
                "channel_1_energy": "kWh",
                "channel_1_current": "A",
                "channel_2_energy": "kWh",
                "channel_2_current": "A",
                "channel_3_energy": "kWh",
                "channel_3_current": "A"
            },
            "meta_data": {
                "channel_1_energy": [],
                "channel_1_current": [],
                "channel_2_energy": [],
                "channel_2_current": [],
                "channel_3_energy": [],
                "channel_3_current": []
            }
        },
        {
            "sensor_id": "9704474",
            "system_id": 2571,
            "type_id": 1009,
            "names": {
                "temperature": "Blah Temperature",
                "door_status": "Blah Door",
                "door_countt": "Blah Door Count",
                "light_status": "Blah Light Status",
                "light_count": "Blah Light Count"
            },
            "units": {
                "temperature": "°C",
                "door_status": [
                    {
                        "value": 0,
                        "unit": "standby"
                    },
                    {
                        "value": 1,
                        "unit": "running"
                    }
                ],
                "door_count": "",
                "light_status": [
                    {
                        "value": 0,
                        "unit": "armed"
                    },
                    {
                        "value": 1,
                        "unit": "disarmed"
                    }
                ],
                "light_count": ""
            },
                "meta_data": {
                    "temperature": {
                        "custom_meta_field": "custom meta value"
                    },
                    "door_status": {},
                    "door_count": {},
                    "light_status": {},
                    "light_count": {}
                }
            }
        ]
}
```

The response includes the **sensors** field, which is an array of objects, one for each sensor, with the following fields:

Field	**Meaning**
sensor_id	The ID of the sensor, matching the device label. This is a string.
system_id	The ID of the system to which the sensor belongs.
type_id	Identifies the type of the sensor, which can be queried using the **getSensorTypes** action described below.
names	An object with a key for every data point configured in Realtime Online, whose value is the name configured therein.
units	An object with a key for every data point configured, whose value is either the name of the unit, or in the case of an enumerated type of data, an array of objects for each possible value that the data can take, giving the name for that value.
meta_data	An object with a key for every data point configured in Realtime Online, containing key value pairs for each custom meta field defined in *Sensor Export Fields*. The keys are derived from the names entered into Realtime Online by taking the lower

	case name string, and replacing spaces with underscores: *Custom Meta Field* becomes **custom_meta_field**.

Get Sensor Types

```
{
    "action": "getSensorTypes",
    "request_date": "2020-02-04T14:05:25+00:00"
}
```

The **getSensorTypes** request requires no extra parameters.

```
{
    "action": "getSensorTypes",
    "status": 200,
    "message": "OK",
    "request_date": "2020-02-04T14:05:25+00:00",
    "sensor_types": [
        {
            "type_id": 1,
            "name": "SmartRF Temperature",
            "fields": [
                "temperature",
                "temperature_min",
                "temperature_max"
            ]
        },
        {
            "type_id": 2,
            "name": "SmartRF Pulse/Status",
            "fields": [
                "status",
                "pulse_count"
            ]
        },
        {
            "type_id": 4,
            "name": "SmartRF 3-Channel Meter",
            "fields": [
                "channel_1_energy",
                "channel_2_energy",
                "channel_3_energy"
            ]
        },
        {
            "type_id": 113,
            "name": "SmartRF Dual Status/Pulse",
            "fields": [
                "channel_1_status",
                "channel_1_pulse_count",
                "channel_2_status",
                "channel_2_pulse_count"
            ]
        }
    ]
}
```

The response includes the **sensor_types** field, which is an array of objects with the following fields:

Field	Meaning
type_id	Uniquely identifies the transmitter type.
name	Describes the transmitter type.
fields	An array of strings listing all available data points that can be configured in Realtime Online.

Get Sensor Records

```
{
    "action": "getSensorRecords",
    "request_date": "2020-02-04T14:11:26+00:00",
    "sensors": [
        {
            "sensor_id": "6322905",
            "start_date": "2019-05-09T00:00:00+01:00",
            "end_date": "2019-05-09T01:00:00+01:00"
        },
        {
            "sensor_id": "6311678",
            "start_date": "2019-05-09T00:00:00+01:00",
            "end_date": "2019-05-09T01:00:00+01:00"
        }
    ]
}
```

The **getSensorRecords** request requires an array of systems, each containing its **system_id** and an array of **sensors** to request. Each sensor requires the following fields:

Field	Meaning
sensor_id	The ID of the sensor, as returned by **getSensors**.
start_date	The start of the period for which to request data from this sensor.

end_date	The end of the period for which to request data from this sensor.

```
{
    "action": "getSensorRecords",
    "status": 200,
    "message": "OK",
    "request_date": "2020-02-04T14:11:26+00:00",
    "systems": [
        {
            "system_id": 2571,
            "sensors": [
                {
                    "sensor_id": "6322905",
                    "start_date": "2019-05-09T00:00:00+01:00",
                    "end_date": "2019-05-09T01:00:00+01:00",
                    "names": {
                        "temperature": "Export Test Temp",
                        "humidity": "Export Test Humidity"
                    },
                    "units": {
                        "temperature": "°C",
                        "humidity": "%RH"
                    },
                    "data": [
                        {
                            "record_date": "2019-05-08T23:03:28+00:00",
                            "values": {
                                "temperature": 19.3453,
                                "humidity": 57.2858
                            }
                        },
                        {
                            "record_date": "2019-05-08T23:08:45+00:00",
                            "values": {
                                "temperature": 19.3453,
                                "humidity": 57.3163
                            }
                        },
                        {
                            "record_date": "2019-05-08T23:56:15+00:00",
                            "values": {
                                "temperature": 19.2488,
                                "humidity": 57.6368
                            }
                        }
                    ]
                },
                {
                    "sensor_id": "6311678",
                    "start_date": "2019-05-09T00:00:00+01:00",
                    "end_date": "2019-05-09T01:00:00+01:00",
                    "names": {
                        "channel_1_status": "6311678 Status 1",
                        "channel_1_pulse_count": "6311678 Pulse Count 1",
                        "channel_2_status": "6311678 Status 2",
                        "channel_2_pulse_count": "6311678 Pulse 2"
                    },
                    "units": {
                        "channel_1_status": [
                            {
                                "value": 0,
                                "unit": "off"
                            },
                            {
                                "value": 1,
                                "unit": "on"
                            }
                        ],
                        "channel_1_pulse_count": "kWh",
                        "channel_2_status": [
                            {
                                "value": 0,
                                "unit": "armed"
                            },
                            {
                                "value": 1,
                                "unit": "disarmed"
                            }
                        ],
                        "channel_2_pulse_count": "Unit"
                    },
                    "data": [
                        {
                            "record_date": "2019-05-08T23:03:48+00:00",
                            "values": {
                                "channel_1_status": 0,
                                "channel_2_status": 0
                            }
                        },
                        {
                            "record_date": "2019-05-08T23:11:48+00:00",
                            "values": {
                                "channel_1_status": 0,
                                "channel_2_status": 0
                            }
                        },
                        {
                            "record_date": "2019-05-09T00:00:00+00:00",
                            "values": {
                                "channel_1_pulse_count": 0,
                                "channel_2_pulse_count": 0
                            }
                        }
                    ]
                },
                {
                    "sensor_id": "6311678",
                    "start_date": "2019-05-09T00:00:00+01:00",
                    "end_date": "2019-05-09T01:00:00+01:00"
                }
            ]
        }
    ]
}
```

The response is also grouped into systems, inside which each sensor shares the **names** and **units** fields with the **getSensors** response. For each sensor, the **data** key is an array of objects, one object for each record.

It's important to note that each record will not necessarily contain a value for every data point that the sensor has. The most common situation in which this may occur is when a mixture of pulse count and non-pulse count points is configured for the sensor. The pulse counts will be converted into consumption data on fifteen minute intervals, while the non-pulse-count data will be returned for the intervals at which the sensor transmitted to the gateway.

Get Latest Record Date

```
{
    "action": "getLatestRecordDate",
    "request_date": "2020-02-04T14:11:26+00:00",
    "systems": [
        {
            "system_id": 2571,
            "sensors": [
                {
                    "sensor_id": "6322905"
                },
                {
                    "sensor_id": "6311678"
                }
            ]
        }
    ]
}
```

The **getLatestRecordDate** request requires an array of systems, each containing its **system_id** and an array of **sensors** to request.

```
{
    "action": "getLatestRecordDate",
    "status": 200,
    "message": "OK",
    "request_date": "2020-02-04T14:11:26+00:00",
    "systems": [
        {
            "system_id": 2571,
            "sensors": [
                {
                    "sensor_id": "6322905",
                    "latest_record_date": "2019-05-09T00:00:00+01:00"
                },
                {
                    "sensor_id": "6311678",
                    "latest_record_date": "2019-05-09T00:00:00+01:00"
                }
            ]
        }
    ]
}
```

The response is grouped into systems, inside which each sensor contains the following fields.

Field	Meaning
sensor_id	The ID of the sensor, matching the device label. This is a string.
latest_record_date	The date of the most recent record received from the sensor, in an ISO-8601 format. This field will be absent if status returns an error message (message not 'OK')

Note that for sensors which contain pulse records, which are calculated across 15 minute intervals, the latest record date will be the earlier of the last transmission date and the end of the previous 15 minute window.

Note also that the response is independent of the supplied **request_date**.

Appendix

Error Codes

The following is a list of the typical error codes that may be given by the API.

Status	Message	Explanation
400	Payload Empty	The payload contained no data at all.
401	Authentication failed	Either the token did not match any configured in Realtime Online, or the hash that the server calculated from the request and the secret key did not match the one provided in the HTTP header.
401	Missing hash header	The HTTP header specifying the hash of the payload and secret key was missing from the request (or possibly misspelt).
401	Missing token header	The HTTP header specifying the API token was missing from the request.

403	The specified request date is in the future	The **request_date** field specifies a date that is too far ahead of the time on the server.
403	The specified request date is too old	The **request_date** field specifies a date that is too far behind the time on the server.
404	Unknown Action	The **action** field specifies an action which does not exist.
413	Payload Too Large	The payload was larger than the maximum permitted by the API: 2,000,000 bytes.
415	Could not decode payload '' as JSON:	The JSON decoder failed to decode the payload. The payload is given in the error message, so if this is empty there has been some error getting the payload into the body of the request.
415	Expected a JSON object, got some other type	The payload was valid JSON but it wasn't an object. All requests are required to be an object.
429	Exceeded number of requests per day	Usage of the API for the given token is being throttled, because too many requests have been made in a rolling 24 hour window.
429	Exceeded number of requests per ten minutes	Usage of the API for the given token is being throttled, because too many requests have been made in a rolling 10 minute window.

In addition to this, a 400 error might also include a message indicating a required parameter which has been missed, or does not correspond to a configured system, or sensor, etc. This will be detailed in the optional Failure Response, if present.

Failure Response

The below is an example error response for a 404 error. This error occurs because the requested action is not recognised by the API.

```
{
    "status": 404,
    "code": 1,
    "message": "Unknown Action",
    "request_date": "2020-10-13T16:52:10+01:00",
    "request_payload": {
        "action": "fakeAction",
        "request_date": "2020-10-13T17:52:10+02:00",
        "systems": [
            {
                "system_id": 2571,
                "sensors": [
                    {
                        "sensor_id": "11000005800001",
                        "start_date": "2020-09-21T07:27:00+00:00",
                        "end_date": "2020-09-21T15:59:00+00:00"
                    }
                ]
            }
        ]
    }
}
```

The 400 error below occurs because part of the requested data was not available (because one of the requested sensors is not available on this system - an invalid sensor ID is passed).

Note that the error response follows the same structure as the original request.

```
{
    "status": 400,
    "message": "Failed with errors",
    "request_date": "2020-10-13T16:52:10+01:00",
    "request_payload": {
        "action": "getRecords",
        "request_date": "2020-10-13T17:52:10+02:00",
        "systems": [
            {
                "system_id": 2571,
                "sensors": [
                    {
                        "sensor_id": "1",
                        "start_date": "2020-09-21T07:27:00+00:00",
                        "end_date": "2020-09-21T15:59:00+00:00"
                    },
                    {
                        "sensor_id": "11000005800001",
                        "start_date": "2020-09-21T07:27:00+00:00",
                        "end_date": "2020-09-21T15:59:00+00:00"
                    }
                ]
            }
        ]
    },
    "systems": [
        {
            "system_id": 2571,
            "code": 0,
            "message": "OK",
            "detail": "",
            "sensors": [
                {
                    "sensor_id": "1",
                    "code": 30,
                    "message": "Sensor does not exist or is not
accessible",
                    "detail": ""
                },
                {
                    "sensor_id": "11000005800001",
                    "code": 0,
                    "message": "OK",
                    "detail": ""
                }
            ]
        }
    ]
}
```

The 400 error below occurs because part of the requested data was not available (because one of the requested systems is not available to this API key).

Note that the response does not contain any sensors for the failed system. The server does not process a given object any further once it encounters errors.

```
{
    "status": 400,
    "message": "Failed with errors",
    "request_date": "2020-10-13T16:52:10+01:00",
    "request_payload": {
        "action": "getRecords",
        "request_date": "2020-10-13T17:52:10+02:00",
        "systems": [
            {
                "system_id": 9999999999999,
                "sensors": [
                    {
                        "sensor_id": "6322905",
                        "start_date": "2020-09-21T07:27:00+00:00",
                        "end_date": "2020-09-21T15:59:00+00:00"
                    }
                ]
            },
            {
                "system_id": 2571,
                "sensors": [
                    {
                        "sensor_id": "6322905",
                        "start_date": "2020-09-21T07:27:00+00:00",
                        "end_date": "2020-09-21T15:59:00+00:00"
                    }
                ]
            }
        ]
    },
    "systems": [
        {
            "system_id": 9999999999999,
            "code": 20,
            "message": "System does not exist or is not accessible",
            "detail": ""
        },
        {
            "system_id": 2571,
            "code": 0,
            "message": "OK",
            "detail": "",
            "sensors": [
                {
                    "sensor_id": "6322905",
                    "code": 0,
                    "message": "OK",
                    "detail": ""
                }
            ]
        }
    ]
}
```

Each object present in the response will contain **code**, **message** and **detail**, which correspond to the table below. It will also contain as much of the original request as possible, usually including, for example, **system_id** or **sensor_id**.

If the object has a code other than 0 - i.e. a failure code, any nested arrays or objects will be absent from the failure response.

The **detail** field contains human-readable error detail to give further context to the error codes, for example to note which parameter is missing/invalid.

Code	**Message**	**Explanation**
0	OK	The component did not return any error
1	Invalid	The requested component does not exist or is not available. This is a generic error.
10	Required parameter missing	A required section of the request body is missing. See detail for more information.
11	Parameter type is invalid	The supplied parameter type does not match the required type.
20	System does not exist or is not accessible	The requested system does not exist, or cannot be accessed with these API credentials.

30	Sensor does not exist or is not accessible	The requested sensor does not exist on the requested system / with these API credentials.

Note that future versions of this API may add additional failure response codes and associated failure response messages. Code 0 (OK) will be the only success code.

Additional Resources

Demonstration implementations of this API are available from Invisible Systems Ltd. on request.

A PHP implementation should usually be bundled with this document.

Additionally, a minimal example written in Javascript is available on JS Fiddle

A minimal python (Python 3) implementation is included below. This can be run out of the box on Google Colab.

```
#pip3 install requests

import requests
import json
import hashlib
import datetime

url = "https://www.realtime-online.com/api/v3/json/"
token = "134ee7b730bd"
secretKey = "asdf5%123456"

# Form the request, in this example, getting all sensors for the listed
system IDs
request_body = {
  "action": "getSensors",
  "request_date": datetime.datetime.now().isoformat(),
  "systems": ["2571"]
}

# As specified in the Authentication section
magicString = json.dumps(request_body) + secretKey
contentHash = hashlib.sha256(magicString.encode()).hexdigest()
headers = {
    "X-RT2-API-Token": token,
    "X-RT2-API-Hash": contentHash
}

resp = requests.post(url, data=json.dumps(request_body), headers=headers)

jsonResp = json.loads(resp.text)
print(json.dumps(jsonResp, indent=4))
```