PHEMI RESTful API User Guide

Contents

REST API Overview

This section provides a brief overview of the PHEMI RESTful API.

REST API Requirements

This section describes the requirements for using the REST API.

Interface

As described above, the interface is built as a RESTful web service using HTTP as the underlying transport protocol.

User Session Authentication

An authenticated user session needs to be created before using the interface. During this process, a valid user credential is required from which a user session is created as the user logs in successfully. A cookie that holds the session information is required to perform further operations, such as ingesting data, querying aggregations and logging out, etc.

Response Sizes

All query responses should be small enough to be buffered in memory on the client side (i.e., a maximum of a few megabytes of decompressed data).

This requirement can only be met if the query for multiple identifiers encompasses a reasonably sized result set (e.g., identifiers returned across a reasonably short time range).

Response Time

Unless otherwise noted, aggregated queries using a single tag will return within a 250 ms response time, exclusive of network latency between Semios and PHEMI Central.

When multiple identifiers are queried (e.g., a set of sensor ids returned by a range query), the response time will be less than the total number of identifiers in the set multiplied times 250 ms (e.g., 10 sensors in the set would give a max response time of 2.5 seconds). For best performance a single identifier is preferred for the query.

Units of Measurement

Each data item should be ingested as the same base type (one of DOUBLE, LONG or STRING). The units of measurements must also be consistent (for example, a duration in milliseconds units should always be milliseconds).

Data Types

Ingested data and query parameter data types must be consistent. More specifically, the data types output by the DPF must be consistent with the data types used to query data (see Query Parameters below).

REST API Query Parameters

REST API queries are specified as URI parameters to an HTTP request. A query must provide parameters that specify the field to search, the value to search for, the data type of the field, and the comparison operator to use (=, >, <, etc.).

Query parameters or values must be URL-encoded when required (for example, a space is encoded as %20).

Query Operators

URI query operators follow the MongoDB convention of prefixing the operator with a double underscore "__"; for example, "__eq" is equals, "__ge" is greater than or equal to, etc. Supported query operators are shown below.

Table 1: REST Query Operators

Operator	Symbol	Description	Example
[None]		When no operator is specified, the equals operator is assumed.	`agg_id=id=sensor_2`
`__eq`	=	Performs an exact match	`agg_id__eq=id=sensor_2`
`__gt`	>	Queries for values greater than the specified value	`derived:attribute__gt=STRING:aaaa`
`__gte`	>=	Queries for values greater than or equal to the specified value	`derived:attribute__gte=DOUBLE:5.50`
`__lt`	<	Queries for values less than the specified value	`derived:attribute__lt=DATE:2014-10-15T00:54:32.750Z`
`__lte`	<=	Queries for values less than or equal to the specified value	`derived:attribute__lte=LONG:345543`
`__in`		Queries for values in a list	`derived:attribute__in=STRING:blue,red,green,purple`

Data Types in Queries

The data type must be specified as part of the query parameter and it must match the data type that is output by the data processing function that processed the ingested data.

Supported data types are shown below.

Table 2: Data Types Supported in Queries

Data Type	Description
`STRING`	String data type. String is the default data type.
`DOUBLE`	Double-precision floating point.
`LONG`	Long integer.
`DATE`	Date data type. Format is either the data in milliseconds since the start of the epoch or a date string formatted according to ISO 8601.

If no type is specified in the query parameter, the default type used is STRING. If the type specified in the query parameter does not match the type specified to the system when the document was ingested, the following will occur:

- If a type conversion cannot occur such as converting a non-numeric STRING to a LONG, then a **400** code is returned indicating a type conversion error occurred
- If a type conversion could occur, but is of the wrong type then no results will be returned

An example query parameter that matches (equals) a specific device ID with a LONG data type would be:

```
derived:model__eq=LONG:38
```

REST Request Format

All API requests start with the same URI prefix containing the data source identifier. The following is the API request that generates the URI prefix for a specific data source name:

```
GET /rest/data_sources?name=Semios
```

An HTTP response of 200 OK indicates success and contains the following JSON:

```
{
   "_links": {
      "curies": [
          {
             "href": "http://docs.phemi.com/relations/{rel}",
             "name": "ph",
             "templated": true
          }
       ],
       "ph:data_source": [
          {
             "href": "/rest/data_sources/53ea9cd8e779892f24ee8972",
             "title": "Semios"
          }
       ],
       "ph:find": {
          "href": "/rest/data_sources/{id}",
          "templated": true
       },
       "self": {
          "href": "/rest/data_sources?name=Semios"
       }
    },
    "count": 1
}
```

From the response, use the URI specified in *"[_link"s]"[ph:data_sourc"e]"[hre"f]* for all future requests (i.e., "/rest/data_sources/53ea9cd8e779892f24ee8972"). The URI prefix /rest/data_sources/<id> in the returned response is used throughout this document.

Note that the URI prefix cannot be taken literally from the above example but must be determined when the Semios Application starts for the first time. Once the URI is obtained, it does not have to be re-queried for in subsequent sessions as it is valid for the life of a specific datasource.

Queried Data

The Semios Web Application retrieves device log records from PHEMI Central using the Query Data APIs.

There are four kinds of data to retrieve:

- Metadata: Descriptive information generically describing the ingested data
- Derived data: individual pieces of data extracted and transformed from ingested data used for later analysis
- Aggregated data: calculated data based on the derived data
- Raw Data: The data that was ingested

To support each type of data, separate HTTP GET APIs are provided.

REST Request Reference

This section documents REST requests supported in the PHEMI RESTful API.

User Login

Log on to PHEMI Central and create an active session.

URI

```
POST /rest/user_sessions
```

Request Parameters

This request logs on to PHEMI Central by sending an HTTP POST request. The POST request expects valid user credentials in JSON format in the payload body, as in the following example.

```
{
   “username”: ”valid-username”,
   “password”: “valid-password”
}
```

Request Headers

None.

Response

The following are the HTTP status codes that can be expected:

201 Created

Successful login, the session has been created for the current user.

400 Bad Request

Failed login: the JSON body was not well-formed, or validation errors occurred.

401 Unauthorized

Failed login: the user credentials were not accepted.

409 Conflict

Failed login: A current session is already in progress for the specified user.

For a successful login (`201 Created` return code), the body of the response will contain status code, session ID, and URI for the session; additionally, a session cookie is returned. The following is an example.

```
{
   "code": 201,
   "id": "phemi-agile-session:b5a04a2353814a679a9df303a387aca2"
}
```

In this example:

- `code`: is the status code returned by the HTTP request.
- `id`: is the session ID

Usage Guidelines

This example shows Python code that logs a user in, saves the user session cookie, makes a request and logs out.

```
import urllib
import httplib2
import json

http = httplib2.Http()

# Log in
url = 'http//hostname:port/rest/user_sessions'
body = {'username': 'username', 'password': 'password'}
headers = {'Content-type': 'application/json'}
login_response, login_content = http.request (url,
                                               'POST',
                                               headers=headers,
                                               body=json.dumps(body))

# Save the login session cookie into the header for the next
    call
headers = {'Cookie': login_response['set-cookie']}

# Make a REST call and include the login session cookie in the
    header
url = 'http://hostname:port/rest/data_sources/datasource_id/raw_data'
query_response, query_content = http.request(url,
                                              'GET',
                                              headers=headers)

# Log out using the login session cookie in the header
url = 'http://hostname:port/rest/user_sessions'
headers = {'Cookie': login_response['set-cookie']}
logout_response, logout_content = http.request(url,
                                                'DELETE',
                                                headers=headers)
```

User Logout

Log off PHEMI Central and delete the session.

URI

```
DELETE /rest/user_sessions
```

Request Parameters

This request logs the user off PHEMI Central by sending a DELETE request containing the cookie for the user session. request. The DELETE request does not expect a JSON body or any request parameters. However, a cookie that contains the user session information is required.

Request Headers

None.

Response

The following are the HTTP status codes that can be expected:

204 No Content

The session for the current user has been deleted.

404 Not Found

The specified user session does not exist.

Usage Guidelines

This example shows Python code that logs a user in, then uses the login cookie to log the user off.

```
import urllib
import httplib2
import json

http = httplib2.Http()

# Log in
url = 'http//hostname:port/rest/user_sessions'
body = {'username': 'username', 'password': 'password'}
headers = {'Content-type': 'application/json'}
login_response, login_content = http.request (url,
                                               'POST',
                                               headers=headers,
                                               body=json.dumps(body))

# Include the login session in the header
url = 'http://hostname:port/rest/user_sessions'
headers = {'Cookie': login_response['set-cookie']}
logout_response, logout_content = http.request(url,
                                              'DELETE',
                                              headers=headers)
```

Data Ingest

Submit data to PHEMI Central.

URI

```
POST /rest/data_sources/datasource_id/ingestions
```

Request Parameters

This request submits data to PHEMI Central by issuing a POST request to the REST ingest URI. The POST request expects a JSON payload with the Semios device data:

- A version identifier
- A device identifier
- A set of non-hierarchical logs (data)
- A set of non-hierarchical tags

Request Headers

Prefer: respond-async

The presence of the `Prefer` header with the value `respond-async` will cause requests to be executed in the background. Control is returned to the REST client immediately. If this header is used, a return code of `202 Accepted` is returned.

Response

The following are the HTTP status codes that can be expected:

200 OK

The ingestion request completed successfully. The raw data has been persisted in PHEMI Central.

202 Accepted

The ingestion request was accepted, but did not complete.

400 Bad Request

The JSON body of the request is not well-formed.

401 Unauthorized

The user session is invalid.

403 Forbidden

The user session does not have permission to perform an ingestion.

404 Not Found

The data source does not exist.

500 Internal Server Error

An internal system error has occurred.

Usage Guidelines

In the event of a `202 Accepted` return code, the body of the response will contain status and meta information. The following is an example.

```
{
   "_links": {
      "curies": [
         {
            "href": "http://docs.phemi.com/rels/{rel}",
            "name": "ph",
            "templated": true
         }
       ],
       "ph:task_history": {
          "href": "/rest/tms/task_histories/54108814e779891b9b1a1eb4",
          "title": "Ingest and Derive for Data Source ECG"
       },
       "self": {
          "href": "/rest/tms/ingest_statuses/54108814e779891b9b1a1eb4"
       }
    },
    "error": null,
    "result": null,
    "status": "pending"
}
```

If the task is still running, the return status code will continue to be `202 Accepted`; otherwise, if the task finished and was successful, a `200 OK` response code is returned. If the task failed, a `500 Internal Server Error` is returned along with error details in the response body.

In the example below, the task is still running as the status shows on the last line. To determine the status of the ingestion at a later time, use the ***self href*** link specified in this response. In this example, the following is the link to use to determine the running status:

```
/rest/tms/ingest_statuses/54108814e779891b9b1a1eb4
```

The following is an example of a status while the ingest task is still running:

```
{
   "_id": {
      "$oid": "54108814e779891b9b1a1eb4
   },
   "_links": {
      "curies": [
         {
            "href": "http://docs.phemi.com/rels/{rel}",
            "name": "ph",
            "templated": true
         }
      ],
      "self": {
         "href": "/rest/tms/task_histories/54108814e779891b9b1a1eb4"
      }
   },
   "action": "chained",
   "error": "",
   "extra_props": {
      "tasks": [
         {
            "_id": {
               "$oid": "54468bb0e779891099693f44"
            },
            "action": "ingest",
            "error": "",
            "extra_props": {
               "data_source_id": "543d87d8e779897cfc5d3330",

               "data_source_name": "Semios Sensors",
               "total_metasize": 212,
               "total_objects": 1,
               "total_rawsize": 780
            },
            "finished": "2014-10-21T16:37:04.576Z",
            "name": "Semios Retention",
            "queued": "2014-10-21T16:37:04.464Z",
            "started": 2014-10-21T16:37:04.498Z",
            "status": "finished",
            "uniquename": "datasource-Semios Sensors",
            "username": "admin"
         },
         {
            "_id": {
                "$oid": "54468bb0e779891099693f45"
            },
            "action": "derive",
            "error": "",
            "extra_props": {},
            "finished": "2014-10-21T16:37:19.683Z",
            "name": "Semios Retention",
            "queued": "2014-10-21T16:37:04.488Z",
            "started": "2014-10-21T16:37:04.583Z",
            "status": "finished",
            "uniquename": "datasource-Semios Retention",
             "username": "admin"
          }
       ]
    },
    "finished": "2014-10-21T16:37:19.686Z",
    "name": "Ingest and Derive for Data Source Semios Sensors",
    "queued": "2014-10-21T16:37:04.491Z",
    "started": "2014-10-21T16:37:04.494Z",
    "status": "finished",
    "uniquename": "Ingest and Derive for Data Source Semios Sensors",
    "username": "admin"
}
```

Notice that the task status is set to “finished” and the time the task finished is also given. If the DPF has not been configured to process the derived data immediately, a finished task only indicates that the ingestion of raw data is complete (the raw data has been persisted in PHEMI Central). To determine when generation of derived data has been completed, follow the instructions in the Derivation Completion section.

Query Sensor Data

Query data from a data source.

URI

```
GET /rest/data_sources/datasource_id/raw_data?attribute-
name__operator=lexicoder:filter-value
```

Request Parameters

This request allows you to query data from a given data source based on various criteria such as attribute values and ranges. Query parameters or values must be correctly escaped for URLs as necessary.

This request supports the following parameters.

attribute-name__operator=lexicoder*:*filter-value

Queries the specified data source for the specified data. In this parameter:

- *attribute-name*: The raw-data attribute being queried. The attribute name is defined in the DPF for the data source.
- *__operator*: The comparison operator specifying how the query should be performed. Supported operators are listed in *Query Operators* on page 4.
- *lexicoder*: The data type of the queried value. The data type is defined in the DPF that creates the derived data during data ingestion. Supported data types are described in *Data Types in Queries* on page 4.
- *filter-value*: The value to query on. For example, in `/rest/datasources/`*`datasource-id`*`/raw_data?derived:id__eq=STRING:sensor_1138`, the *filter-value* is `sensor_1138`.

skip=*num-rows-to-skip*

Skips the specified number of rows in the query result and returns subsequent rows in the resultset. Rows are considered to start numbering at 1.

By default, `skip` is 0, which means no results are skipped. Specifying a value for `skip` that is larger than the number of rows in the resultset is not an error, but no results are returned. Specifying a negative value for `skip` is an error, and causes an exception.

As an example, assume a resultset of 50 rows, numbered row 1 to row 50. The results returned for various values of `skip` are shown below.

Table 3: "skip" Parameter Behavior

Value for "skip"	Rows Returned
Undefined	1 to 50
`skip=0`	1 to 50
`skip=1`	2 to 50
`skip=10`	11 to 50
`skip=50`	None
`skip=100`	None
`skip=-1`	Error

The following example skips the first 50 rows of the query results and returns from row 51 onwards.

```
/rest/datasources/datasource-id/raw_data?
derived:id__eq=STRING:sensor_1138&skip=50
```

Note that you can use `skip` parameter in combination with the `limit` parameter to achieve pagination-type functionality; see the `limit` parameter for an example.

`limit=`*`max-rows-to-return`*

Specifies the maximum number of rows to return from the resultset. By default, `limit` is set to 50. The largest value permitted for `limit` is 2500; specifying a larger value is an error.

As an example, assume a resultset of 100 rows, numbered row 1 to row 100. The results returned for various values of `limit` are shown below.

Table 4: "limit" Parameter Behavior

Value for "limit"	Rows Returned
Undefined	100
`limit=1`	1
`limit=2`	1 and 2
`limit=10`	1 to 10
`limit=100`	1 to 100
`limit=200`	1 to 100

The following example limits returned results to 50 rows.

```
/rest/datasources/datasource-id/raw_data?
derived:id__eq=STRING:sensor_1138&limit=50
```

The `limit` parameter can be used with the `skip` parameter to select from the resultset in various ways. For example, specifying `skip=1, limit=50` returns rows 2 through 51 of the resultset. Incrementing `skip` by the value of `limit` results in pagination functionality, as in the first two lines of the following example. In this example, assume that the resultset is 100 rows.

```
skip=0,limit=50     # Returns rows 1 to 50
skip=50,limit=50    # Returns rows 51 to 100
skip=75, limit=50   # Returns rows 76 to 100
skip=99, limit=50   # Returns row 100
skip=100, limit=50  # Returns nothing
```

`order_by=`*`comma-separated-list`*

Specifies how the returned resultset should be ordered. Order is specified as a comma-separated list of attributes. The hyphen is a reverse-ordering operator: prepending a hyphen ("-") in front of a value reverses the normal ordering or data. For example, if `derived:data|wind_speed` is normally returned in ascending order, `-derived:data|wind_speed`is returned in descending order.

Consider the following request:

```
/rest/datasources/datasource-id/
raw_data?derived:id__eq=STRING:sensor_113&
order_by=derived:data|temperature,-derived:data|wind_speed
```

In this example, the resultset returned by the query would be ordered first by `derived:data|temperature`and then by `derived:data|wind_speed` in reverse order.

The attributes listed in the `order_by` attribute apply only to the rows actually returned by the query. The `order_by` specification has no effect on what rows the query returns, only on the order of the rows returned.

An exception to this behavior occurs when one of the attributes specified for the `order_by` parameter matches an attribute in the query itself, as in the following example:

```
/rest/datasources/datasource-id/
```

```
raw_data?derived:id__eq=STRING:sensor_113&
order_by=-derived:id
```

If a reverse ordering operator is present, as it is in the example, it is the operator is applied first, to make a more efficient query. In this case, the query could potentially return rows from the database in a different order than a query not using the operator.

Request Headers

None.

Response

When an HTTP request is made against the raw data REST interface, the following HTTP status codes can be expected depending on the query result:

200 OK

The query was successful.

400 Bad Request

Invalid criteria parameter values were provided.

401 Unauthorized

The user is not authenticated.

403 Forbidden

The authenticated user is not authorized to perform the query.

404 Not Found

A resource associated with the query (for example, a data source) does not exist.

500 Internal Server Error

An internal system error has occurred.

Usage Guidelines

In the following example:

- The data source ID is `540e5056e77989182bd5aad0`.
- The ID of each derived data item takes the form `sensor_`*`n`*
- Data is to be returned for `sensor_1130` up to but not including `sensor_1140`.

```
GET /rest/data_sources/540e5056e77989182bd5aad0/raw_data?
 derived:id__gte=STRING:sensor_1130& derived:id__lt=STRING:sensor_1140
```