runeq

Rune Labs

Jan 23, 2024

CONTENTS:

Python SDK to query data from the Rune Labs API

CHAPTER

ONE

INSTALLATION

Install using pip:

```
>>> pip install runeq
```

Or directly from source:

```
>>> python3 setup.py install
```

CHAPTER

TWO

DOCUMENTATION

2.1 Quickstart

2.1.1 Prerequisites

API Credentials

To access Rune's APIs, you will need to obtain API credentials. For multi-patient analyses, we recommended using **user access tokens**.

Note: If you belong to multiple organizations, note that only one organization is "active" at a time. You can only access resources that belong to your active organization. This impacts both what is returned by the SDK *and* what you see in the Rune web portal.

You can change your active organization through code (see *Set Active Org*) or in the Rune web portal (click on the profile icon, in the top right corner).

To create a new access token:

1. Log in to the Rune web portal
2. Click on the profile icon, in the top right corner.
3. Click on **User Settings**.
4. On the left sidebar, click on **Access Tokens**.
5. Click **CREATE ACCESS TOKEN**.
6. Copy the token ID and secret *before closing the page*. The secret will never be shown again.

See *Configuration Setup* for details about how to use these credentials with this library.

It is highly recommended that you rotate your access tokens every 3-6 months, by creating a new token and deactivating the old one. Store your access tokens securely, and do not share them.

Configuration Setup

`runeq` uses a YAML-formatted file to manage configuration settings (e.g. API credentials). The easiest way to set up this configuration is via the `runeq` command line tool, which is installed along with the Python library.

To get started, open a terminal and run the following command in a Python environment where `runeq` is installed. This command will prompt you to enter an access token ID and secret, and it will create a configuration file in the default location.

```
runeq configure setup
```

This command also provides options to get and set specific values in your config file. To see help documentation:

```
runeq configure --help
```

If you want to create or manage a configuration file manually, refer to the example config for the expected contents.

Once a configuration file exists, you won't need to repeat this step (unless you're rotating your access token, getting set up on a different computer, etc).

2.1.2 Initialization

To get started with the library, use `initialize`. This loads credentials from your configuration file (see *Configuration Setup*).

```
from runeq import initialize

initialize()
```

To see information about your authenticated user:

```
from runeq.resources.user import get_current_user

my_user = get_current_user()
print(my_user)
print('Active Org:', my_user.active_org_name)
```

2.1.3 Usage

Set Active Org

To get metadata about all the organizations that you belong to:

```
from runeq.resources.org import get_orgs

all_orgs = get_orgs()
for org in all_orgs:
    print(org)
```

You can set your active organization using an org ID:

```
from runeq.resources.org import set_active_org

org_id = "aa0c21f97d6a0593b0a247c68f015d68b787655e"
active_org = set_active_org(org_id)
print('Active Org:', active_org.name)
```

Explore Metadata

After initializing the library, you can fetch metadata about various resources.

For example, you can fetch metadata about all the patients in your active org:

```
from runeq.resources.patient import get_all_patients

patients = get_all_patients()

for patient in patients:
    print(patient)
    for device in patient.devices:
        print(' ', device)

    print('')
```

get_all_patients returns a *PatientSet*. This object can be serialized as a list of dictionaries, e.g. to save the metadata to a file:

```
import json

with open('patients.json', 'w') as f:
    json.dump(patients.to_list(), f, indent=4)
```

You can also convert a *PatientSet* to a collection of devices (a *DeviceSet*). This may be more convenient for a columnar data format, like a pandas DataFrame.

```
import pandas as pd

devices = patients.devices
devices_df = pd.DataFrame(devices.to_list())
```

Similarly to fetching information about patients, you can fetch information about projects, and metadata related to the patients within projects (and cohorts).

You can find information about a single project:

```
from runeq.resources.project import get_project

project = get_project(project_id="example_id")
print(project.to_dict())
```

To view all the patients in a project, and their related project metrics you can use the following example:

```
from runeq.resources.project import get_project_patients
```

(continues on next page)

(continued from previous page)

```
project_patients = get_project_patients(project_id="example_id")

for project_patient in project_patients:
    print(project_patient)
    for metric in project_patient.metrics:
        print(' ', metric)

    print('')
```

It may be easier to view a single project patient in a dataframe which you can do by:

```
from runeq.resources.project import get_project_patients

project_patients = get_project_patients(project_id="example_id")
target_patient_id = "patient_id_example"

df = project_patients[target_patient_id].get_patient_metadata_dataframe()

df
```

Fetch Timeseries Data

Use *get_patient_stream_metadata* to get a *StreamMetadataSet* with details about a particular patient's data. If you're interested in a more specific set of streams, the function accepts additional filters.

```
from runeq.resources.stream_metadata import get_patient_stream_metadata

patient_id = "c4bd060df1454aa0adc978985512c6e9"
patient_streams = get_patient_stream_metadata(patient_id)
print(f'Found {len(patient_streams)} streams')
```

Once you have a *StreamMetadataSet*, you can use the **filter** operation to get a more specific subset of streams:

```
# Filter for data collected from a particular device
device_id = "eb#8c31"
device_streams = patient_streams.filter(device_id=device_id)

# Filter by broad category
neural_streams = patient_streams.filter(category="neural")

# Specify multiple arguments to find streams that match
# all criteria
neural_device_streams = patient_streams.filter(
    category="neural",
    device_id=device_id,
)

# Use a custom filter function
import time

def in_last_two_weeks(stream) -> bool:
```

(continues on next page)

(continued from previous page)

```
    """Return True if stream has data in the last two weeks"""
    two_weeks_ago = time.time() - 14*24*60*60
    return stream.max_time > two_weeks_ago

recent_vitals_streams = patient_streams.filter(
    category="vitals",
    filter_function=in_last_two_weeks
)
```

You can also combine multiple *StreamMetadataSet* s, using **update**:

```
from runeq.resources.stream_metadata import StreamMetadataSet

lfp_power_streams = patient_streams.filter(
    category="neural",
    measurement="lfp_trend_log_power",
)
tremor_streams = patient_streams.filter(
    category="symptom",
    measurement="tremor",
    stream_type_id="duration"
)

lfp_and_tremor_streams = StreamMetadataSet()
lfp_and_tremor_streams.update(lfp_power_streams)
lfp_and_tremor_streams.update(tremor_streams)
```

Using a *StreamMetadataSet*, you can fetch the **availability** of all or any of the streams:

```
availability_df = lfp_and_tremor_streams.get_batch_availability_dataframe(
    start_time=1662000000,
    end_time=1663123000,
    resolution=3600,
    batch_operation="any",
)
```

Note: The API for "batch availability" has a limit on the number of streams that it can process at a time. If you're running the example code with a patient who has multiple devices, the snippet above may exceed the API limit. Try limiting the number of streams in the set using a custom filter function, to select for a few of those device IDs.

When you're ready to fetch data, you can gather all the raw stream data into a pandas dataframe:

```
stream_df = lfp_and_tremor_streams.get_stream_dataframe(
    start_time=1662499000,
    end_time=1663123000,
)
```

You can also work directly with responses from the V2 Stream API. See *stream* and *StreamMetadata* for details.

2.2 Configuration

Configuration for accessing Rune APIs.

class runeq.config.**Config**(**args*, ***kwargs*)

Holds configuration (e.g. auth credentials, URLs, etc)

__init__(**args*, ***kwargs*)

Initialize configuration options.

Parameters

- ***args** – Accepts at most 1; a filename. If provided, values will be loaded from the file, using *load_yaml()*. It is invalid to provide both a filename **and** keyword arguments.
- ****kwargs** – Passed to set_values().

Examples

There are three valid ways to create a config:

```
>>> cfg = Config()
# Load from default file location (~/.rune/config)
```

```
>>> cfg = Config('./example_config.yaml')
# Load from a specified YAML file
```

```
>>> cfg = Config(access_token_id='foo', access_token_secret='bar')
# Set values using keyword arguments. This can be used with any
# valid combination of config options; the example above sets a
# user access token.
```

property auth_headers

Authentication headers for HTTP requests to Rune APIs.

load_yaml(*filename='~/.rune/config'*)

Set values from a YAML file. Keys from the file are passed directly to set_values(), as kwargs.

Parameters

filename – File path for a YAML-formatted config file

set_values(*auth_method=None*, *access_token_id=None*, *access_token_secret=None*, *client_key_id=None*, *client_access_key=None*, *jwt=None*, *cognito_client_id=None*, *cognito_refresh_token=None*, *cognito_region_name='us-west-2'*, *stream_url=None*, *graph_url=None*, ***kwargs*)

Set configuration values.

Parameters

- **auth_method** – What type of authentication credentials to use. Must be one of 'access_token', 'client_keys', or 'jwt'. If not set, the auth method is inferred based on which credentials are specified (as long as it's unambiguous).
- **access_token_id** – User access token ID
- **access_token_secret** – User access token secret
- **client_key_id** – Client key ID

- **client_access_key** – Client access key
- **jwt** – JWT
- **stream_url** – base URL to use for the stream API
- **graph_url** – base URL to use for the graph API
- ****kwargs** – Arbitrary values may be provided, but they are ignored.

2.3 Resources

Fetch data from Rune APIs.

By default, globally-initialized clients are used for all API requests (see *initialize*). Functions that make API requests also accept optional client(s), which can be used in lieu of the global initialization.

Metadata is fetched from Rune's GraphQL API (https://graph.runelabs.io/graphql), using a *GraphClient*. Timeseries data is fetched from the V2 Stream API, using a *StreamClient*.

For example usage patterns, see *Quickstart*.

2.3.1 API Clients

Clients for Rune's GraphQL API and V2 Stream API.

runeq.resources.client.**initialize**(**args*, ***kwargs*)

Initializes the library with specified configuration options. Sets global clients for requests to the GraphQL API and the V2 Stream API.

Parameters

- ***args** – Accepts at most 1; a filename. If provided, values will be loaded from the file. It is invalid to provide both a filename **and** keyword arguments.
- ****kwargs** – Initialize client with keyword arguments. If using client keys, specify the client_key_id & client_access_key. If using access tokens, specify access_token_id & access_token_secret.

Examples

There are several valid ways to use this function:

```
>>> initialize()
# Load the default config file (~/.rune/config)
```

```
>>> initialize('./example_config.yaml')
# Load values from a YAML file at a specified path
```

```
>>> initialize(access_token_id='foo', access_token_secret='bar')
>>> initialize(client_key_id='foo', client_access_key='bar')
# Set configuration values using keyword arguments (instead
# of a file). This can be used with any valid combination of
# config options (e.g. with an access token OR a client key).
```

runeq.resources.client.**global_graph_client**() → *GraphClient*

Returns the globally configured GraphQL client. Use *initialize* to configure the client.

Raises
errors.InitializationError – if the library was not initialized.

runeq.resources.client.**global_stream_client**() → *StreamClient*

Returns the globally configured Stream API client. Use *initialize* to configure the client.

Raises
errors.InitializationError – if the library was not initialized.

class runeq.resources.client.**GraphClient**(*config:* Config)

Rune GraphQL Client to query stream metadata.

__init__(*config:* Config)

Initialize the Graph API Client.

execute(*statement: str*, ***variables*) → Dict

Execute a GraphQL query against the API.

class runeq.resources.client.**StreamClient**(*config:* Config)

Client to query the V2 Stream API.

__init__(*config:* Config)

Initialize the Stream API Client.

get_data(*path: str*, ***params*) → Iterator[Union[str, dict]]

Makes request(s) to an endpoint of the V2 Stream API. Iterates over responses, following pagination headers until all data has been fetched.

Parameters

- **path** – Path for an endpoint of the V2 Stream API.
- ****params** – Query parameters. If the format parameter is "json", responses are parsed as JSON. Otherwise, the response is returned as text.

Returns
Iterator over the API responses. If the format parameter is "json", each value is a dictionary. Otherwise, each value is a CSV-formatted string (the default for the V2 API).

Raises
errors.APIError –

2.3.2 Patient Metadata

Fetch metadata about patients, including their devices.

Patients

runeq.resources.patient.**get_patient**(*patient_id: str*, *client: Optional[GraphClient] = None*) → *Patient*

Get the patient with the specified patient ID.

Parameters

- **patient_id** – Patient ID
- **client** – If specified, this client is used to fetch metadata from the API. Otherwise, the global GraphClient is used.

runeq.resources.patient.**get_all_patients**(*client: Optional[GraphClient] = None*) → *PatientSet*

Get a set of all patients the user has access to.

Parameters

client – If specified, this client is used to fetch metadata from the API. Otherwise, the global GraphClient is used.

class runeq.resources.patient.**Patient**(*id: str*, *name: str*, *created_at: float*, *devices: DeviceSet*, ***attributes*)

A patient.

A patient represents a person about whom data was measured. This may include a StrivePD user, a person with a neural implant, etc. Each patient has a unique identifier ("patient ID").

__init__(*id: str*, *name: str*, *created_at: float*, *devices: DeviceSet*, ***attributes*)

Initialize with metadata.

Parameters

- **id** – ID of the patient
- **name** – Human-readable display name for the patient
- **created_at** – When the patient's record was created (unix timestamp)
- **devices** – Devices that belong to the patient
- ****attributes** – Other attributes associated with the patient

static denormalize_id(*patient_id: str*) → str

Add resource prefix to a patient ID (if it doesn't exist).

This constructs the form of the ID that is used for requests to the GraphQL API.

Parameters

patient_id – Patient ID

device(*device_id: str*) → *Device*

Return the patient's device with the specified device ID.

Parameters

device_id – Device ID

Raises

ValueError – if the patient does not have a device with the ID

get(*attr: str*, *default: Any = None*)

Get the value of any attribute in self.attributes.

property id: str

ID of the item.

static normalize_id(*patient_id: str*) → str

Strip resource prefix from a patient ID (if it exists).

Parameters
patient_id – Patient ID

to_dict() → dict

Dictionary representation of the Patient attributes.

class runeq.resources.patient.**PatientSet**(*items: Iterable[Patient] = ()*)

A collection of Patients.

__init__(*items: Iterable[Patient] = ()*)

Initialize with Patients.

add(*item: ItemBase*)

Add a single item to the set. Must be the same type as other members of the collection

property devices: *DeviceSet*

Set of all devices that belong to the patients in this collection.

get(*id: str*) → ItemBase

Get an item by id.

Raises
ValueError – if the set does not contain an item with the ID.

ids() → Iterator[str]

Iterator over the IDs of the items in this set.

remove(**items: Union[str, ItemBase]*)

Remove item(s) from this set.

to_dataframe() → DataFrame

Convert items to a dataframe (wraps *to_list()*)

to_list() → List[dict]

List of all the items in the set. Each item is formatted as a dictionary, using the item's *to_dict()* method.

update(*items: Iterable[ItemBase]*)

Add an iterable of item(s) to this set. All items must be the same class as members of this collection.

Devices

runeq.resources.patient.**get_device**(*patient: Union[Patient, str]*, *device_id: str*, *client: Optional[GraphClient] = None*) → *Device*

Get a patient's device, by the device ID.

Note that if a Patient object is provided, this function serves as a wrapper around Patient.device(). If a patient ID is provided, metadata is fetched from the API.

Parameters

- **patient** – a patient ID or Patient object
- **device_id** – Device ID
- **client** – If specified, this client is used to fetch metadata from the API. Otherwise, the global GraphClient is used.

runeq.resources.patient.**get_all_devices**(*patients: Union[PatientSet, List[str]] = None*, *client: Optional[GraphClient] = None*) → *DeviceSet*

Get a set of all devices belonging to a set of patients. If a specific set is not specified, returns all devices belonging to all patients in the user's active organization.

Parameters

- **patients** – a list of patient IDs or a PatientSet
- **client** – If specified, this client is used to fetch metadata from the API. Otherwise, the global GraphClient is used.

class runeq.resources.patient.**Device**(*id: str*, *patient_id: str*, *name: str*, *created_at: float*, *device_type_id: str*, ***attributes*)

A patient device.

A device is a sensor serving as the datasource, such as a neural implant, phone, or wearable. Each device belongs to a patient. A patient may have multiple devices: each one has a unique identifier ("device ID").

__init__(*id: str*, *patient_id: str*, *name: str*, *created_at: float*, *device_type_id: str*, ***attributes*)

Initialize with metadata.

Parameters

- **id** – ID of the device
- **patient_id** – ID of the patient
- **name** – Human-readable name for the device
- **created_at** – When the device was created (unix timestamp)
- ****attributes** – Other attributes associated with the device

static denormalize_id(*patient_id: str*, *device_id: str*) → str

Add resource prefix and suffix to a patient/device ID.

This constructs the form of the ID that is used for requests to the GraphQL API.

Parameters

- **patient_id** – ID of the patient who owns the device
- **device_id** – Device ID

get(*attr: str*, *default: Any = None*)

Get the value of any attribute in self.attributes.

property id: str

ID of the item.

static normalize_id(*device_id: str*) → str

Strip resource prefix and suffix from a device ID (if they exist).

Parameters
device_id – Device ID

to_dict() → dict

Dictionary representation of the item's attributes.

class runeq.resources.patient.**DeviceSet**(*items: Iterable[Device] = ()*)

A collection of Devices.

__init__(*items: Iterable[Device] = ()*)

Initialize with Devices.

add(*item: ItemBase*)

Add a single item to the set. Must be the same type as other members of the collection

get(*id: str*) → ItemBase

Get an item by id.

Raises

ValueError – if the set does not contain an item with the ID.

ids() → Iterator[str]

Iterator over the IDs of the items in this set.

remove(**items: Union[str, ItemBase]*)

Remove item(s) from this set.

to_dataframe() → DataFrame

Convert items to a dataframe (wraps *to_list()*)

to_list() → List[dict]

List of all the items in the set. Each item is formatted as a dictionary, using the item's *to_dict()* method.

update(*items: Iterable[ItemBase]*)

Add an iterable of item(s) to this set. All items must be the same class as members of this collection.

2.3.3 Organization Metadata

Fetch metadata about organizations. A research lab or clinical site is typically represented as an organization.

runeq.resources.org.**get_org**(*org_id: str*, *client: Optional[GraphClient] = None*) → *Org*

Get the org with the specified ID.

Parameters

- **org_id** – Organization ID
- **client** – If specified, this client is used to fetch metadata from the API. Otherwise, the global GraphClient is used.

runeq.resources.org.**get_orgs**(*client: Optional[GraphClient] = None*) → *OrgSet*

Get all the organizations that the current user is a member of.

Parameters

client – If specified, this client is used to fetch metadata from the API. Otherwise, the global GraphClient is used.

runeq.resources.org.**set_active_org**(*org: Union[str, Org]*, *client: Optional[GraphClient] = None*) → *Org*

Set the active organization for the current user.

Parameters

- **org** – an org ID or Org
- **client** – If specified, this client is used to fetch metadata from the API. Otherwise, the global GraphClient is used.

Returns

The active organization

class runeq.resources.org.**Org**(*id: str*, *name: str*, *created_at: float*, *tags: Iterable = ()*, ***attributes*)

Metadata for an organization.

__init__(*id: str*, *name: str*, *created_at: float*, *tags: Iterable = ()*, ***attributes*)

Initialize with metadata.

Parameters

- **id** – ID of the organization
- **name** – Human-readable name
- **created_at** – When the organization was created (unix timestamp)
- **tags** – Organization tags
- ****attributes** – Other attributes associated with the organization

static denormalize_id(*org_id: str*) → str

Add resource prefix and suffix to an org ID (if they don't exist).

This constructs the form of the ID that is used for requests to the GraphQL API.

Parameters
org_id – Organization ID

get(*attr: str*, *default: Any = None*)

Get the value of any attribute in self.attributes.

property id: str

ID of the item.

static normalize_id(*org_id: str*) → str

Strip resource prefix and suffix from an org ID (if they exist).

Parameters
org_id – Organization ID

to_dict() → dict

Dictionary representation of the item's attributes.

class runeq.resources.org.**OrgSet**(*items: Iterable[Org] = ()*)

A collection of Organizations.

__init__(*items: Iterable[Org] = ()*)

Initialize with Orgs.

add(*item: ItemBase*)

Add a single item to the set. Must be the same type as other members of the collection

get(*id: str*) → ItemBase

Get an item by id.

Raises
ValueError – if the set does not contain an item with the ID.

ids() → Iterator[str]

Iterator over the IDs of the items in this set.

remove(**items: Union[str, ItemBase]*)

Remove item(s) from this set.

to_dataframe() → DataFrame

Convert items to a dataframe (wraps *to_list()*)

to_list() → List[dict]

List of all the items in the set. Each item is formatted as a dictionary, using the item's *to_dict()* method.

update(*items: Iterable[ItemBase]*)

Add an iterable of item(s) to this set. All items must be the same class as members of this collection.

2.3.4 Stream Metadata

Fetch metadata about streams, including stream types.

runeq.resources.stream_metadata.**get_all_stream_types**(*client: Optional[GraphClient] = None*) → *StreamTypeSet*

Get all stream types.

Parameters

client – If specified, this client is used to fetch metadata from the API. Otherwise, the global GraphClient is used.

runeq.resources.stream_metadata.**get_patient_stream_metadata**(*patient_id: str*, *device_id: Optional[str] = None*, *stream_type_id: Optional[str] = None*, *algorithm: Optional[str] = None*, *category: Optional[str] = None*, *measurement: Optional[str] = None*, *client: Optional[GraphClient] = None*, ***parameters*) → *StreamMetadataSet*

Get stream metadata for a patient's streams, matching ALL filter parameters. Only the patient ID is required.

Parameters

- **patient_id** – Patient ID
- **device_id** – Device ID
- **stream_type_id** – Stream type ID
- **algorithm** – A versioned label that describes the process that was used to derive this time-series.
- **category** – A broad categorization of the data type (e.g. neural, vitals, etc)
- **measurement** – A specific label for what is being measured (e.g. heart_rate, step_count, etc).
- **client** – If specified, this client is used to fetch metadata from the API. Otherwise, the global GraphClient is used.
- ****parameters** – Key/value pairs that label the stream.

runeq.resources.stream_metadata.**get_stream_availability_dataframe**(*stream_ids: Union[str, Iterable[str]]*, *start_time: Union[float, date]*, *end_time: Union[float, date]*, *resolution: int*, *batch_operation: Optional[str] = None*, *limit: Optional[int] = None*, *page_token: Optional[str] = None*, *timestamp: Optional[str] = 'iso'*, *timezone: Optional[int] = None*, *stream_client: Optional[StreamClient] = None*, *graph_client: Optional[GraphClient] = None*) → DataFrame

Get stream availability data as a dataframe. If a single stream_id is passed in, the dataframe will be enriched with the stream's metadata. Otherwise it will contain just the availability data.

Note that this is different from the get_stream_availability_dataframe method on the `StreamMetadataSet`. This dataframe contains the availability of each *individual* stream.

Parameters

- **stream_ids** – 1 or multiple stream IDs
- **start_time** – Start time for the query, provided as a unix timestamp (in seconds) or a datetime.date.
- **end_time** – End time for the query, provided as a unix timestamp (in seconds) or a datetime.date.
- **resolution** – Interval between returned timestamps, in seconds.
- **batch_operation** – Either "any" or "all", which determines what type of batch calculation will determine availability for the batch of streams. Availability values will equal 1 when data is available for "all" or "any" of the requested streams in the given interval.
- **limit** – Maximum number of timestamps to return, across *all pages* of the response. A limit of 0 (default) will fetch all available data.
- **page_token** – Token to fetch the subsequent page of results. The value is obtained from the 'X-Rune-Next-Page-Token' response header field.
- **timestamp** – Optional enum "unix", "unixns", or "iso", which determines how timestamps are formatted in the response
- **timezone** – Optional timezone offset, in seconds, used to calculate string-based timestamp formats such as datetime and iso. For example, PST (UTC-0800) is represented as -28800. If omitted, the timezone is UTC.
- **stream_client** – If specified, this client is used to fetch data from the API. Otherwise, the global StreamClient is used.
- **graph_client** – If specified, this client is used to fetch metadata from the API. Otherwise, the global GraphClient is used.

runeq.resources.stream_metadata.**get_stream_dataframe**(*stream_ids: Union[str, Iterable[str]]*, *start_time: Optional[Union[float, date]] = None*, *start_time_ns: Optional[int] = None*, *end_time: Optional[Union[float, date]] = None*, *end_time_ns: Optional[int] = None*, *limit: Optional[int] = None*, *page_token: Optional[str] = None*, *timestamp: Optional[str] = 'iso'*, *timezone: Optional[int] = None*, *translate_enums: Optional[bool] = True*, *stream_client: Optional[StreamClient] = None*, *graph_client: Optional[GraphClient] = None*) → DataFrame

Get stream(s) as enriched dataframe with stream data and metadata.

Parameters

- **stream_ids** – 1 or multiple stream IDs
- **start_time** – Start time for the query, provided as a unix timestamp (in seconds) or a datetime.date.
- **start_time_ns** – Start time for the query, provided as a unix timestamp (in nanoseconds).
- **end_time** – End time for the query, provided as a unix timestamp (in seconds) or a datetime.date.
- **end_time_ns** – End time for the query, provided as a unix timestamp (in nanoseconds).format: Optional enum "json" or "csv", which determines the content type of the response
- **limit** – Maximum number of timestamps to return, across *all pages* of the response. A limit of 0 (default) will fetch all available data.
- **page_token** – Token to fetch the subsequent page of results. The value is obtained from the 'X-Rune-Next-Page-Token' response header field.
- **timestamp** – Optional enum "unix", "unixns", or "iso", which determines how timestamps are formatted in the response
- **timezone** – Optional timezone offset, in seconds, used to calculate string-based timestamp formats such as datetime and iso. For example, PST (UTC-0800) is represented as -28800. If omitted, the timezone is UTC.
- **translate_enums** – If True, enum values are returned as their string representation. Otherwise, enums are returned as integer values.
- **stream_client** – If specified, this client is used to fetch data from the API. Otherwise, the global StreamClient is used.
- **graph_client** – If specified, this client is used to fetch metadata from the API. Otherwise, the global GraphClient is used.

Raises

RuneError – if any of the stream IDs is not found.

runeq.resources.stream_metadata.**get_stream_metadata**(*stream_ids: Union[str, Iterable[str]]*, *client: Optional[GraphClient] = None*) → Union[*StreamMetadata*, *StreamMetadataSet*]

Get stream metadata for the specified stream_id(s).

Parameters

- **stream_ids** – ID of the stream or list of IDs
- **client** – If specified, this client is used to fetch metadata from the API. Otherwise, the global GraphClient is used.

Returns
if a single stream ID is specified StreamMetadataSet: if multiple stream IDs are specified

Return type
StreamMetadata

Raises
RuneError – if any of the stream IDs are not found

class runeq.resources.stream_metadata.**Dimension**(*id: str*, *data_type: str*, *quantity_name: str*, *unit_name: str*, ***attributes*)

A dimension of a stream type. This is akin to a column in a table, where each value in the timeseries is a row.

__init__(*id: str*, *data_type: str*, *quantity_name: str*, *unit_name: str*, ***attributes*)

Initialize with metadata.

Parameters

- **id** – Dimension ID
- **data_type** – Data type (e.g. sfloat, timestamp, etc…)
- **quantity_name** – Name of the quantity measured
- **unit_name** – Name of the unit measurement

get(*attr: str*, *default: Any = None*)

Get the value of any attribute in self.attributes.

property id: str

ID of the item.

to_dict() → dict

Dictionary representation of the item's attributes.

class runeq.resources.stream_metadata.**StreamMetadata**(*id: str*, *created_at: float*, *algorithm: str*, *device_id: str*, *patient_id: str*, *stream_type:* StreamType, *min_time: float*, *max_time: float*, *parameters: dict*, ***attributes*)

Metadata for a stream (i.e. timeseries data). This class also has methods that fetch data for the stream.

__init__(*id: str*, *created_at: float*, *algorithm: str*, *device_id: str*, *patient_id: str*, *stream_type:* StreamType, *min_time: float*, *max_time: float*, *parameters: dict*, ***attributes*)

Initialize with metadata.

Parameters

- **id** – Stream ID
- **created_at** – When the stream was created, as a Unix timestamp, as a Unix timestamp (in seconds).
- **algorithm** – A versioned label that describes the process that was used to derive this timeseries.
- **device_id** – Device ID
- **patient_id** – Patient ID

- **stream_type** – Stream type, which categorizes the stream data.
- **min_time** – The earliest timestamp in the stream, as a Unix timestamp (in seconds).
- **max_time** – The latest timestamp in the stream, as a Unix timestamp (in seconds).
- **parameters** – Key/value pairs that label the stream.

get(*attr: str*, *default: Any = None*)

Get the value of any attribute in self.attributes.

get_stream_availability_dataframe(*start_time: Union[float, date]*, *end_time: Union[float, date]*, *resolution: int*, *limit: Optional[int] = None*, *page_token: Optional[str] = None*, *timestamp: Optional[str] = 'iso'*, *timezone: Optional[int] = None*, *stream_client: Optional[*StreamClient*] = None*) → DataFrame

Get stream availability as an enriched Pandas dataframe. The dataframe includes columns with metadata for this stream. This allows for the concatenation of dataframes with availability for multiple streams.

Parameters

- **start_time** – Start time for the query, provided as a unix timestamp (in seconds) or a datetime.date.
- **end_time** – End time for the query, provided as a unix timestamp (in seconds) or a datetime.date.
- **resolution** – Interval between returned timestamps, in seconds.
- **limit** – Maximum number of timestamps to return, across *all pages* of the response. A limit of 0 (default) will fetch all available data.
- **page_token** – Token to fetch the subsequent page of results. The value is obtained from the 'X-Rune-Next-Page-Token' response header field.
- **timestamp** – Optional enum "unix", "unixns", or "iso", which determines how timestamps are formatted in the response
- **timezone** – Optional timezone offset, in seconds, used to calculate string-based timestamp formats such as datetime and iso. For example, PST (UTC-0800) is represented as -28800. If omitted, the timezone is UTC.
- **stream_client** – If specified, this client is used to fetch data from the API. Otherwise, the global StreamClient is used.

get_stream_dataframe(*start_time: Optional[Union[float, date]] = None*, *start_time_ns: Optional[int] = None*, *end_time: Optional[Union[float, date]] = None*, *end_time_ns: Optional[int] = None*, *limit: Optional[int] = None*, *page_token: Optional[str] = None*, *timestamp: Optional[str] = 'iso'*, *timezone: Optional[int] = None*, *translate_enums: Optional[bool] = True*, *stream_client: Optional[*StreamClient*] = None*) → DataFrame

Get stream data as an enriched Pandas dataframe. In addition to the raw stream data, the dataframe also includes columns with stream metadata. This allows for the concatenation of dataframes with data from multiple streams.

Parameters

- **start_time** – Start time for the query, provided as a unix timestamp (in seconds) or a datetime.date.
- **start_time_ns** – Start time for the query, provided as a unix timestamp (in nanoseconds).

- **end_time** – End time for the query, provided as a unix timestamp (in seconds) or a datetime.date.
- **end_time_ns** – End time for the query, provided as a unix timestamp (in nanoseconds).format: Optional enum "json" or "csv", which determines the content type of the response
- **limit** – Maximum number of timestamps to return, across *all pages* of the response. A limit of 0 (default) will fetch all available data.
- **page_token** – Token to fetch the subsequent page of results. The value is obtained from the 'X-Rune-Next-Page-Token' response header field.
- **timestamp** – Optional enum "unix", "unixns", or "iso", which determines how timestamps are formatted in the response
- **timezone** – Optional timezone offset, in seconds, used to calculate string-based timestamp formats such as datetime and iso. For example, PST (UTC-0800) is represented as -28800. If omitted, the timezone is UTC.
- **translate_enums** – If True, enum values are returned as their string representation. Otherwise, enums are returned as integer values.
- **stream_client** – If specified, this client is used to fetch data from the API. Otherwise, the global StreamClient is used.

property id: str

ID of the item.

iter_stream_data(*start_time: Optional[Union[int, float, date, datetime]] = None*, *start_time_ns: Optional[int] = None*, *end_time: Optional[Union[int, float, date, datetime]] = None*, *end_time_ns: Optional[int] = None*, *format: Optional[str] = 'csv'*, *limit: Optional[int] = None*, *page_token: Optional[str] = None*, *timestamp: Optional[str] = 'iso'*, *timezone: Optional[int] = None*, *translate_enums: Optional[bool] = True*, *client: Optional[StreamClient] = None*) → Iterator[Union[str, dict]]

Iterate over CSV-formatted data for this stream.

Parameters

- **start_time** – Start time for the query, provided as a unix timestamp (in seconds), a datetime.datetime, or a datetime.date.
- **start_time_ns** – Start time for the query, provided as a unix timestamp (in nanoseconds).
- **end_time** – End time for the query, provided as a unix timestamp (in seconds), a datetime.datetime, or a datetime.date.
- **end_time_ns** – End time for the query, provided as a unix timestamp (in nanoseconds).
- **format** – Either "csv" (default) or "json". Determines the content type of the API response, as well as the type that is returned from this function.
- **limit** – Maximum number of timestamps to return, across *all pages* of the response. A limit of 0 (default) will fetch all available data.
- **page_token** – Token to fetch the subsequent page of results. The value is obtained from the "X-Rune-Next-Page-Token" response header field.
- **timestamp** – One of "unix", "unixns", or "iso", which determines how timestamps are formatted in the response

- **timezone** – Timezone offset, in seconds, used to calculate string-based timestamp formats such as datetime and iso. For example, PST (UTC-0800) is represented as -28800. If omitted, the timezone is UTC.
- **translate_enums** – If True, enum values are returned as their string representation. Otherwise, enums are returned as integer values.
- **client** – If specified, this client is used to fetch data from the API. Otherwise, the global *StreamClient* is used.

Returns
An iterator over paginated API responses. If format is "json", each response is a dict. If format is "csv", each response is a CSV-formatted string.

to_dict() → dict

Dictionary representation of the StreamMetadata attributes.

class runeq.resources.stream_metadata.**StreamMetadataSet**(*items: Iterable[*StreamMetadata*] = ()*)

A collection of StreamMetadata.

__init__(*items: Iterable[*StreamMetadata*] = ()*)

Initialize with StreamMetadatas

add(*item: ItemBase*)

Add a single item to the set. Must be the same type as other members of the collection

filter(*stream_id: Optional[str] = None*, *patient_id: Optional[str] = None*, *device_id: Optional[str] = None*, *stream_type_id: Optional[str] = None*, *algorithm: Optional[str] = None*, *category: Optional[str] = None*, *measurement: Optional[str] = None*, *filter_function: Optional[Callable[[*StreamMetadata*], bool]] = None*, ***parameters*) → *StreamMetadataSet*

Filters streams for those that match ALL optional filter parameters. Returns a new StreamMetadataSet.

Parameters

- **stream_id** – Stream ID
- **patient_id** – Patient ID
- **device_id** – Device ID
- **stream_type_id** – Stream type ID
- **algorithm** – A versioned label that describes the process that was used to derive this timeseries.
- **category** – A broad categorization of the data type (e.g. neural, vitals, etc)
- **measurement** – A specific label for what is being measured (e.g. heart_rate, step_count, etc).
- **filter_function** – User-defined filter function which accepts a Stream as a single argument and returns a boolean indicating whether to keep that stream.

get(*id: str*) → ItemBase

Get an item by id.

Raises
ValueError – if the set does not contain an item with the ID.

get_batch_availability_dataframe(*start_time: Union[float, date]*, *end_time: Union[float, date]*, *resolution: int*, *batch_operation: Optional[str] = None*, *limit: Optional[int] = None*, *page_token: Optional[str] = None*, *timestamp: Optional[str] = 'iso'*, *timezone: Optional[int] = None*, *stream_client: Optional[StreamClient] = None*) → DataFrame

Get stream availability data as a Pandas dataframe. Depending on the specified **batch_operation**, this fetches the availability of **all** or **any** of the streams in the collection.

If you want to gather the availability of each *individual* stream in a dataframe, see get_stream_availability_dataframe()

Parameters

- **start_time** – Start time for the query, provided as a unix timestamp (in seconds) or a datetime.date.
- **end_time** – End time for the query, provided as a unix timestamp (in seconds) or a datetime.date.
- **resolution** – Interval between returned timestamps, in seconds.
- **batch_operation** – Either "any" or "all", which determines what type of batch calculation will determine availability for the batch of streams. Availability values will equal 1 when data is available for "all" or "any" of the requested streams in the given interval.
- **limit** – Maximum number of timestamps to return, across *all pages* of the response. A limit of 0 (default) will fetch all available data.
- **page_token** – Token to fetch the subsequent page of results. The value is obtained from the 'X-Rune-Next-Page-Token' response header field.
- **timestamp** – Optional enum "unix", "unixns", or "iso", which determines how timestamps are formatted in the response
- **timezone** – Optional timezone offset, in seconds, used to calculate string-based timestamp formats such as datetime and iso. For example, PST (UTC-0800) is represented as -28800. If omitted, the timezone is UTC.
- **stream_client** – If specified, this client is used to fetch data from the API. Otherwise, the global StreamClient is used.

get_stream_dataframe(*start_time: Optional[Union[float, date]] = None*, *start_time_ns: Optional[int] = None*, *end_time: Optional[Union[float, date]] = None*, *end_time_ns: Optional[int] = None*, *limit: Optional[int] = None*, *page_token: Optional[str] = None*, *timestamp: Optional[str] = 'iso'*, *timezone: Optional[int] = None*, *translate_enums: Optional[bool] = True*, *stream_client: Optional[StreamClient] = None*) → DataFrame

Get raw data for all streams in the collection, as an enriched Pandas dataframe. The dataframe includes columns with metadata for each stream.

Parameters

- **start_time** – Start time for the query, provided as a unix timestamp (in seconds) or a datetime.date.
- **start_time_ns** – Start time for the query, provided as a unix timestamp (in nanoseconds).
- **end_time** – End time for the query, provided as a unix timestamp (in seconds) or a datetime.date.

- **end_time_ns** – End time for the query, provided as a unix timestamp (in nanoseconds).format: Optional enum "json" or "csv", which determines the content type of the response
- **limit** – Maximum number of timestamps to return, across *all pages* of the response. A limit of 0 (default) will fetch all available data.
- **page_token** – Token to fetch the subsequent page of results. The value is obtained from the 'X-Rune-Next-Page-Token' response header field.
- **timestamp** – Optional enum "unix", "unixns", or "iso", which determines how timestamps are formatted in the response
- **timezone** – Optional timezone offset, in seconds, used to calculate string-based timestamp formats such as datetime and iso. For example, PST (UTC-0800) is represented as -28800. If omitted, the timezone is UTC.
- **translate_enums** – If True, enum values are returned as their string representation. Otherwise, enums are returned as integer values.
- **stream_client** – If specified, this client is used to fetch data from the API. Otherwise, the global StreamClient is used.

ids() → Iterator[str]

Iterator over the IDs of the items in this set.

remove(**items: Union[str, ItemBase]*)

Remove item(s) from this set.

to_dataframe() → DataFrame

Convert items to a dataframe (wraps *to_list()*)

to_list() → List[dict]

List of all the items in the set. Each item is formatted as a dictionary, using the item's *to_dict()* method.

update(*items: Iterable[ItemBase]*)

Add an iterable of item(s) to this set. All items must be the same class as members of this collection.

class runeq.resources.stream_metadata.**StreamType**(*id: str*, *name: str*, *description: str*, *dimensions: List[*Dimension*]*, ***attributes*)

A stream type to categorize streams.

It represents the physical quantity being measured (voltage, acceleration, etc), including the unit of measurement. It also describes the shape of the stream's data, as one or more dimensions.

__init__(*id: str*, *name: str*, *description: str*, *dimensions: List[*Dimension*]*, ***attributes*)

Initialize with metadata.

Parameters

- **id** – StreamType ID
- **name** – Human-readable name of the stream type
- **description** – Human-readable description of the stream type
- **dimensions** – List of Dimensions in this stream type

get(*attr: str*, *default: Any = None*)

Get the value of any attribute in self.attributes.

property id: str

ID of the item.

to_dict() → dict

Dictionary representation of the Stream Type attributes.

class runeq.resources.stream_metadata.**StreamTypeSet**(*items: Iterable[StreamType] = ()*)

A collection of StreamTypes.

__init__(*items: Iterable[StreamType] = ()*)

Initialize with StreamTypes.

add(*item: ItemBase*)

Add a single item to the set. Must be the same type as other members of the collection

get(*id: str*) → ItemBase

Get an item by id.

Raises

ValueError – if the set does not contain an item with the ID.

ids() → Iterator[str]

Iterator over the IDs of the items in this set.

remove(**items: Union[str, ItemBase]*)

Remove item(s) from this set.

to_dataframe() → DataFrame

Convert items to a dataframe (wraps *to_list()*)

to_list() → List[dict]

List of all the items in the set. Each item is formatted as a dictionary, using the item's *to_dict()* method.

update(*items: Iterable[ItemBase]*)

Add an iterable of item(s) to this set. All items must be the same class as members of this collection.

2.3.5 Stream Data

Query data directly from the V2 Stream API.

runeq.resources.stream.**get_stream_availability**(*stream_ids: Union[str, Iterable[str]]*, *start_time: Union[int, float, date, datetime]*, *end_time: Union[int, float, date, datetime]*, *resolution: int*, *batch_operation: Optional[str] = None*, *format: Optional[str] = 'csv'*, *limit: Optional[int] = None*, *page_token: Optional[str] = None*, *timestamp: Optional[str] = 'iso'*, *timezone: Optional[int] = None*, *timezone_name: Optional[str] = None*, *client: Optional[StreamClient] = None*) → Iterator[Union[str, dict]]

Fetch the availability of 1 or multiple streams. When multiple stream IDs are specified, this fetches the availability of **all** or **any** of the streams (depending on the **batch_operation**).

Parameters

- **stream_ids** – 1 or multiple stream IDs. If multiple stream IDs are specified, **batch_operation** is also required.

- **start_time** – Start time for the query, provided as a unix timestamp (in seconds), a datetime.datetime, or a datetime.date.
- **end_time** – End time for the query, provided as a unix timestamp (in seconds), a datetime.datetime, or a datetime.date.
- **resolution** – Interval between returned timestamps, in seconds.
- **batch_operation** – Either "any" or "all", which determines what type of batch calculation will determine availability for multiple streams. Availability values will equal 1 when data is available for "all" or "any" of the requested streams in the given interval. This argument is required when multiple **stream_ids** are specified.
- **format** – Either "csv" (default) or "json". Determines the content type of the API response.
- **limit** – Maximum number of timestamps to return, across *all pages* of the response. A limit of 0 (default) will fetch all available data.
- **page_token** – Token to fetch the subsequent page of results. The value is obtained from the "X-Rune-Next-Page-Token" response header field.
- **timestamp** – One of "unix", "unixns", or "iso", which determines how timestamps are formatted in the response
- **timezone** – Timezone offset, in seconds, used to calculate string-based timestamp formats such as datetime and iso. For example, PST (UTC-0800) is represented as -28800. If omitted, the timezone is UTC.
- **timezone_name** – The name from the IANA timezone database used to calculate string-based timestamp formats such as datetime and iso. Returns the correct UTC offset for a given date/time in order to account for daylight savings time.
- **client** – If specified, this client is used to fetch data from the API. Otherwise, the global *StreamClient* is used.

Returns

An iterator over paginated API responses. If format is "json", each response is a dict. If format is "csv", each response is a CSV-formatted string.

Raises

ValueError – if batch_operation is not specified and querying for more than 1 stream_id

runeq.resources.stream.**get_stream_data**(*stream_id: str*, *start_time: Optional[Union[int, float, date, datetime]] = None*, *start_time_ns: Optional[int] = None*, *end_time: Optional[Union[int, float, date, datetime]] = None*, *end_time_ns: Optional[int] = None*, *format: Optional[str] = 'csv'*, *limit: Optional[int] = None*, *page_token: Optional[str] = None*, *timestamp: Optional[str] = 'iso'*, *timezone: Optional[int] = None*, *timezone_name: Optional[str] = None*, *translate_enums: Optional[bool] = True*, *client: Optional[*StreamClient*] = None*) → Iterator[Union[str, dict]]

Fetch raw data for a stream.

Parameters

- **stream_id** – ID of the stream
- **start_time** – Start time for the query, provided as a unix timestamp (in seconds), a datetime.datetime, or a datetime.date.
- **start_time_ns** – Start time for the query, provided as a unix timestamp (in nanoseconds).

- **end_time** – End time for the query, provided as a unix timestamp (in seconds), a datetime.datetime, or a datetime.date.
- **end_time_ns** – End time for the query, provided as a unix timestamp (in nanoseconds).
- **format** – Either "csv" (default) or "json". Determines the content type of the API response.
- **limit** – Maximum number of timestamps to return, across *all pages* of the response. A limit of 0 (default) will fetch all available data.
- **page_token** – Token to fetch the subsequent page of results. The value is obtained from the "X-Rune-Next-Page-Token" response header field.
- **timestamp** – One of "unix", "unixns", or "iso", which determines how timestamps are formatted in the response
- **timezone** – Timezone offset, in seconds, used to calculate string-based timestamp formats such as datetime and iso. For example, PST (UTC-0800) is represented as -28800. If omitted, the timezone is UTC.
- **timezone_name** – The name from the IANA timezone database used to calculate string-based timestamp formats such as datetime and iso. Returns the correct UTC offset for a given date/time in order to account for daylight savings time.
- **translate_enums** – If True, enum values are returned as their string representation. Otherwise, enums are returned as integer values.
- **client** – If specified, this client is used to fetch data from the API. Otherwise, the global *StreamClient* is used.

Returns

An iterator over paginated API responses. If format is "json", each response is a dict. If format is "csv", each response is a CSV-formatted string.

2.3.6 User Metadata

Fetch metadata about Rune platform users.

runeq.resources.user.**get_current_user**(*client: Optional[*GraphClient*] = None*) → *User*

Get information about the current user (based on the API credentials).

Parameters

client – If specified, this client is used to fetch metadata from the API. Otherwise, the global GraphClient is used.

class runeq.resources.user.**User**(*id: str*, *name: str*, *created_at: float*, *active_org_id: str*, *active_org_name: str*, ***attributes*)

A user of the Rune platform.

__init__(*id: str*, *name: str*, *created_at: float*, *active_org_id: str*, *active_org_name: str*, ***attributes*)

Initialize with metadata.

Parameters

- **id** – User ID
- **name** – Human-readable display name for the user
- **created_at** – When the user was created (unix timestamp)
- **active_org_id** – ID of the user's currently active organization

- **active_org_name** – Display name of the user's currently active organization

get(*attr: str*, *default: Any = None*)

Get the value of any attribute in self.attributes.

property id: str

ID of the item.

static normalize_id(*user_id: str*) → str

Strip resource prefix and suffix from the user ID (if they exist).

Parameters
user_id – User ID

to_dict() → dict

Dictionary representation of the item's attributes.

2.3.7 Project Metadata

Fetch metadata about projects.

Projects

runeq.resources.project.**get_projects**(*client: Optional[GraphClient] = None*) → *ProjectSet*

Get all the projects that the current user has access to.

Parameters
client – If specified, this client is used to fetch metadata from the API. Otherwise, the global GraphClient is used.

runeq.resources.project.**get_project**(*project_id: str*, *client: Optional[GraphClient] = None*) → Project

Get the project with the specified ID.

Parameters

- **project_id** – Project ID
- **client** – If specified, this client is used to fetch metadata from the API. Otherwise, the global GraphClient is used.

runeq.resources.project.**get_project_patients**(*project_id: str*, *client: Optional[GraphClient] = None*) → *ProjectPatientMetadataSet*

Get all patients in a project and their associated project data metrics.

Parameters

- **project_id** – ID of the project
- **client** – If specified, this client is used to fetch metadata from the API. Otherwise, the global GraphClient is used.

```
Project(id: str, title: str, status: str, type: str, started_at: float,
updated_at: float, created_at: float, created_by: str, updated_by: str,
cohorts: CohortSet {
```

Project}, **attributes)

Metadata for a project.

A project is a generic container for a group of patients with workflow-related metadata: status, project type, description, milestones dates, etc. Data availability QC metrics are computed on a regular basis for all patients in a project.

Project.__init__(id: str, title: str, status: str, type: str, started_at: float, updated_at: float, created_at: float, created_by: str, updated_by: str, cohorts: CohortSet {

Project.__init__}, **attributes)

Initialize with data.

Parameters

- **id** – ID of the project
- **title** – Human-readable name
- **status** – Status of the project
- **type** – Type of project. Possible types include: EXPLORATORY, CLINICAL_TRIAL,RETROSPECTIVE_STUDY, PROSPECTIVE_STUDY, SANDBOX
- **started_at** – Time the project started (unix timestamp)
- **created_at** – Time the project was created (unix timestamp)
- **created_by** – Display name of who created the project
- **updated_at** – Time the project was last updated (unix timestamp)
- **updated_by** – Display name of who updated the project
- **cohorts** – Sub-containers of patients in a project
- ****attributes** – Other attributes associated with the project

Project.**get**(*attr: str*, *default: Any = None*)

Get the value of any attribute in self.attributes.

property Project.**id: str**

ID of the item.

Project.**to_dict**() → dict

Dictionary representation of the Project attributes.

class runeq.resources.project.**ProjectSet**(*items: Iterable[Project] = ()*)

A collection of Projects.

__init__(*items: Iterable[Project] = ()*)

Initialize with Projects.

add(*item: ItemBase*)

Add a single item to the set. Must be the same type as other members of the collection

get(*id: str*) → ItemBase

Get an item by id.

Raises

ValueError – if the set does not contain an item with the ID.

ids() → Iterator[str]

Iterator over the IDs of the items in this set.

remove(**items: Union[str, ItemBase]*)

Remove item(s) from this set.

to_dataframe() → DataFrame

Convert items to a dataframe (wraps *to_list()*)

to_list() → List[dict]

List of all the items in the set. Each item is formatted as a dictionary, using the item's *to_dict()* method.

update(*items: Iterable[ItemBase]*)

Add an iterable of item(s) to this set. All items must be the same class as members of this collection.

class runeq.resources.project.**ProjectPatientMetadata**(*id: str*, *updated_at: float*, *created_at: float*, *created_by: str*, *updated_by: str*, *metrics:* MetricSet, ***attributes*)

A patient who has been placed within a project and is now associated with the project.

__init__(*id: str*, *updated_at: float*, *created_at: float*, *created_by: str*, *updated_by: str*, *metrics:* MetricSet, ***attributes*)

Initialize with data.

Parameters

- **id** – Patient ID of the patient in the cohort
- **created_at** – Time patient was added to the project (unix timestamp)
- **created_by** – Display name of who added the patient to the project
- **updated_at** – Time project patient was last updated (unix timestamp)
- **updated_by** – Display name of who updated the project patient record
- **metrics** – Project data metrics related to the patient's data
- ****attributes** – Other attributes associated with the project

get(*attr: str*, *default: Any = None*)

Get the value of any attribute in self.attributes.

get_patient_metadata_dataframe() → DataFrame

Returns a new dataframe displaying the patient's metadata.

property id: str

ID of the item.

to_dict() → dict

Dictionary representation of the Project Patient attributes.

class runeq.resources.project.**ProjectPatientMetadataSet**(*items: Iterable[*ProjectPatientMetadata*] = ()*)

A collection of ProjectPatientMetadata.

__init__(*items: Iterable[*ProjectPatientMetadata*] = ()*)

Initialize with ProjectPatientMetadata.

add(*item: ItemBase*)

Add a single item to the set. Must be the same type as other members of the collection

get(*id: str*) → ItemBase

Get an item by id.

Raises

ValueError – if the set does not contain an item with the ID.

ids() → Iterator[str]

Iterator over the IDs of the items in this set.

remove(**items: Union[str, ItemBase]*)

Remove item(s) from this set.

to_dataframe() → DataFrame

Convert items to a dataframe (wraps *to_list()*)

to_list() → List[dict]

List of all the items in the set. Each item is formatted as a dictionary, using the item's *to_dict()* method.

update(*items: Iterable[ItemBase]*)

Add an iterable of item(s) to this set. All items must be the same class as members of this collection.

Cohorts

runeq.resources.project.**get_cohort_patients**(*cohort_id: str*, *client: Optional[*GraphClient*] = None*) → *CohortPatientMetadataSet*

Get all patients in a cohort.

Parameters

- **cohort_id** – ID of the cohort
- **client** – If specified, this client is used to fetch metadata from the API. Otherwise, the global GraphClient is used.

class runeq.resources.project.**Cohort**(*id: str*, *title: str*, *updated_at: float*, *created_at: float*, *created_by: str*, *updated_by: str*, ***attributes*)

A generic sub-container for a group of patients within a project.

A cohort is a generic container within a project for a group of patients with workflow-related metadata: description, milestones dates, metrics, etc.

A single project can have multiple cohorts.

__init__(*id: str*, *title: str*, *updated_at: float*, *created_at: float*, *created_by: str*, *updated_by: str*, ***attributes*)

Initialize with data.

Parameters

- **id** – ID of the project
- **title** – Human-readable name
- **created_at** – Time the cohort was created (unix timestamp)
- **created_by** – Display name of who created the cohort
- **updated_at** – Time the cohort was last updated (unix timestamp)
- **updated_by** – Display name of who updated the cohort

- ****attributes** – Other attributes associated with the cohort

get(*attr: str*, *default: Any = None*)

Get the value of any attribute in self.attributes.

property id: str

ID of the item.

to_dict() → dict

Dictionary representation of the item's attributes.

class runeq.resources.project.**CohortSet**(*items: Iterable[Cohort] = ()*)

A collection of Cohorts.

__init__(*items: Iterable[Cohort] = ()*)

Initialize with Cohorts.

add(*item: ItemBase*)

Add a single item to the set. Must be the same type as other members of the collection

get(*id: str*) → ItemBase

Get an item by id.

Raises

ValueError – if the set does not contain an item with the ID.

ids() → Iterator[str]

Iterator over the IDs of the items in this set.

remove(**items: Union[str, ItemBase]*)

Remove item(s) from this set.

to_dataframe() → DataFrame

Convert items to a dataframe (wraps *to_list()*)

to_list() → List[dict]

List of all the items in the set. Each item is formatted as a dictionary, using the item's *to_dict()* method.

update(*items: Iterable[ItemBase]*)

Add an iterable of item(s) to this set. All items must be the same class as members of this collection.

class runeq.resources.project.**CohortPatientMetadata**(*id: str*, *updated_at: float*, *created_at: float*, *created_by: str*, *updated_by: str*, *metrics:* MetricSet, ***attributes*)

Cohort related information about a patient contained in a cohort.

__init__(*id: str*, *updated_at: float*, *created_at: float*, *created_by: str*, *updated_by: str*, *metrics:* MetricSet, ***attributes*)

Initialize with data.

Parameters

- **id** – Patient ID of the patient in the cohort
- **created_at** – Time patient was added to the cohort (unix timestamp)
- **created_by** – Display name of who added the patient to the cohort
- **updated_at** – Time cohort patient was last updated (unix timestamp)
- **updated_by** – Display name of who updated the cohort patient record

- **metrics** – Project data metrics related to the patient's data
- ****attributes** – Other attributes associated with the cohort

get(*attr: str*, *default: Any = None*)

Get the value of any attribute in self.attributes.

get_patient_metadata_dataframe() → DataFrame

Returns a new dataframe displaying the patient's metadata.

property id: str

ID of the item.

to_dict() → dict

Dictionary representation of the Project Patient attributes.

class runeq.resources.project.**CohortPatientMetadataSet**(*items: Iterable[CohortPatientMetadata] = ()*)

A collection of CohortPatientMetadata.

__init__(*items: Iterable[CohortPatientMetadata] = ()*)

Initialize with CohortPatientMetadata.

add(*item: ItemBase*)

Add a single item to the set. Must be the same type as other members of the collection

get(*id: str*) → ItemBase

Get an item by id.

Raises

ValueError – if the set does not contain an item with the ID.

ids() → Iterator[str]

Iterator over the IDs of the items in this set.

remove(**items: Union[str, ItemBase]*)

Remove item(s) from this set.

to_dataframe() → DataFrame

Convert items to a dataframe (wraps *to_list()*)

to_list() → List[dict]

List of all the items in the set. Each item is formatted as a dictionary, using the item's *to_dict()* method.

update(*items: Iterable[ItemBase]*)

Add an iterable of item(s) to this set. All items must be the same class as members of this collection.

Metrics

class runeq.resources.project.**Metric**(*id: str*, *type: str*, *data_type: str*, *time_interval: str*, *updated_at: float*, *created_at: float*, *value: Optional[float] = None*)

A measurement related to processed patient data. Each metric is related to a single processed data type, project, and time interval. These metrics are calculated periodically and updated.

__init__(*id: str*, *type: str*, *data_type: str*, *time_interval: str*, *updated_at: float*, *created_at: float*, *value: Optional[float] = None*)

Initialize with data.

Parameters

- **id** – Unique identifier of the patient
- **type** – Type of measurement/metric. Possible types include: TOTAL_HOURS, LATEST_DATA, LAST_UPLOAD
- **data_type** – Processed stream data type that is being measured. Possible data types include: APPLEWATCH_SYMPTOM, APPLEWATCH_TREMOR, APPLEWATCH_DYSKINESIA, APPLEWATCH_HEART_RATE, PERCEPT_TREND_LOG_LFP
- **value** – Value of the metric (float)
- **time_interval** – Period over which the metric was calculated. Possible time intervals include: FOURTEEN_DAYS, NINETY_DAYS, PROJECT_ALL
- **created_at** – Time patient was added to the cohort (unix timestamp)
- **updated_at** – Time cohort patient was last updated (unix timestamp)

get(*attr: str*, *default: Any = None*)

Get the value of any attribute in self.attributes.

property id: str

ID of the item.

to_dict() → dict

Dictionary representation of the item's attributes.

class runeq.resources.project.**MetricSet**(*items: Iterable[Metric] = ()*)

A collection of Metrics.

__init__(*items: Iterable[Metric] = ()*)

Initialize with Metrics.

add(*item: ItemBase*)

Add a single item to the set. Must be the same type as other members of the collection

get(*id: str*) → ItemBase

Get an item by id.

Raises

ValueError – if the set does not contain an item with the ID.

ids() → Iterator[str]

Iterator over the IDs of the items in this set.

remove(**items: Union[str, ItemBase]*)

Remove item(s) from this set.

to_dataframe() → DataFrame

Convert items to a dataframe (wraps *to_list()*)

to_list() → List[dict]

List of all the items in the set. Each item is formatted as a dictionary, using the item's *to_dict()* method.

update(*items: Iterable[ItemBase]*)

Add an iterable of item(s) to this set. All items must be the same class as members of this collection.

2.4 V1 Stream API

The *stream* module contains classes that fetch timeseries data using V1 Stream API.

Warning: The V1 API is supported, but it is not longer under active development. We recommend using the *resources* module instead, which provides access to data from the V2 Stream API.

However, you should be aware that the data returned by the V1 and V2 APIs are independent. Some types of data may only be available from one API or the other. If you have questions about which API to use, please contact Rune Support.

2.4.1 Usage

Initialization

The global initialization method, as described in the Quickstart (*Initialization*), does NOT apply to requests made by the *stream* module.

Start by creating a *V1Client*, using a *Config*. The configuration class uses the file that was created via the command line tool (see *Configuration Setup*).

```
from runeq import Config, stream

cfg = Config()
v1client = stream.V1Client(cfg)
```

Methods on the *V1Client* are used to create **accessors** for each type of data that can be queried using the V1 Stream API (e.g. accelerometry, events, local field potentials, etc).

The example below initializes an *Accel* class, which will allow us to fetch accelerometry data:

```
accel = v1client.Accel(
    patient_id='992967a09cad48378f7b628aff5bdf6c',
    device_id='ABCDEF',
    start_time=1562482800,
    end_time=1563692400,
)
```

Accessors can be initialized with default query parameters, which will be used for all API requests.

JSON Endpoints

An accessor can be used to iterate over paginated data from the respective JSON endpoint:

```
for result in accel.iter_json_data():
     print(result.keys())
```

To override the default query parameters, use keyword arguments:

```
for text in accel.iter_json_data(device_id='patient-ABC,device-456'):
    pass  # do something
```

CSV Endpoints

An accessor can also iterate over paginated data from the respective CSV endpoint.

Here, we use the accessor to build up a pandas DataFrame, containing the complete result set.

```
import io
import pandas as pd

df = pd.DataFrame()
for text in accel.iter_csv_text():
    page_df = pd.read_csv(io.StringIO(text))
    df.append(page_df)
```

We can also iterate over each point from the CSV response. Each line from the CSV is returned as a dict:

```
for point in accel.points():
    print(point)

# the accessor itself is also an iterator
for point in accel:
    print(point)
```

To override the default query parameters, use keyword arguments:

```
for point in accel.points(end_time=1563692400):
    pass  # do something

for text in accel.iter_csv_text(device_id='patient-ABC,device-456'):
    pass  # do something

# etc
```

Note that CSV-formatted data is not supported for all resources: refer to the API documentation for details.

2.4.2 V1Client

class runeq.stream.**V1Client**(*cfg:* Config)

V1Client is a factory class. It holds configuration, which is used to initialize accessor classes for Stream V1 endpoints.

Accel(***defaults*) → *Accel*

Initialize an Accel accessor.

Parameters
****defaults** – Default query parameters

BandPower(***defaults*) → *BandPower*

Initialize a BandPower accessor.

Parameters
****defaults** – Default query parameters

Event(***defaults*) → *Event*

Initialize an Event accessor.

Parameters
****defaults** – Default query parameters

HeartRate(***defaults*) → *HeartRate*

Initialize a HeartRate accessor.

Parameters
****defaults** – Default query parameters

LFP(***defaults*) → *LFP*

Initialize an LFP accessor.

Parameters
****defaults** – Default query parameters

ProbabilitySymptom(***defaults*) → *ProbabilitySymptom*

Initialize a ProbabilitySymptom accessor.

Parameters
****defaults** – Default query parameters

Rotation(***defaults*) → *Rotation*

Initialize a Rotation accessor.

Parameters
****defaults** – Default query parameters

Span(***defaults*) → *Span*

Initialize a Span accessor.

Parameters
****defaults** – Default query parameters

State(***defaults*) → *State*

Initialize a State accessor.

Parameters
****defaults** – Default query parameters

property config: *Config*

Return configuration.

2.4.3 Accessors

Query data from the Rune Labs Stream API (V1).

Each resource that is exposed by the Stream API can be queried through an accessor class.

Accel

class runeq.stream.v1.**Accel**(*cfg:* Config, ***defaults*)

Query accelerometry data streams.

property expr_availability

Availability expression for this resource.

Raises
NotImplementedError – if the resource doesn't support an availability query expression.

get_csv_response(**_params_) → Response

Make a GET request to the resource's CSV endpoint.

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Returns
requests.Response

get_json_response(**_params_) → Response

Make a GET request to the resource's JSON endpoint.

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis, for this request.

Returns
requests.Response

iter_csv_availability(**_params_) → Iterator[dict]

Convenience method to query the CSV endpoint, using the availability expression. May not be supported for all resources.

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Yields
text body of each response.

Raises
NotImplementedError – if the resource doesn't support an availability query expression.

iter_csv_text(**_params_) → Iterator[str]

Iterate over CSV text results, from the resource's CSV endpoint.

Follows pagination to get a complete set of results, starting with the page specified in the *page* kwarg (or the first page, by default).

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Yields
text body of each response.

Raises
APIError – when a request fails

iter_json_availability(**_params_) → Iterator[dict]

Convenience method to query the JSON endpoint, using the availability expression. This may not be supported for all endpoints.

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Yields
dict with the "result" from the JSON body of each response.

Raises
NotImplementedError – if the resource doesn't support an availability query expression.

`iter_json_data`(***params*) → Iterator[dict]

Iterate over results from the resource's JSON endpoint.

Follows pagination to get a complete set of results, starting with the page specified in the *page* kwarg (or the first page, by default).

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Yields
dict with the "result" from the JSON body of each response.

Raises
APIError – when a request fails

`points`(***params*) → Iterator[dict]

Iterate over points from CSV response, yielding dictionaries.

This may involve multiple requests to the CSV endpoint, to follow pagination.

Parameters
****params** – query parameters for the request(s). These override self.defaults on a key-by-key basis.

Yields
dict – Keys are the headers from the CSV response. Values are converted to numeric types where applicable. If numpy is available, np.float64 is used.

BandPower

`class runeq.stream.v1.BandPower`(*cfg:* Config, ***defaults*)

Query band power data streams.

`property expr_availability`

Availability expression for this resource.

Raises
NotImplementedError – if the resource doesn't support an availability query expression.

`get_csv_response`(***params*) → Response

Make a GET request to the resource's CSV endpoint.

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Returns
requests.Response

`get_json_response`(***params*) → Response

Make a GET request to the resource's JSON endpoint.

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis, for this request.

Returns
requests.Response

`iter_csv_availability`(***params*) → Iterator[dict]

Convenience method to query the CSV endpoint, using the availability expression. May not be supported for all resources.

Parameters
`**params` – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Yields
text body of each response.

Raises
`NotImplementedError` – if the resource doesn't support an availability query expression.

`iter_csv_text`(***params*) → Iterator[str]

Iterate over CSV text results, from the resource's CSV endpoint.

Follows pagination to get a complete set of results, starting with the page specified in the *page* kwarg (or the first page, by default).

Parameters
`**params` – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Yields
text body of each response.

Raises
`APIError` – when a request fails

`iter_json_availability`(***params*) → Iterator[dict]

Convenience method to query the JSON endpoint, using the availability expression. This may not be supported for all endpoints.

Parameters
`**params` – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Yields
dict with the "result" from the JSON body of each response.

Raises
`NotImplementedError` – if the resource doesn't support an availability query expression.

`iter_json_data`(***params*) → Iterator[dict]

Iterate over results from the resource's JSON endpoint.

Follows pagination to get a complete set of results, starting with the page specified in the *page* kwarg (or the first page, by default).

Parameters
`**params` – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Yields
dict with the "result" from the JSON body of each response.

Raises
`APIError` – when a request fails

points(**_params_) → Iterator[dict]

Iterate over points from CSV response, yielding dictionaries.

This may involve multiple requests to the CSV endpoint, to follow pagination.

Parameters
****params** – query parameters for the request(s). These override self.defaults on a key-by-key basis.

Yields
dict – Keys are the headers from the CSV response. Values are converted to numeric types where applicable. If numpy is available, np.float64 is used.

Event

class runeq.stream.v1.**Event**(*cfg:* Config, ***defaults*)

Query patient events.

property expr_availability

Availability expression for this resource.

Raises
NotImplementedError – if the resource doesn't support an availability query expression.

get_json_response(**_params_) → Response

Make a GET request to the resource's JSON endpoint.

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis, for this request.

Returns
requests.Response

iter_json_availability(**_params_) → Iterator[dict]

Convenience method to query the JSON endpoint, using the availability expression. This may not be supported for all endpoints.

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Yields
dict with the "result" from the JSON body of each response.

Raises
NotImplementedError – if the resource doesn't support an availability query expression.

iter_json_data(**_params_) → Iterator[dict]

Iterate over results from the resource's JSON endpoint.

Follows pagination to get a complete set of results, starting with the page specified in the *page* kwarg (or the first page, by default).

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Yields
dict with the "result" from the JSON body of each response.

Raises
APIError – when a request fails

HeartRate

class runeq.stream.v1.**HeartRate**(*cfg:* Config, ***defaults*)

Query heart rate data streams.

property expr_availability

Availability expression for this resource.

Raises
NotImplementedError – if the resource doesn't support an availability query expression.

get_csv_response(***params*) → Response

Make a GET request to the resource's CSV endpoint.

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Returns
requests.Response

get_json_response(***params*) → Response

Make a GET request to the resource's JSON endpoint.

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis, for this request.

Returns
requests.Response

iter_csv_availability(***params*) → Iterator[dict]

Convenience method to query the CSV endpoint, using the availability expression. May not be supported for all resources.

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Yields
text body of each response.

Raises
NotImplementedError – if the resource doesn't support an availability query expression.

iter_csv_text(***params*) → Iterator[str]

Iterate over CSV text results, from the resource's CSV endpoint.

Follows pagination to get a complete set of results, starting with the page specified in the *page* kwarg (or the first page, by default).

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Yields
text body of each response.

Raises
APIError – when a request fails

`iter_json_availability`(***params*) → Iterator[dict]

Convenience method to query the JSON endpoint, using the availability expression. This may not be supported for all endpoints.

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Yields
dict with the "result" from the JSON body of each response.

Raises
NotImplementedError – if the resource doesn't support an availability query expression.

`iter_json_data`(***params*) → Iterator[dict]

Iterate over results from the resource's JSON endpoint.

Follows pagination to get a complete set of results, starting with the page specified in the *page* kwarg (or the first page, by default).

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Yields
dict with the "result" from the JSON body of each response.

Raises
APIError – when a request fails

`points`(***params*) → Iterator[dict]

Iterate over points from CSV response, yielding dictionaries.

This may involve multiple requests to the CSV endpoint, to follow pagination.

Parameters
****params** – query parameters for the request(s). These override self.defaults on a key-by-key basis.

Yields
dict – Keys are the headers from the CSV response. Values are converted to numeric types where applicable. If numpy is available, np.float64 is used.

LFP

`class runeq.stream.v1.LFP`(*cfg:* Config, ***defaults*)

Query local field potential (LFP) data streams.

`property expr_availability`

Availability expression for this resource.

Raises
NotImplementedError – if the resource doesn't support an availability query expression.

`get_csv_response`(***params*) → Response

Make a GET request to the resource's CSV endpoint.

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Returns
requests.Response

`get_json_response`(***params*) → Response

Make a GET request to the resource's JSON endpoint.

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis, for this request.

Returns
requests.Response

`iter_csv_availability`(***params*) → Iterator[dict]

Convenience method to query the CSV endpoint, using the availability expression. May not be supported for all resources.

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Yields
text body of each response.

Raises
NotImplementedError – if the resource doesn't support an availability query expression.

`iter_csv_text`(***params*) → Iterator[str]

Iterate over CSV text results, from the resource's CSV endpoint.

Follows pagination to get a complete set of results, starting with the page specified in the *page* kwarg (or the first page, by default).

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Yields
text body of each response.

Raises
APIError – when a request fails

`iter_json_availability`(***params*) → Iterator[dict]

Convenience method to query the JSON endpoint, using the availability expression. This may not be supported for all endpoints.

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Yields
dict with the "result" from the JSON body of each response.

Raises
NotImplementedError – if the resource doesn't support an availability query expression.

iter_json_data(**params) → Iterator[dict]

Iterate over results from the resource's JSON endpoint.

Follows pagination to get a complete set of results, starting with the page specified in the *page* kwarg (or the first page, by default).

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Yields
dict with the "result" from the JSON body of each response.

Raises
APIError – when a request fails

points(**params) → Iterator[dict]

Iterate over points from CSV response, yielding dictionaries.

This may involve multiple requests to the CSV endpoint, to follow pagination.

Parameters
****params** – query parameters for the request(s). These override self.defaults on a key-by-key basis.

Yields
dict – Keys are the headers from the CSV response. Values are converted to numeric types where applicable. If numpy is available, np.float64 is used.

ProbabilitySymptom

class runeq.stream.v1.**ProbabilitySymptom**(*cfg:* Config, **defaults*)

Query the probability of a symptom.

property expr_availability

Availability expression for this resource.

Raises
NotImplementedError – if the resource doesn't support an availability query expression.

get_csv_response(**params) → Response

Make a GET request to the resource's CSV endpoint.

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Returns
requests.Response

get_json_response(**params) → Response

Make a GET request to the resource's JSON endpoint.

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis, for this request.

Returns
requests.Response

`iter_csv_availability`(**params*) → Iterator[dict]

Convenience method to query the CSV endpoint, using the availability expression. May not be supported for all resources.

Parameters
`**params` – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Yields
text body of each response.

Raises
`NotImplementedError` – if the resource doesn't support an availability query expression.

`iter_csv_text`(**params*) → Iterator[str]

Iterate over CSV text results, from the resource's CSV endpoint.

Follows pagination to get a complete set of results, starting with the page specified in the *page* kwarg (or the first page, by default).

Parameters
`**params` – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Yields
text body of each response.

Raises
APIError – when a request fails

`iter_json_availability`(**params*) → Iterator[dict]

Convenience method to query the JSON endpoint, using the availability expression. This may not be supported for all endpoints.

Parameters
`**params` – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Yields
dict with the "result" from the JSON body of each response.

Raises
`NotImplementedError` – if the resource doesn't support an availability query expression.

`iter_json_data`(**params*) → Iterator[dict]

Iterate over results from the resource's JSON endpoint.

Follows pagination to get a complete set of results, starting with the page specified in the *page* kwarg (or the first page, by default).

Parameters
`**params` – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Yields
dict with the "result" from the JSON body of each response.

Raises
APIError – when a request fails

points(**_params_) → Iterator[dict]

Iterate over points from CSV response, yielding dictionaries.

This may involve multiple requests to the CSV endpoint, to follow pagination.

Parameters
****params** – query parameters for the request(s). These override self.defaults on a key-by-key basis.

Yields
dict – Keys are the headers from the CSV response. Values are converted to numeric types where applicable. If numpy is available, np.float64 is used.

Rotation

class runeq.stream.v1.**Rotation**(*cfg:* Config, ***defaults*)

Query rotation data streams.

property expr_availability

Availability expression for this resource.

Raises
NotImplementedError – if the resource doesn't support an availability query expression.

get_csv_response(**_params_) → Response

Make a GET request to the resource's CSV endpoint.

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Returns
requests.Response

get_json_response(**_params_) → Response

Make a GET request to the resource's JSON endpoint.

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis, for this request.

Returns
requests.Response

iter_csv_availability(**_params_) → Iterator[dict]

Convenience method to query the CSV endpoint, using the availability expression. May not be supported for all resources.

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Yields
text body of each response.

Raises
NotImplementedError – if the resource doesn't support an availability query expression.

`iter_csv_text`(**params) → Iterator[str]

Iterate over CSV text results, from the resource's CSV endpoint.

Follows pagination to get a complete set of results, starting with the page specified in the *page* kwarg (or the first page, by default).

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Yields
text body of each response.

Raises
APIError – when a request fails

`iter_json_availability`(**params) → Iterator[dict]

Convenience method to query the JSON endpoint, using the availability expression. This may not be supported for all endpoints.

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Yields
dict with the "result" from the JSON body of each response.

Raises
NotImplementedError – if the resource doesn't support an availability query expression.

`iter_json_data`(**params) → Iterator[dict]

Iterate over results from the resource's JSON endpoint.

Follows pagination to get a complete set of results, starting with the page specified in the *page* kwarg (or the first page, by default).

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Yields
dict with the "result" from the JSON body of each response.

Raises
APIError – when a request fails

`points`(**params) → Iterator[dict]

Iterate over points from CSV response, yielding dictionaries.

This may involve multiple requests to the CSV endpoint, to follow pagination.

Parameters
****params** – query parameters for the request(s). These override self.defaults on a key-by-key basis.

Yields
dict – Keys are the headers from the CSV response. Values are converted to numeric types where applicable. If numpy is available, np.float64 is used.

Span

class runeq.stream.v1.**Span**(*cfg:* Config, ***defaults*)

Query spans.

property expr_availability

Availability expression for this resource.

Raises
NotImplementedError – if the resource doesn't support an availability query expression.

get_json_response(***params*) → Response

Make a GET request to the resource's JSON endpoint.

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis, for this request.

Returns
requests.Response

iter_json_availability(***params*) → Iterator[dict]

Convenience method to query the JSON endpoint, using the availability expression. This may not be supported for all endpoints.

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Yields
dict with the "result" from the JSON body of each response.

Raises
NotImplementedError – if the resource doesn't support an availability query expression.

iter_json_data(***params*) → Iterator[dict]

Iterate over results from the resource's JSON endpoint.

Follows pagination to get a complete set of results.

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Yields
dict with the "result" from the JSON body of each response.

Raises
APIError – when a request fails

State

class runeq.stream.v1.State(*cfg:* Config, ***defaults*)

Query device state.

property expr_availability

Availability expression for this resource.

Raises
NotImplementedError – if the resource doesn't support an availability query expression.

get_csv_response(***params*) → Response

Make a GET request to the resource's CSV endpoint.

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Returns
requests.Response

get_json_response(***params*) → Response

Make a GET request to the resource's JSON endpoint.

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis, for this request.

Returns
requests.Response

iter_csv_availability(***params*) → Iterator[dict]

Convenience method to query the CSV endpoint, using the availability expression. May not be supported for all resources.

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Yields
text body of each response.

Raises
NotImplementedError – if the resource doesn't support an availability query expression.

iter_csv_text(***params*) → Iterator[str]

Iterate over CSV text results, from the resource's CSV endpoint.

Follows pagination to get a complete set of results, starting with the page specified in the *page* kwarg (or the first page, by default).

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Yields
text body of each response.

Raises
APIError – when a request fails

iter_json_availability(***params*) → Iterator[dict]

Convenience method to query the JSON endpoint, using the availability expression. This may not be supported for all endpoints.

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Yields
dict with the "result" from the JSON body of each response.

Raises
NotImplementedError – if the resource doesn't support an availability query expression.

iter_json_data(***params*) → Iterator[dict]

Iterate over results from the resource's JSON endpoint.

Follows pagination to get a complete set of results, starting with the page specified in the *page* kwarg (or the first page, by default).

Parameters
****params** – Query parameters for the request. These override self.defaults, on a key-by-key basis.

Yields
dict with the "result" from the JSON body of each response.

Raises
APIError – when a request fails

points(***params*) → Iterator[dict]

Iterate over points from CSV response, yielding dictionaries.

This may involve multiple requests to the CSV endpoint, to follow pagination.

Parameters
****params** – query parameters for the request(s). These override self.defaults on a key-by-key basis.

Yields
dict – Keys are the headers from the CSV response. Values are converted to numeric types where applicable. If numpy is available, np.float64 is used.

2.5 Errors

Error classes.

exception runeq.errors.**APIError**(*status_code*, *details*)

Rune API Error. Includes details about error type.

exception runeq.errors.**InitializationError**

Rune Initialization Error. Raised when a user has not initialized their API credentials.

exception runeq.errors.**RuneError**

Base class for Rune SDK errors.

CHAPTER

THREE

SOURCE

The source code is available on Github.

API documentation can be browsed at https://docs.runelabs.io.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

r

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V