# HOBOlink® Web Services V3 Developer's Guide




1-508-759-9500 (U.S. and International)
1-800-LOGGERS (564-4377) (U.S. only)
www.onsetcomp.com/support/contact





25113-B

# Introduction

This guide describes how external clients can access different REST endpoints. Onset exposes several web services that provide users with the ability to view loggers and retrieve data from loggers. The current version of the REST API expands upon previous versions (2.0 and 2.5) in that it uses OAuth to grant access. Below are details on how to generate tokens for the web service as well as how to access different web services and what they provide.

# Required Technical Capabilities

Using Onset's web services require the following technical capabilities:

- Web development experience
- REST web services client development
- OAuth familiarity, particularly client credentials grants
- Access to a HOBOlink account and a Remote Monitoring Station and/or an MX series logger

# OAuth Token

Onset supports the client_credentials grant types in the public API at this point:

- Grant: Client Credentials - defined in rfc6749 section 4.4
  - Required form fields: grant_type=client_credentials, client_id, client_secret

The endpoint to generate the tokens is POST /auth/token. Below is an example request, with the details passed in the body of the POST request:

**Example Token Info Request**

```
POST /ws/auth/token HTTP/1.1
Host: webservice.hobolink.com
Accept: application/json
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials&client_id=Client+Name&client_secret=xxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxx
```

The client_id and client_secret should be replaced by the client-specific values provided by Onset.

The production API JSONDoc sandbox can be found at https://webservice.hobolink.com/ws/auth/info/index.html# . Navigate to Get Documentation → API → Token→ /token . Enter the Body object in the format specified above (grant_type=client_credentials&...) and Submit to see the token response.

## Success

The response from this endpoint is defined based on rfc6749 section 5.1 and is as follows:

*Status Code*: 200 OK

*Content-Type*: application/json

*Additional Required Headers*: Cache-Control: no-store, Pragma: no-cache

Body: JSON

- access_token - issued access token from OAuth server
- token_type - "bearer"
- expires_in - lifetime of token in seconds

The "access_token" field is the token that is included in authentication headers on requests or entered in JSONDoc endpoints that require a token for testing purposes.

Note that tokens are only valid for a limited period of time. When a token is expired, the client will receive 401 Unauthorized responses to web service requests. Your client should build in handling to get a new token when a token expires and becomes invalid.

**Example Token Success**

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store, no-transform
Pragma: no-cache

{
  "access_token":"44a11681c637fdb512764bb6b0965e44",
  "token_type":"bearer",
  "expires_in": 600,
}
```

## Failure

Error response is defined based on rfc6749 section 5.2 and is as follows:

*Status Code*: 400 Bad Request, 401 Unauthorized

*Content-Type*: application/json

*Additional Required Headers*: Cache-Control: no-store, Pragma: no-cache

*Body*: JSON

- error - ASCII error code limited to this list
  - invalid_request - missing/unsupported parameters (except grant type) or otherwise malformed (Status 400)
  - invalid_client - client authentication not included or authentication failed (Status 401)
  - invalid_grant - the grant parameters (resource owner credentials) are not valid or refresh token is not valid/expired/revoked (Status 400)
  - unsupported_grant_type - the grant type is not supported by the server (Status 400)
- error_description - Human-readable ASCII text used to provide additional understanding of error that occurred

**Example Token Failure**

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Cache-Control: no-store, no-transform
Pragma: no-cache

{
  "error_description": "The request is malformed, a required parameter is missing or a parameter
   has an invalid value.",
  "error": "invalid_request"
}
```

## Command Line Requests

If desired, it is possible to run the same request via a CLI using commands like curl. The format of the request and response will be the same as above. An example of a curl request for a client_credentials token would look something like this:

```
curl -i -H "Content-type: application/x-www-form-urlencoded" -H "Accept: application/json" -d
"grant_type=client_credentials" -d "client_id=Client+Name" -d
"client_secret=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" --data-urlencode -X POST
https://webservice.hobolink.com/ws/auth/token
```

In to order to control access to different endpoints and data, Onset uses an OAuth token privilege system. When a token is used to access an endpoint, the associated privileges are retrieved, and the request is passed to the endpoint only if the retrieved privileges contain the one required by the endpoint. If the privilege is not present, the user will receive a Forbidden status code. Privileges are assigned and managed at a user level. When a token is generated with client_credentials grant, then the default client user will be used to retrieve privileges. If access to a specific endpoint is required, please reach out to Onset.

# Get Data via File Endpoints

To request data for a particular logger, use the GET https://webservice.hobolink.com/ws/data/file/{format}/user/{userId} endpoint. This endpoint takes path and query parameters to indicate the user, logger, and range of data that is expected. When querying for data from this endpoint, data can be either returned for a specific time frame, or Onset can keep track of data previously returned and only new data will be returned on subsequent calls. Details on the different types of querying and what each parameter means can be found below:

| Parameter | Meaning |
|---|---|
| format | Path parameter. The format data should be returned in. Currently only JSON is supported. |
| userId | Path parameter. The numeric ID of the user account where the loggers have uploaded data. This can be pulled from the HOBOlink URL: www.hobolink.com/users/<userId> |
| loggers | Query parameter. A comma-delimited list of logger serial numbers that data should be grabbed for. The list is limited to 10 different loggers at a time. Note that the userId above must "own" the logger data you are requesting, meaning that stations were registered on that account at time of data upload or app was logged in under that user account when standalone loggers were read out and data pushed to cloud. |
| start_date_time | Query parameter. The date furthest in the past data should be grabbed for. Should be in the format yyyy-MM-dd HH:mm:ss in UTC time zone. Example, if the date 2020-01-01 00:00:00 is used, only data from New Year's day 2020 (GMT +0) and beyond will be returned. |
| end_date_time | Query parameter. This is only needed if time frame querying is desired (see later section). Should be in the format yyyy-MM-dd HH:mm:ss in UTC time zone. |
| only_new_data | Query parameter. This is only required if managed data tracking querying is desired (see later section). Simple string boolean (true/false). |
| last_successful_query_time | Query parameter. This is only required if using managed data tracking and it is desired to reset tracking to a previous call (see later section). Should be in the format yyyy-MM-dd HH:mm:ss in UTC time zone. |

The production API JSONDoc sandbox can be found at https://webservice.hobolink.com/ws/data/info/index.html.

- Navigate to Get Documentation → API → File → /file/{format}/user/{userId}.
- Fill Token field with valid access token obtained from OAuth Token endpoint (see previous section).
- Fill in only fields needed for your desired query test. The following data query sections detail what parameters need to be filled for each type of query.

Note that JSONDoc site can become unresponsive with very large result sets; please test with small time frames to start.

## Responses

### *Success*

On a successful query, the data will be returned as an observation_list in the specified format (only JSON currently supported). Each observation will indicate the logger, sensor, measurement type, data type, timestamp, value, and unit. Sample output can be seen below:


```
{
  "observation_list": [
    {
      "logger_sn": "99603325",
      "sensor_sn": "99508399-4",
      "timestamp": "2019-11-20 00:00:00Z",
      "data_type_id": "1",
      "si_value": 0.9995219339475939,
```

```
      "si_unit": "meters",
      "us_value": 3.2792714368359346,
      "us_unit": "feet",
      "scaled_value": 0,
      "scaled_unit": null,
      "sensor_key": 9658456,
      "sensor_measurement_type": "Water Level"
    },
    ...
    {
      "logger_sn": "99603325",
      "sensor_sn": "99508399-3",
      "timestamp": "2019-11-20 01:00:00Z",
      "data_type_id": "1",
      "si_value": 21.784423828125,
      "si_unit": "°C",
      "us_value": 71.211962890625,
      "us_unit": "°F",
      "scaled_value": 0,
      "scaled_unit": null,
      "sensor_key": 9658455,
      "sensor_measurement_type": "Water Temperature"
    }
  ],
  "message": "OK: Found: 169 results.",
  "max_results": false
}
```


Most fields above are self explanatory. You can view a full list in Data JSONDoc > Objects > observationlistjson (top level json structure) and observationjson (individual items in observation_list).

Field notes:

- "timestamp" is in format yyyy-mm-dd hh:mm:ssZ and always in UTC time zone (GMT+0 offset)
- "data_type_id" describes the type of data collected, as some logging configurations can take more than one reading type at each timestamp.
  - 1 - Normal, non-statistics data (most sensors)
  - 2 - Minimum statistic
  - 3 - Maximum statistic
  - 4 - Average statistic
  - 5 - Standard deviation statistic
- "scaled_value" and "scaled_unit" may be set or not, depending upon user-configured scaling. If sensor measurements have been scaled, "scaled_unit" will be a unit string for the scaled unit result and "scaled_value" will have the scaling output value. If sensor is not scaled by user, "scaled_unit" will be null and "scaled_value" is always zero.
- "sensor_key" is a tracking key internal to the Onset system. It can be useful for debugging data pointer issues in managed data tracking mode, but client applications should not rely on it being present or use it to do their own tracking.
- "max_results" indicates if request returned the maximum possible results (100,000). If "max_results" is false, all results could be returned in a single request, and there is no more data to request. If "max_results" is true, application will need to make more requests to obtain all data desired. When using the time frame querying (specified below), a smaller time frame will have to be specified manually until time frame safely returns all results and data can be requested in time segments. When using managed data tracking query (see section below), running the query again will retrieve the next batch, and application can continue requesting new data until max_results becomes false.

### *Failures*

In the case that data cannot be returned from the query, an error will be returned in JSON format. Below is an example error:


```
{
  "error": "VAL-034",
  "message": "No user ID provided.",
  "error_description": "Invalid request."
}
```

In the above example, the fields correspond to the following:

**error**: A simple error code identifying which type of error was seen.

**message**: A message providing some context around the error.

**error_description**: A basic description of what type of error was seen.

The following is a list of all the possible error codes that can be returned:

| Error | Description | Status Code |
|---|---|---|
| CFG-003 | Error retrieving property. | 400 |
| DBM-004 | Database error retrieving the user. | 500 |
| DBM-026 | Database error retrieving data. | 500 |
| DBM-027 | Database error saving data. | 500 |
| SYS-001 | System is busy. | 509 |
| SYS-002 | Too many requests. | 509 |
| VAL-001 | Time period start time is null. | 400 |
| VAL-002 | Time period start time is in the future. | 400 |
| VAL-004 | Device serial number or user name is required. | 400 |
| VAL-006 | Bad query date format. | 400 |
| VAL-033 | Invalid format. | 400 |
| VAL-034 | Invalid request. | 400 |
| WCE-001 | WebCache connection not available. | 500 |

## Time Frame Querying

Time frame querying allows the client to query for data within a specified time frame. Data that has occurred between the start and end times, inclusive, will be returned. Repeating the same request will return the same data each time (unless end time extends over end of current data and new points arrive between client requests).

Always fill Token field with a valid access token from OAuth token endpoint. Always fill path parameters (format JSON, userId number of HOBOlink account where loggers are registered).

Include the following query parameters:

- loggers - one or more logger serial numbers (comma separated)
- start_date_time - start of time frame in UTC, format yyyy-mm-dd hh:mm:ss
- end_date_time - end of time frame in UTC, format yyyy-mm-dd hh:mm:ss

Omit query parameters only_new_data and last_successful_query_time (do not fill in JSONDoc).

**Example Time Frame Query**

```
GET HTTP/1.1
Accept: application/json
Authorization: Bearer xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

https://webservice.hobolink.com/ws/data/file/JSON/user/99999?loggers=99999999&start_date_time=2019
-11-20+00%3A00%3A00&end_date_time=2019-11-20+01%3A00%3A00
```

Expected Return: Data points from logger 99999999 configured by user 99999 at or after 2019-11-20 00:00:00, and before or at 2019-11-20 01:00:00.

**Important:** Data returned may contain gaps depending upon system processing time or motes in recovery mode.

## Managed Data Tracking Querying

In cases where consistent querying is performed, clients may not want to keep track of what data they have received, or want to add logic to update the URL. To help alleviate this, Onset provides an option to use a managed data tracking query format. In order to use this type of querying, the only_new_data parameter must be present, and must be set to 'true'. The startTime has a similar function in this type of query, where no data prior to the startTime will be returned. However, unlike the Time Frame Querying, the data being returned doesn't start at the startTime, but rather at the last timestamp returned in a managed query.

Always fill Token field with a valid access token from OAuth token endpoint. Always fill path parameters (format JSON, userId number of HOBOlink account where loggers are registered).

Include the following query parameters:

- loggers - one or more logger serial numbers (comma separated)
- start_date_time - start of time frame in UTC, format yyyy-mm-dd hh:mm:ss
- only_new_data - true

Omit query parameters end_date_time and last_successful_query_time.

**Example Time Frame Query**

```
GET HTTP/1.1
Accept: application/json
Authorization: Bearer xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

https://webservice.hobolink.com/ws/data/file/JSON/user/99999?loggers=99999999&start_date_time=2019
-11-20+00%3A00%3A00&only_new_data=true
```

Expected Return: The first time the query is executed, then data starting at 2019-11-20 00:00:00 will be looked up. The number of results returned is limited to 100,000 per query, so the first 100,000 results at or after 2019-11-20 00:00:00 will be returned. The second time the *same* query is executed, the next batch of 100,000 data points will be returned, and so on. Assuming that there is only one sensor with data to be returned, and the sensor was logging at a 30 sec interval, the below table shows the time ranges that would be returned for each query. Assumption is current time is February 1st, 2020 at midnight. Note that all dates are in UTC.

| Query Number | Time Range Returned | Notes |
|---|---|---|
| 1 | 2019-11-20 00:00:00 - 2019-12-24 17:19:30 | First query, so begins at startTime and continues for 100,000 points |
| 2 | 2019-12-24 17:20:00 - 2020-01-28 10:40:00 | Next batch of 100,000 points |
| 3 | 2020-01-28 10:40:00 - 2020-01-31 23:59:30 | Last batch catches up to current time, returning the remaining 10,240 data points |

The same query can continue to be used as long as needed, grabbing only the data that has been generated since the last query. Each time the query returns a successful result code, the time of the latest data point is saved, keeping the time range moving forward. The downside to using this query format is that a query can't be retried if some data point are lost or corrupted. Time frame querying format can always be used to grab a specific selection of data, but is only really useful if the time frame can be specified. If data needs to be re-queried in a more programmatic way, the managed data query format can take an additional parameter to back track the data being returned; the last_successful_query_time parameter.

Data returned will not contain gaps and will only be delivered when full, contiguous data sets are available. This means that data processing delays or data recovery mode as exists on HOBOnet wireless sensors may delay data availability until all data has been obtained.

### *Data Replay*

The last_successful_query_time parameter can only be used to reset managed data pointers on the Onset side. This may be useful if you need to re-request past data or account for corruption on the client side. Data replay can only be used for replaying managed data tracking queries, so only_new_data must be set to true. The last_successful_query_time parameter takes a date in the same format as start_date_time (yyyy-MM-dd HH:mm:ss), again in UTC, and is used to back track the current state of the managed data queries to that point in time. Each time a managed data query is completed successfully, the time of the query and the current latest data point are saved. When data replay is used, the query that occurred before or at the specified last_successful_query_time will be identified, and the latest data point returned at that time will be used to determine the next batch.

Always fill Token field with a valid access token from OAuth token endpoint. Always fill path parameters (format JSON, userId number of HOBOlink account where loggers are registered).

Include following query parameters:

- loggers - one or more logger serial numbers (comma separated)
- start_date_time - start of time frame in UTC, format yyyy-mm-dd hh:mm:ss
- only_new_data - true
- last_succcessful_query_time - time of last successful request in UTC, format yyyy-mm-dd hh:mm:ss. Pointers are reset to end of results returned to client at that time (or before if no exact match). Setting this parameter to a timestamp before any client requests were executed essentially resets pointers to a clean slate.

Omit query parameter end_date_time.

**Example Managed Data Tracking Query**

```
GET HTTP/1.1
Accept: application/json
Authorization: Bearer xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

https://webservice.hobolink.com/ws/data/file/JSON/user/99999?loggers=99999999&start_date_time=2019
-11-20+00%3A00%3A00&only_new_data=true&last_successful_query_time=2019-12-30+00%3A00%3A00
```

Expected Return: Using the managed data tracking example above, let's say that the 3 queries were completed at the following times:

| Query Number | Time of Query |
|---|---|
| 1 | 2019-12-30 00:00:00 |
| 2 | 2020-01-29 00:00:00 |
| 3 | 2020-02-01 00:00:00 |

We then go back and realize the data from queries 2 and 3 got corrupted in transit back to the caller. In order to avoid replaying all the data or figuring out exactly which start_date_time we need to specify, we just use the last_successful_query_time 2019-12-30 00:00:00

This will reset pointers to results from the 2019-12-30 00:00:00 query number 1 and return results for query number 2. Then, we can continue on to execute the original query (*without* the last_successful_query_time), and managed data tracking will pick up with query number 3 and so forth.

If we wanted to replay all queries, we would simply set last_successful_query_time to a date far in the past, and it will act as if none of the queries have been run before.

Note that last_successful_query_time should only be used once, to reset the pointers. After that reset request, query format reverts to managed tracking mode without last_successful_query_time to continue moving forward.

## Get Data Endpoint Limitations

- Clients are limited to 30 requests per minute per URL. Exceeding will return HTTP status 429, error code SYS-002 for too many requests.
- Clients are limited to 75 concurrent data request threads. Exceeding will return HTTP status 509, error code SYS-001 for server busy, too many queries.