viresclient Documentation

Ashley Smith

May 03, 2024

VIRTUAL RESEARCH ENVIRONMENTS

```
pip install viresclient
```

```
conda install -c conda-forge viresclient
```

viresclient is a Python package which connects to a VirES server, of which there are two: *VirES for Swarm* (https://vires.services) and *VirES for Aeolus* (https://aeolus.services), through the WPS interface. This package handles product requests and downloads, enabling easy access to data and models from ESA's Earth Explorer missions, Swarm and Aeolus. This service is provided for ESA by EOX. For enquiries about the service and problems with accessing your account, please email info@vires.services. For help with usage, please email ashley.smith@ed.ac.uk (for Swarm data) or raise an issue on GitHub.

For code recipes and more, see Swarm Notebooks & Aeolus Notebooks. To start experimenting right away, *viresclient* is installed on the "Virtual Research Environment" (VRE), which is a managed Jupyter-based system provided for ESA by EOX. The service is free and open to all, accessible through your VirES account - check the notebooks to read more and get started.

Data and models are processed on demand on the VirES server - a combination of measurements from any time interval can be accessed. These are the same data that can be accessed by the VirES GUI. *viresclient* handles the returned data to allow direct loading as a single pandas.DataFrame, or xarray.Dataset.

```python
from viresclient import SwarmRequest

# Set up connection with server
request = SwarmRequest()
# Set collection to use
# - See https://viresclient.readthedocs.io/en/latest/available_parameters.html
request.set_collection("SW_OPER_MAGA_LR_1B")
# Set mix of products to fetch:
#  measurements (variables from the given collection)
#  models (magnetic model predictions at spacecraft sampling points)
#  auxiliaries (variables available with any collection)
# Optionally set a sampling rate different from the original data
request.set_products(
    measurements=["F", "B_NEC"],
    models=["CHAOS-Core"],
    auxiliaries=["QDLat", "QDLon"],
    sampling_step="PT10S"
)
# Fetch data from a given time interval
# - Specify times as ISO-8601 strings or Python datetime
data = request.get_between(
    start_time="2014-01-01T00:00",
    end_time="2014-01-01T01:00"
)
# Load the data as an xarray.Dataset
ds = data.as_xarray()
```

```
<xarray.Dataset>
Dimensions:           (NEC: 3, Timestamp: 360)
Coordinates:
* Timestamp          (Timestamp) datetime64[ns] 2014-01-01 ... 2014-01-01T00:59:50
Dimensions without coordinates: NEC
Data variables:
  Spacecraft         (Timestamp) <U1 'A' 'A' 'A' 'A' 'A' ... 'A' 'A' 'A' 'A'
  Latitude           (Timestamp) float64 -1.229 -1.863 -2.496 ... 48.14 48.77
  Longitude          (Timestamp) float64 -14.12 -14.13 -14.15 ... 153.6 153.6
  Radius             (Timestamp) float64 6.878e+06 6.878e+06 ... 6.868e+06
  F                  (Timestamp) float64 2.287e+04 2.281e+04 ... 4.021e+04
  F_CHAOS-Core       (Timestamp) float64 2.287e+04 2.282e+04 ... 4.02e+04
  B_NEC              (Timestamp, NEC) float64 2.01e+04 -4.126e+03 ... 3.558e+04
  B_NEC_CHAOS-Core   (Timestamp, NEC) float64 2.011e+04 ... 3.557e+04
  QDLat              (Timestamp) float64 -11.99 -12.6 -13.2 ... 41.59 42.25
  QDLon              (Timestamp) float64 58.02 57.86 57.71 ... -135.9 -136.0
Attributes:
  Sources:          ['SW_OPER_MAGA_LR_1B_20140101T000000_20140101T235959_050...
  MagneticModels:   ["CHAOS-Core = 'CHAOS-Core'(max_degree=20,min_degree=1)"]
  RangeFilters:     []
```

CHAPTER

ONE

HOW TO ACKNOWLEDGE VIRES

You can reference *viresclient* directly using the DOI of our zenodo record. VirES uses data from a number of different sources so please also acknowledge these appropriately.

> "We use the Python package, viresclient [1], to access [...] from ESA's VirES for Swarm service [2]"
> [1] https://doi.org/10.5281/zenodo.2554162
> [2] https://vires.services

1.1 Installation and First Usage

Note: For VRE users (it's free! read more: Swarm, Aeolus), viresclient is already installed and configured so skip these steps

1.1.1 1. Installation

Python 3.6 is required. Testing is primarily on Linux, but macOS and Windows should also work. Available through both pip and conda (conda-forge).

pip

conda

mamba

```
pip install viresclient
```

```
conda install --channel conda-forge viresclient
```

```
mamba install --channel conda-forge viresclient
```

Recommended setup if starting without Python already

There are many ways to work with Python. We recommend using conda/mamba to manage your programming environment because of the availability of many data science packages through conda.

conda

mamba

1. Install Miniconda: https://docs.conda.io/en/latest/miniconda.html
2. Set the conda-forge channel as the priority to install packages from:

```
conda config --add channels conda-forge
conda config --set channel_priority strict
```

You should do this to avoid mixing packages from the anaconda channel (which can result in broken environments), and try to get all packages from conda-forge where available for consistency.

3. Create a new conda environment with some recommended packages, including viresclient:

```
conda create --name myenv python=3.10 jupyterlab scipy matplotlib pandas xarray␣
↪cartopy h5py netCDF4 pytables ipywidgets viresclient
```

4. Activate the new environment (you do this each time you want to use it):

```
conda activate myenv
```

Mamba is a drop-in replacement for conda. You can install it into an existing (base) conda environment (`conda install -c conda-forge mamba`) and then just use `mamba` in place of `conda` in any commands - mamba is significantly faster. You can also install *mambaforge* directly to get mamba and conda-forge immediately configured in the base environment.

1. Download and install the mambaforge installer or check the mamba documentation
2. Create a new environment for your development work:

```
mamba create --name myenv python=3.10 jupyterlab scipy matplotlib pandas xarray␣
↪cartopy h5py netCDF4 pytables ipywidgets viresclient
```

3. Activate it to use it:

```
mamba activate myenv
```

1.1.2 2. First usage / Configuration

Swarm

Aeolus

Note: *For Jupyter notebook users*, just try:

```
from viresclient import SwarmRequest
request = SwarmRequest()
```

and you will automatically be prompted to set a token.

A first usage guide is provided as a Jupyter notebook (view). To run the notebook on your computer running Jupyter locally, right click here to download, or use git to get the whole example repository:

```
git clone https://github.com/Swarm-DISC/Swarm_notebooks.git
```

Access to the service is through the same user account as on the web interface (https://vires.services/) and is enabled through an access token (essentially a password). To get a token, log in to the website and click on your name on the top right to access the settings (or follow this link). From here, click on "Manage access tokens" and follow the instructions to create a new token.

To set your token in the client, use either the Python interface:

```
from viresclient import set_token
set_token("https://vires.services/ows")
# (you will now be prompted to enter the token)
```

or the command line tool:

```
$ viresclient set_token https://vires.services/ows
Enter access token: r-8-mlkP_RBx4mDv0di5Bzt3UZ52NGg-

$ viresclient set_default_server https://vires.services/ows
```

See also: see *Configuration Details* and *Access Token Management*

Note: *For Jupyter notebook users*, just try:

```
from viresclient import AeolusRequest
request = AeolusRequest()
```

and you will automatically be prompted to set a token.

A first usage guide is provided as a Jupyter notebook (view). To run the notebook on your computer running Jupyter locally, right click here to download, or use git to get the whole example repository:

```
git clone https://github.com/ESA-VirES/Aeolus-notebooks.git
```

Access to the service is through the same user account as on the web interface (https://aeolus.services/) and is enabled through an access token (essentially a password). To get a token, log in to the website and click on your name on the top right to access the settings (or follow this link). From here, click on "Manage access tokens" and follow the instructions to create a new token.

To set your token in the client, use either the Python interface:

```
from viresclient import set_token
set_token("https://aeolus.services/ows")
# (you will now be prompted to enter the token)
```

or the command line tool:

```
$ viresclient set_token https://aeolus.services/ows
Enter access token: r-8-mlkP_RBx4mDv0di5Bzt3UZ52NGg-
```

(continues on next page)

(continued from previous page)

```
$ viresclient set_default_server https://aeolus.services/ows
```

See also: see *Configuration Details* and *Access Token Management*

1.1.3 3. Example use

Note: A brief introduction is given here. For more possibilities, see *Introduction to notebooks*, and *VirES capabilities*.

Swarm

Aeolus

See also Swarm access through VirES

Choose which collection to access (see *Available parameters for Swarm* for more options):

```
import datetime as dt
from viresclient import SwarmRequest

request = SwarmRequest()
request.set_collection("SW_OPER_MAGA_LR_1B")
```

Next, use `.set_products()` to choose a combination of variables to retrieve, specified by keywords.

- `measurements` are measured by the satellite and members of the specified `collection`
- `models` are evaluated on the server at the positions of the satellite
- `auxiliaries` are additional parameters not unique to the `collection`
- if `residuals` is set to `True` then only data-model residuals are returned
- optionally use `sampling_step` to specify a resampling of the original time series (an ISO-8601 duration).

```
request.set_products(
  measurements=["F", "B_NEC"],
  models=["MCO_SHA_2C", "MMA_SHA_2C-Primary", "MMA_SHA_2C-Secondary"],
  auxiliaries=["QDLat", "QDLon", "MLT", "OrbitNumber", "SunZenithAngle"],
  residuals=False,
  sampling_step="PT10S"
)
```

Set a parameter range filter to apply. You can add multiple filters in sequence.

```
request.set_range_filter(parameter="Latitude", minimum=0, maximum=90)
request.set_range_filter("Longitude", 0, 90)
```

Specify the time range from which to retrieve data, make the request to the server:

```
data = request.get_between(
  start_time=dt.datetime(2016,1,1),
  end_time=dt.datetime(2016,1,2)
)
```

Transfer your data to a pandas.DataFrame, or a xarray.Dataset, or just save it as is:

```
df = data.as_dataframe()
ds = data.as_xarray()
data.to_file('outfile.cdf', overwrite=False)
```

The returned data has columns for:

- `Spacecraft, Timestamp, Latitude, Longitude, Radius`
- those specified by `measurements` and `auxiliaries`

... and model values and residuals, named as:

- `F_<model_id>` – scalar field
- `B_NEC_<model_id>` – vector field
- `F_res_<model_id>` – scalar field residual (`F - F_<model_id>`)
- `B_NEC_res_<model_id>` – vector field residual (`B_NEC - B_NEC_<model_id>`)

See Aeolus access through VirES

1.2 Configuration Details

Attention: Be careful not to accidentally add your credentials to online repositories or containers. You can use the CLI command `viresclient clear_credentials` to remove them from your environment. By default, this is just a file located at `~/.viresclient.ini`, where ~ is your home directory which is dependent on the operating system.

Tip: To get started quickly in Jupyter notebooks:

```
from viresclient import SwarmRequest
r = SwarmRequest("https://vires.services/ows")
```

OR:

```
from viresclient import AeolusRequest
r = AeolusRequest("https://aeolus.services/ows")
```

... then follow automatic instructions to configure token if not already set

Note: URL's on this page assume using *VirES for Swarm*. If using *Aeolus* instead, replace {`https://vires.services/ows` and `SwarmRequest`} with {`https://aeolus.services/ows` and `AeolusRequest`}

While it is possible to enter the server URL and access credentials (see *Access Token Management*) each time a new request object is created,

```
from viresclient import SwarmRequest
```

(continues on next page)

(continued from previous page)

```
# both URL and access token passed as request object's parameters
request = SwarmRequest(
    url="https://vires.services/ows",
    token="r-8-mlkP_RBx4mDv0di5Bzt3UZ52NGg-"
)
```

it is more convenient to omit them from the code and store them in a private configuration file. This configuration can be done using the *viresclient.set_token()* convenience function, the underlying *viresclient.ClientConfig()* module, or the command line interface (CLI) - see below. These will all set the configuration options in a file which is by default located at `~/.viresclient.ini` which can be edited directly, containing for example:

```
[https://vires.services/ows]
token = r-8-mlkP_RBx4mDv0di5Bzt3UZ52NGg-

[default]
url = https://vires.services/ows
```

When creating the configuration file manually make sure the file is readable by its owner only:

```
$ chmod 0600 ~/.viresclient.ini
$ ls -l ~/.viresclient.ini
-rw-------  1 owner owner  361 May 12 09:12 /home/owner/.viresclient.ini
```

When the configuration file is present, then the url and token options can be omitted from requests:

```
# access token read from configuration
request = SwarmRequest(url="https://vires.services/ows")

# both default URL and access token read from configuration
request = SwarmRequest()
```

The following sections describe how to set the configuration.

1.2.1 Configuration via CLI

The `viresclient` shell command can be used to set the server access configuration:

```
$ viresclient set_token https://vires.services/ows
Enter access token: r-8-mlkP_RBx4mDv0di5Bzt3UZ52NGg-

$ viresclient set_default_server https://vires.services/ows
```

Clear the configuration from the default location with:

```
$ viresclient clear_credentials
```

See also: *Command Line Interface (CLI)*

1.2.2 Configuration via Python

Use the following code to store the token in the `viresclient` configuration:

```
from viresclient import ClientConfig

cc = ClientConfig()
cc.set_site_config("https://vires.services/ows", token="r-8-mlkP_RBx4mDv0di5Bzt3UZ52NGg-
↪")
cc.default_url = "https://vires.services/ows"
cc.save()
```

Alternatively, use the convenience function:

```
from viresclient import set_token
set_token("https://vires.services/ows")
# (you will now be prompted to enter the token)
```

which calls the same code as above, but makes sure the token remains hidden so that it can't accidentally be shared.

1.2.3 For developers & DISC users

The accounts for the staging server (`staging.vires.services`), and DISC server (`staging.viresdisc.vires.services`) are separate. Tokens can be similarly generated on these and stored in the same configuration file alongside the others:

```
$ viresclient set_token https://staging.vires.services/ows
Enter access token: r-8-mlkP_RBx4mDv0di5Bzt3UZ52NGg-

$ viresclient set_token https://staging.viresdisc.vires.services/ows
Enter access token: VymMHhWjZ-9nSVs-FuPC27ca8C6cOyij
```

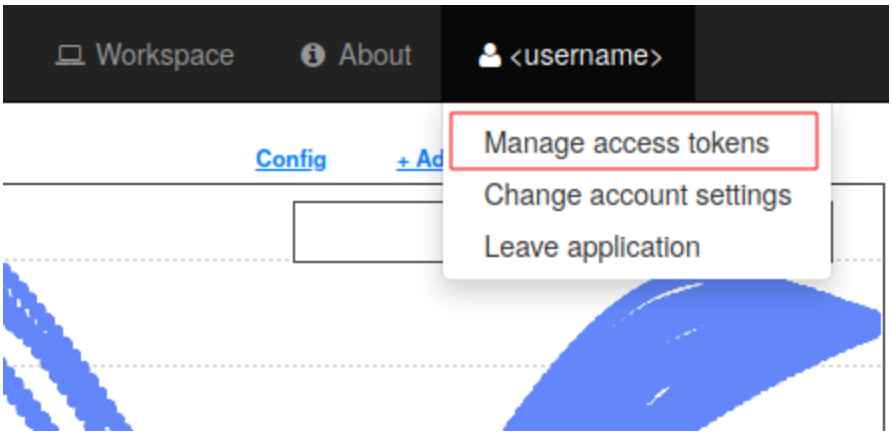
Using `SwarmRequest()` without the `url` parameter will use the default URL set above. To access a non-default server the URL parameter must be used:

```
from viresclient import SwarmRequest

# request using the default server (https://vires.services/ows)
request = SwarmRequest()

# request to an alternative, non-default server
request = SwarmRequest(url="https://staging.viresdisc.vires.services/ows")
```

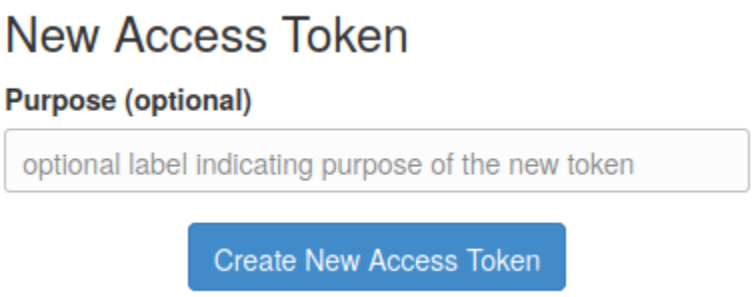
1.3 Access Token Management

An access token is required to access the VirES server. The tokens are managed via the web user interface.

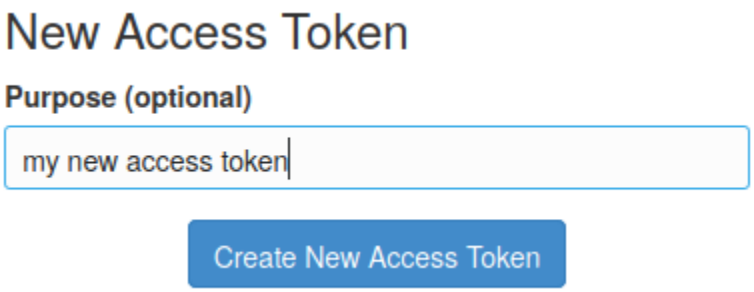
Assuming you have an existing VirES account and you can access the VirES web client (https://vires.services or https://aeolus.services), you get to token manager via *Manage access tokens* item in your account menu (displaying your username)



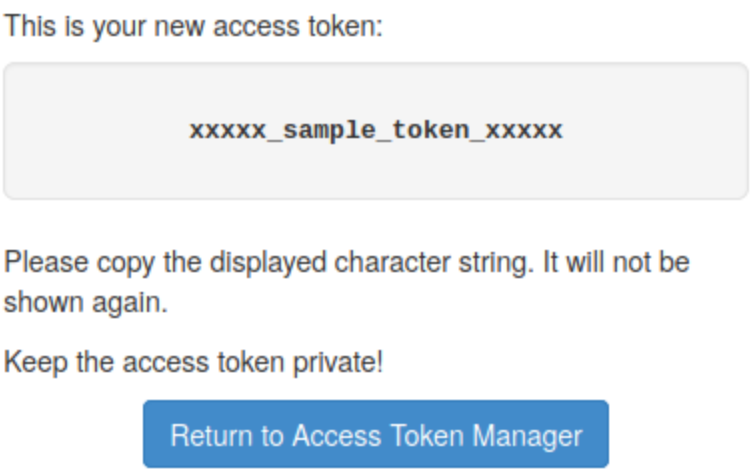
The token manager user interface allows creation of new tokens by pressing the *Create New Access Token* button



To distinguish different tokens, it is recommended to tag a new token by a brief label indicating its purpose

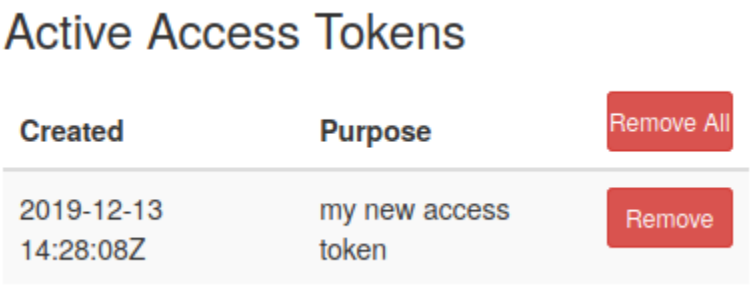


Once the *Create New Access Token* button has been pressed, a new token is generated and displayed to the user



As this point, you should copy the token to its destination.

The token manager list the active access tokens and allows their revocation



Note: *The tokens are secret.* Therefore, do not keep thier copies or share them with others. When a token is needed generate new one. When a token is no longer needed revoke it.

1.4 VirES capabilities

VirES provides more than just *access* to data. Some operations can be peformed on the data in-situ on the server side before being delivered to you.

Swarm

Aeolus

Data subsetting/filtering

Select data satisfying given conditions (e.g. searching a geographical range; selecting by quality flags)
viresclient.SwarmRequest.set_range_filter()
viresclient.SwarmRequest.set_choice_filter()
viresclient.SwarmRequest.set_bitmask_filter()
viresclient.SwarmRequest.add_filter() (for arbitrary filters)

Data resampling

Time series can be resampled to a given cadence

See *sampling_step* option in *viresclient.SwarmRequest.set_products()*

Querying information about data

For example:
viresclient.SwarmRequest.available_times()
viresclient.SwarmRequest.get_orbit_number()
viresclient.SwarmRequest.get_times_for_orbits()

Geomagnetic model evaluation

Forwards evaluation of magnetic field models when a magnetic dataset is selected (e.g. `MAGx_LR`). For more detail, see *Geomagnetic model handling*.
viresclient.SwarmRequest.available_models()
viresclient.SwarmRequest.get_model_info()
models option in *viresclient.SwarmRequest.set_products()*

Identifying conjunctions between spacecraft

viresclient.SwarmRequest.get_conjunctions()

Synchronous and asynchronous processing When using *viresclient.SwarmRequest.get_between()* with small requests, change the default of *asynchronous=True* to *asynchronous=False* to process faster (no progress bar). By default, jobs are processed asynchronously (i.e. entered into a queue) which is appropriate for longer requests. You can only have two asynchronous jobs running at one time.

Data subsetting/filtering

Select data satisfying given conditions (e.g. searching a geographical range; selecting by quality flags)
viresclient.AeolusRequest.set_range_filter()
viresclient.AeolusRequest.set_bbox()

Querying information about data

For example:
viresclient.AeolusRequest.available_times()

Synchronous and asynchronous processing When using *viresclient.AeolusRequest.get_between()* with small requests, change the default of *asynchronous=True* to *asynchronous=False* to process faster (no progress bar). By default, jobs are processed asynchronously (i.e. entered into a queue) which is appropriate for longer requests. You can only have two asynchronous jobs running at one time.

Uploading data

Data of certain formats can be uploaded to the server and then manipulated like existing datasets (available privately within your account)
See *Command Line Interface (CLI)* and *viresclient.DataUpload()*

1.5 Available parameters for Swarm

Tip: Did you know? The *VirES for Swarm* service provides data not only from Swarm but also INTERMAGNET ground observatories (search below for `AUX_OBS`), and recalibrated platform magnetometer data from selected LEO missions (search below for `MAG_`).

Note:

See also: Jupyter notebook about data and model availability - check out the other demo notebooks there too.

You can check which parameters are available with:

```
from viresclient import SwarmRequest
request = SwarmRequest()
request.available_collections()
request.available_measurements("MAG")
request.available_measurements("SW_OPER_MAGA_LR_1B")
request.available_models()
request.available_auxiliaries()
```

The available measurements are segregated according to the "collection" (essentially Swarm products): each `collection` has a number of `measurements` associated with it, and the appropriate collection must be set in order to access the measurements. `auxiliaries` are available together with any set `collection`. `models` provide magnetic model evaluation on demand, at the locations of the time series which is being accessed (when accessing magnetic field data such as `MAG` or `MAG_HR`). Standard positional variables always returned, such as Timestamp, Spacecraft, geocentric Latitude, Longitude, Radius.

See the Swarm Data Handbook for details about the products and Swarm Product Demos (Jupyter notebooks) for basic recipes to get started.

1.5.1 collections

Note: `FAST` data are available for some products. These are processed and made available faster than the traditional operational (`OPER`) data, mainly for space weather monitoring. The collection names are the same, but with `OPER` replaced by `FAST`:

- `SW_FAST_MAGx_LR_1B`
- `SW_FAST_MAGx_HR_1B`
- `SW_FAST_EFIx_LP_1B`
- `SW_FAST_MODx_SC_1B`

Collections are grouped according to a type containing similar measurements (i.e. the same product from different spacecraft). The collection type can be given to *viresclient.SwarmRequest.available_collections()* to retrieve the full collection names. These cover the Swarm data products as below (replace x with A, B, or C for Alpha, Bravo, or Charlie):

Collection full name	Collection type	Description
SW_OPER_MAGx_LR_1B	MAG	Magnetic field (1Hz) from VFM and ASM
SW_OPER_MAGx_HR_1B	MAG_HR	Magnetic field (50Hz) from VFM
SW_OPER_EFIx_LP_1B	EFI	Electric field instrument (Langmuir probe measurements at 2Hz)
SW_OPER_EFIxTIE_2_	EFI_TIE	Estimates of the ion temperatures
SW_EXPT_EFIx_TCT02	EFI_TCT02	2Hz cross-track ion flows
SW_EXPT_EFIx_TCT16	EFI_TCT16	16Hz cross-track ion flows
SW_PREL_EFIxIDM_2_	EFI_IDM	2Hz ion drift velocities and effective masses (SLIDEM project)
SW_OPER_IPDxIRR_2F	IPD	Ionospheric plasma characteristics (derived quantities at 1Hz)
SW_OPER_TECxTMS_2F	TEC	Total electron content
SW_OPER_FACxTMS_2F	FAC	Field-aligned currents (single satellite)
SW_OPER_FAC_TMS_2F	FAC	Field-aligned currents (dual-satellite A-C)
SW_OPER_EEFxTMS_2F	EEF	Equatorial electric field
SW_OPER_IBIxTMS_2F	IBI	Ionospheric bubble index
SW_OPER_MODx_SC_1B	MOD_SC	Spacecraft positions at 1Hz

The AEBS (auroral electrojets and boundaries) products are a bit more complicated:

Collection full name	Collection type	Description
SW_OPER_AEJxLPL_2F	AEJ_LPL	Auroral electrojets line profile - Line current method (LC)
SW_OPER_AEJxLPL_2F:Quality	AEJ_LPL:Quality	-> Quality indicators per orbital section from LC
SW_OPER_AEJxPBL_2F	AEJ_PBL	-> Peaks and boundaries from LC
SW_OPER_AEJxLPS_2F	AEJ_LPS	Auroral electrojets line profile - SECS method
SW_OPER_AEJxLPS_2F:Quality	AEJ_LPS:Quality	-> Quality indicators per orbital section from SECS
SW_OPER_AEJxPBS_2F	AEJ_PBS	-> Peaks and boundaries from SECS
SW_OPER_AEJxPBS_2F:GroundMagneticDisturbance	AEJ_PBS:GroundMagneticDisturbance	-> Location and strength of peak ground disturbance per pass
SW_OPER_AOBxFAC_2F	AOB_FAC	Auroral oval boundaries derived from FACs

The PRISM (Plasmapause Related boundaries in the topside Ionosphere as derived from Swarm Measurements) products are provided as:

Collection full name	Collection type	Description
SW_OPER_MITx_LP_2F	MIT_LP	Minima of the Midlatitude Ionospheric Trough (MIT) - derived from Langmuir Probe (LP) measurements
SW_OPER_MITx_LP_2F:ID	MIT_LP:ID	-> Boundaries of the MIT - derived from the LP
SW_OPER_MITxTEC_2F	MIT_TEC	Minima of the MIT - derived from Total Electron Content (TEC)
SW_OPER_MITxTEC_2F:ID	MIT_TEC:ID	-> Boundaries of the MIT - derived from TEC
SW_OPER_PPIxFAC_2F	PPI_FAC	Midnight Plasmapause Index (PPI)
SW_OPER_PPIxFAC_2F:ID	PPI_FAC:ID	-> Boundaries of the Small-Scale Field Aligned Currents (SSFAC)

The AUX_OBS collections contain ground magnetic observatory data from INTERMAGNET and WDC. Please note that these data are provided under different usage terms than the ESA data, and must be acknowledged accordingly.

Collection full name	Collection type	Description
SW_OPER_AUX_OBSH2_	AUX_OBSH	Hourly values derived from both WDC and INTERMAGNET data
SW_OPER_AUX_OBSM2_	AUX_OBSM	Minute values from INTERMAGNET
SW_OPER_AUX_OBSS2_	AUX_OBSS	Second values from INTERMAGNET

The AUX_OBS collections contain data from all observatories together (distinguishable by the `IAGA_code` variable). Data from a single observatory can be accessed with special collection names like `SW_OPER_AUX_OBSM2_:ABK` where `ABK` can be replaced with the IAGA code of the observatory. Use *viresclient.SwarmRequest.available_observatories()* to find these IAGA codes.

The VOBS collections contain derived magnetic measurements from Geomagnetic Virtual Observatories and have a similar interface as the AUX_OBS collections. The data are organised across several collections:

Collection full name	Collection type	Description
SW_OPER_VOBS_1M_2_	VOBS_SW_1M	Swarm (1-monthly cadence)
OR_OPER_VOBS_1M_2_	VOBS_OR_1M	Ørsted (1-monthly cadence)
CH_OPER_VOBS_1M_2_	VOBS_CH_1M	CHAMP (1-monthly)
CR_OPER_VOBS_1M_2_	VOBS_CR_1M	Cryosat-2 (1-monthly)
CO_OPER_VOBS_1M_2_	VOBS_CO_1M	Composite time series from Ørsted, CHAMP, Cryosat-2, & Swarm (1-monthly)
SW_OPER_VOBS_4M_2_	VOBS_SW_4M	Swarm (4-monthly)
OR_OPER_VOBS_4M_2_	VOBS_OR_4M	Ørsted (4-monthly)
CH_OPER_VOBS_4M_2_	VOBS_CH_4M	CHAMP (4-monthly)
CR_OPER_VOBS_4M_2_	VOBS_CR_4M	Cryosat-2 (4-monthly)
CO_OPER_VOBS_4M_2_	VOBS_CO_4M	Composite time series from Ørsted, CHAMP, Cryosat-2, & Swarm (4-monthly)
SW_OPER_VOBS_1M_2_:SecularVariation	VOBS_SW_1M:SecularVariation	Secular variation (B_SV) from Swarm 1-monthly
(ditto for the others)		

Each VOBS product (e.g. Swarm 1-monthly) is split into two collections (e.g. `SW_OPER_VOBS_1M_2_` (containing `B_OB` & `B_CF`) and `SW_OPER_VOBS_1M_2_:SecularVariation` (containing `B_SV`)) because of the different temporal sampling points (i.e. differing `Timestamp`) of these measurements. Data can also be requested for a specific virtual observatory alone (distinguishable by the `SiteCode` variable) with special collection names like `SW_OPER_VOBS_1M_2_:N65W051` and `SW_OPER_VOBS_1M_2_:SecularVariation:N65W051`.

Calibrated magnetic data are also available from external missions: Cryosat-2, GRACE (A+B), GRACE-FO (1+2), GOCE:

Collection full name	Collection type	Available measurement names
CS_OPER_MAG	MAG_CS	F,B_NEC,B_mod_NEC,B_NEC1,B_NEC2,B_NEC3,B_FGM1,B_FGM2, B_FGM3,q_NEC_CRF,q_error
GRACE_x_MAG (x=A/B)	MAG_GRACE	F,B_NEC,B_NEC_raw,B_FGM,B_mod_NEC,q_NEC_CRF,q_error
GFx_OPER_FGM_ACAL_CORR (x=1/2)	MAG_GRACE	F,B_NEC,B_FGM,dB_MTQ_FGM,dB_XI_FGM,dB_NY_FGM,dB_BT_FGM, dB_ST_FGM,dB_SA_FGM,dB_BAT_FGM,q_NEC_FGM,B_FLAG
GFx_MAG_ACAL_CORR_HR (x=1/2)	MAG_GRACE	F,B_MAG,B_NEC,q_NEC_FGM,B_FLAG,KP_DST_FLAG,Latitude_QD, Longitude_QD
GO_MAG_ACAL_CORR	MAG_GOCE	F,B_MAG,B_NEC,dB_MTQ_SC,dB_XI_SC,dB_NY_SC,dB_BT_SC, dB_ST_SC,dB_SA_SC,dB_BAT_SC,dB_HK_SC,dB_BLOCK_CORR, q_SC_NEC,q_MAG_SC,B_FLAG
GO_MAG_ACAL_CORR_HR	MAG_GOCE_HR	F,B_MAG,B_NEC,q_FGM_NEC,B_FLAG,MAGNETIC_ACTIVITY_FLAG, NaN_FLAG,Latitude_QD,Longitude_QD

The `measurements`, `models`, and `auxiliaries` chosen will match the cadence of the `collection` chosen.

1.5.2 measurements

Choose combinations of measurements from one of the following sets, corresponding to the collection chosen above. The collection full name or collection type can be given to *viresclient.SwarmRequest.available_measurements()* to retrieve the list of available measurements for a given collection (e.g. `request.available_measurements("SW_OPER_MAGA_LR_1B")`)

Collection type	Available measurement names
MAG	F,dF_Sun,dF_AOCS,dF_other,F_error,B_VFM,B_NEC,dB_Sun,dB_AOCS,dB_other,B_error,q_NEC_CRF,Att_error,Flags_F,Flags_B,Flags_q,Flags_Platform,ASM_Freq_Dev
MAG_HR	F,B_VFM,B_NEC,dB_Sun,dB_AOCS,dB_other,B_error,q_NEC_CRF,Att_error,Flags_B,Flags_q,Flags_Platform,ASM_Freq_Dev
EFI	U_orbit,Ne,Ne_error,Te,Te_error,Vs,Vs_error,Flags_LP,Flags_Ne,Flags_Te,Flags_Vs
EFI_TIE	Latitude_GD,Longitude_GD,Height_GD,Radius_GC,Latitude_QD,MLT_QD,Tn_msis,Te_adj_LP,Ti_meas_drift,Ti_model_drift,Flag_ti_meas,Flag_ti_model
EFI_TCT	VsatC,VsatE,VsatN,Bx,By,Bz,Ehx,Ehy,Ehz,Evx,Evy,Evz,Vicrx,Vicry,Vicrz,Vixv,Vixh,Viy,Viz,Vixv_error,Vixh_error,Viy_error,Viz_error,Latitude_QD,MLT_QD,Calibration_flags,Quality_flags
EFI_IDM	Latitude_GD,Longitude_GD,Height_GD,Radius_GC,Latitude_QD,MLT_QD,V_sat_nec,M_i_eff,M_i_eff_err,M_i_eff_Flags,M_i_eff_tbt_model,V_i,V_i_err,V_i_Flags,V_i_raw,N_i,N_i_err,N_i_Flags,A_fp,R_p,T_e,Phi_sc
IPD	Ne,Te,Background_Ne,Foreground_Ne,PCP_flag,Grad_Ne_at_100km,Grad_Ne_at_50km,Grad_Ne_at_20km,Grad_Ne_at_PCP_edge,ROD,RODI10s,RODI20s,delta_Ne10s,delta_Ne20s,delta_Ne40s,Num_GPS_satellites,mVTEC,mROT,mROTI10s,mROTI20s,IBI_flag,Ionosphere_region_flag,IPIR_index,Ne_quality_flag,TEC_STD
TEC	GPS_Position,LEO_Position,PRN,L1,L2,P1,P2,S1,S2,Elevation_Angle,Absolute_VTEC,Absolute_STEC,Relative_STEC,Relative_STEC_RMS,DCB,DCB_Error
FAC	IRC,IRC_Error,FAC,FAC_Error,Flags,Flags_F,Flags_B,Flags_q
EEF	EEF,EEJ,RelErr,Flags
IBI	Bubble_Index,Bubble_Probability,Flags_Bubble,Flags_F,Flags_B,Flags_q

AEBS products:

Collection type	Available measurement names
AEJ_LPL	Latitude_QD,Longitude_QD,MLT_QD,J_NE,J_QD
AEJ_LPL:Quality	RMS_misfit,Confidence
AEJ_PBL	Latitude_QD,Longitude_QD,MLT_QD,J_QD,Flags,PointType
AEJ_LPS	Latitude_QD,Longitude_QD,MLT_QD,J_CF_NE,J_DF_NE,J_CF_SemiQD,J_DF_SemiQD,J_R
AEJ_LPS:Quality	RMS_misfit,Confidence
AEJ_PBS	Latitude_QD,Longitude_QD,MLT_QD,J_DF_SemiQD,Flags,PointType
AEJ_PBS:GroundMagneticDisturbance	B_NE
AOB_FAC	Latitude_QD,Longitude_QD,MLT_QD,Boundary_Flag,Quality,Pair_Indicator

PRISM products:

Collection type	Available measurement names
MIT_LP	`Counter,Latitude_QD,Longitude_QD,MLT_QD,L_value,SZA,Ne,Te,Depth,DR,Width,dL,PW_Gradient,EW_Gradient,Quality`
MIT_LP:ID	`Counter,Latitude_QD,Longitude_QD,MLT_QD,L_value,SZA,Ne,Te,Position_Quality,PointType`
MIT_TEC	`Counter,Latitude_QD,Longitude_QD,MLT_QD,L_value,SZA,TEC,Depth,DR,Width,dL,PW_Gradient,EW_Gradient,Quality`
MIT_TEC:ID	`Counter,Latitude_QD,Longitude_QD,MLT_QD,L_value,SZA,TEC,Position_Quality,PointType`
PPI_FAC	`Counter,Latitude_QD,Longitude_QD,MLT_QD,L_value,SZA,Sigma,PPI,dL,Quality`
PPI_FAC:ID	`Counter,Latitude_QD,Longitude_QD,MLT_QD,L_value,SZA,Position_Quality,PointType`

AUX_OBS products:

Collection type	Available measurement names
AUX_OBSH	`B_NEC,F,IAGA_code,Quality,ObsIndex`
AUX_OBSM	`B_NEC,F,IAGA_code,Quality`
AUX_OBSS	`B_NEC,F,IAGA_code,Quality`

AUX_OBSH contains a special variable, `ObsIndex`, which is set to 0, 1, 2 … to indicate changes to the observatory where the IAGA code has remained the same (e.g. small change of location, change of instrument or calibration procedure).

VOBS products:

Collection full name	Available measurement names
SW_OPER_VOBS_1M_2_	`SiteCode,B_CF,B_OB,sigma_CF,sigma_OB`
SW_OPER_VOBS_1M_2_:SecularVariation	`SiteCode,B_SV,sigma_SV`
(ditto for the others)	

1.5.3 `models`

Models are evaluated along the satellite track at the positions of the time series that has been requested. These must be used together with one of the MAG collections, and one or both of the "F" and "B_NEC" measurements. This can yield either the model values together with the measurements, or the data-model residuals.

Note: For a good estimate of the ionospheric field measured by a Swarm satellite (with the core, crust and magnetosphere effects removed) use a composed model defined as: `models=["'CHAOS-full' = 'CHAOS-Core' + 'CHAOS-Static' + 'CHAOS-MMA-Primary' + 'CHAOS-MMA-Secondary'"]` (click for more info)

This composed model can also be accessed by an alias: `models=["CHAOS"]` which represents the full CHAOS model

See Magnetic Earth for an introduction to geomagnetic models.

```
IGRF,

# Comprehensive inversion (CI) models:
MCO_SHA_2C,                                 # Core
MLI_SHA_2C,                                 # Lithosphere
MMA_SHA_2C-Primary, MMA_SHA_2C-Secondary,   # Magnetosphere
MIO_SHA_2C-Primary, MIO_SHA_2C-Secondary,   # Ionosphere

# Dedicated inversion models:
MCO_SHA_2D,                                 # Core
MLI_SHA_2D, MLI_SHA_2E                      # Lithosphere
MIO_SHA_2D-Primary, MIO_SHA_2D-Secondary    # Ionosphere
AMPS                                        # High-latitude ionosphere

# Fast-track models:
MMA_SHA_2F-Primary, MMA_SHA_2F-Secondary,   # Magnetosphere

# CHAOS models:
CHAOS-Core,                                 # Core
CHAOS-Static,                               # Lithosphere
CHAOS-MMA-Primary, CHAOS-MMA-Secondary      # Magnetosphere

# Other lithospheric models:
MF7, LCS-1

# Aliases for compositions of the above models (shortcuts)
MCO_SHA_2X     # 'CHAOS-Core'
CHAOS-MMA      # 'CHAOS-MMA-Primary' + 'CHAOS-MMA-Secondary'
CHAOS          # 'CHAOS-Core' + 'CHAOS-Static' + 'CHAOS-MMA-Primary' + 'CHAOS-MMA-Secondary'
MMA_SHA_2F     # 'MMA_SHA_2F-Primary' + 'MMA_SHA_2F-Secondary'
MMA_SHA_2C     # 'MMA_SHA_2C-Primary' + 'MMA_SHA_2C-Secondary'
MIO_SHA_2C     # 'MIO_SHA_2C-Primary' + 'MIO_SHA_2C-Secondary'
MIO_SHA_2D     # 'MIO_SHA_2D-Primary' + 'MIO_SHA_2D-Secondary'
SwarmCI        # 'MCO_SHA_2C' + 'MLI_SHA_2C' + 'MIO_SHA_2C-Primary' + 'MIO_SHA_2C-Secondary' +
↪'MMA_SHA_2C-Primary' + 'MMA_SHA_2C-Secondary'
```

Custom (user uploaded) models can be provided as a .shc file and become accessible in the same way as pre-defined models, under the name `"Custom_Model"`.

Flexible evaluation of models and defining new derived models is possible with the "model expressions" functionality whereby models can be defined like:

```
request.set_products(
  ...
  models=["Combined_model = 'MMA_SHA_2F-Primary'(min_degree=1,max_degree=1) + 'MMA_SHA_
↪2F-Secondary'(min_degree=1,max_degree=1)"],
  ...
)
```

In this case, model evaluations will then be available in the returned data under the name "Combined_model", but you can name it however you like.

NB: When using model names containing a hyphen (`-`) then extra single (`'`) or double (`"`) quotes must be used around the model name. This is to distinguish from arithmetic minus (`-`).

1.5.4 auxiliaries

```
SyncStatus, Kp10, Kp, Dst, dDst, IMF_BY_GSM, IMF_BZ_GSM, IMF_V, F107, F10_INDEX,
OrbitDirection, QDOrbitDirection,
OrbitSource, OrbitNumber, AscendingNodeTime,
AscendingNodeLongitude, QDLat, QDLon, QDBasis, MLT, SunDeclination,
SunHourAngle, SunRightAscension, SunAzimuthAngle, SunZenithAngle,
SunLongitude, SunVector, DipoleAxisVector, NGPLatitude, NGPLongitude,
DipoleTiltAngle, F107_avg81d, F107_avg81d_count
```

Note:

- `Kp` provides the Kp values in fractional form (e.g 2.2), and `Kp10` is multiplied by 10 (as integers)
- `F107` is the hourly 10.7 cm solar radio flux value, and `F10_INDEX` is the daily average
- `QDLat` and `QDLon` are quasi-dipole coordinates
- `MLT` is calculated from the QDLon and the subsolar position
- `OrbitDirection` and `QDOrbitDirection` flags indicate if the satellite is moving towards or away from each pole, respectively for geographic and quasi-dipole magnetic poles. +1 for ascending, and -1 for descending (in latitude); 0 for no data.

Note: Check other packages such as hapiclient and others from PyHC for data from other sources.

1.6 Available parameters for Aeolus

Note: Details to follow. See https://notebooks.aeolus.services for examples.

1.7 Geomagnetic model handling

1.7.1 Model evaluation

The geomagnetic models provided by VirES are all based on spherical harmonics, though they differ in their parameterisation and time-dependence. They provide predictions of the different geomagnetic field sources (e.g. core, lithosphere, ionosphere, magnetosphere) and are generally valid near to Earth's surface (i.e. at ground level and in LEO). To select appropriate models for your use case, you should refer to the scientific publications related to these models.

In VirES, we provide model evaluations calculated at the same sample points as the data products. This means that the spherical harmonic expansion is made at the time and location of each datapoint (e.g. in the case of `MAGx_LR`, at every second). The main purpose is to provide the data-model residuals, or magnetic perturbations, useful for studying the external magnetic field, typically ionospheric in origin.

Pending: In a future VirES release, some interpolation will be used to provide the magnetic model predictions along the 50Hz `MAGx_HR` products (to improve speed). The predictions at the 1Hz (`MAGx_LR`) locations will be used and a cubic interpolation performed to provide the predictions at the 50Hz locations.

The magnetic data products provide the magnetic field vector as the parameter `B_NEC` in the NEC (North, East, Centre) frame, as well as the magnetic field intensity/magnitude (scalar), `F`. When also requesting models from VirES (by supplying the `models` kwarg to *viresclient.SwarmRequest.set_products()*), the corresponding model predictions will be returned in parameters named `B_NEC_<model-name>` or `F_<model-name>`. Alternatively, the data-model residual alone, named `B_NEC_res_<model-name>` or `F_res_<model-name>` can be returned directly by also supplying the kwarg `residuals=True`. Models should be provided as a list, like `models=["CHAOS", "IGRF"]`.

1.7.2 Available models

See *Available parameters for Swarm / models* for the list of available models.

You can use *viresclient.SwarmRequest.available_models()* and *viresclient.SwarmRequest.get_model_info()* to query the details of models.

1.7.3 Composed and custom models

When providing `models` to *viresclient.SwarmRequest.set_products()*, they can be customised:

Rename:

`models=["Model='CHAOS-Core'"]`

This will provide the `CHAOS-Core` model renamed to `Model`, so that the returned parameters will include `B_NEC_Model` instead of `B_NEC_CHAOS-Core`.

Compose (combine):

`models=["Model='CHAOS-Core' + 'CHAOS-Static'"]`

This sums together the contribution from `CHAOS-Core` and `CHAOS-Static` into a custom model called `Model`.

Customise:

```
models=["Model='CHAOS-Core'(max_degree=20) +
'CHAOS-Static'(min_degree=21,max_degree=80)"]
```

This limits the spherical harmonic degree used in the model calculation.

Note that single and double quotes are interchangeable, and must be used sometimes in order to enclose a model name and thus distinguish usage of a hyphen (`-`) in the model name from an arithmetic minus.

For more examples, see https://notebooks.vires.services/notebooks/02b__viresclient-available-data#manipulation-of-models

You can query information about your selected models using *viresclient.SwarmRequest.get_model_info()*:

```
from viresclient import SwarmRequest

request = SwarmRequest()

request.get_model_info(
    models=["Model='CHAOS-Core'(max_degree=20) + 'CHAOS-Static'(min_degree=21,max_
↪degree=80)"]
)
```

```
{'Model': {'expression': "'CHAOS-Core'(max_degree=20,min_degree=1) + 'CHAOS-Static'(max_
↪degree=80,min_degree=21)",
  'validity': {'start': '1997-02-07T05:23:17.067838Z',
   'end': '2024-03-01T02:57:24.851521Z'},
  'sources': ['CHAOS-7_static.shc',
   'SW_OPER_MCO_SHA_2X_19970101T000000_20230807T235959_0715',
   'SW_OPER_MCO_SHA_2X_20230808T000000_20240229T235959_0715']}}
```

1.7.4 Model caching

To speed up usage of commonly used expensive models, the server stores and uses a cache of some of the model values (so that they do not always need to be evaluated from scratch). This should happen transparently so you generally do not need to worry about it, but it may be helpful to understand when the cache might *not* be used, causing data requests to take longer.

Note: The caching mechanism can be bypassed (forcing direct evaluation of models) by supplying `ignore_cached_models=True` in *viresclient.SwarmRequest.set_products()*

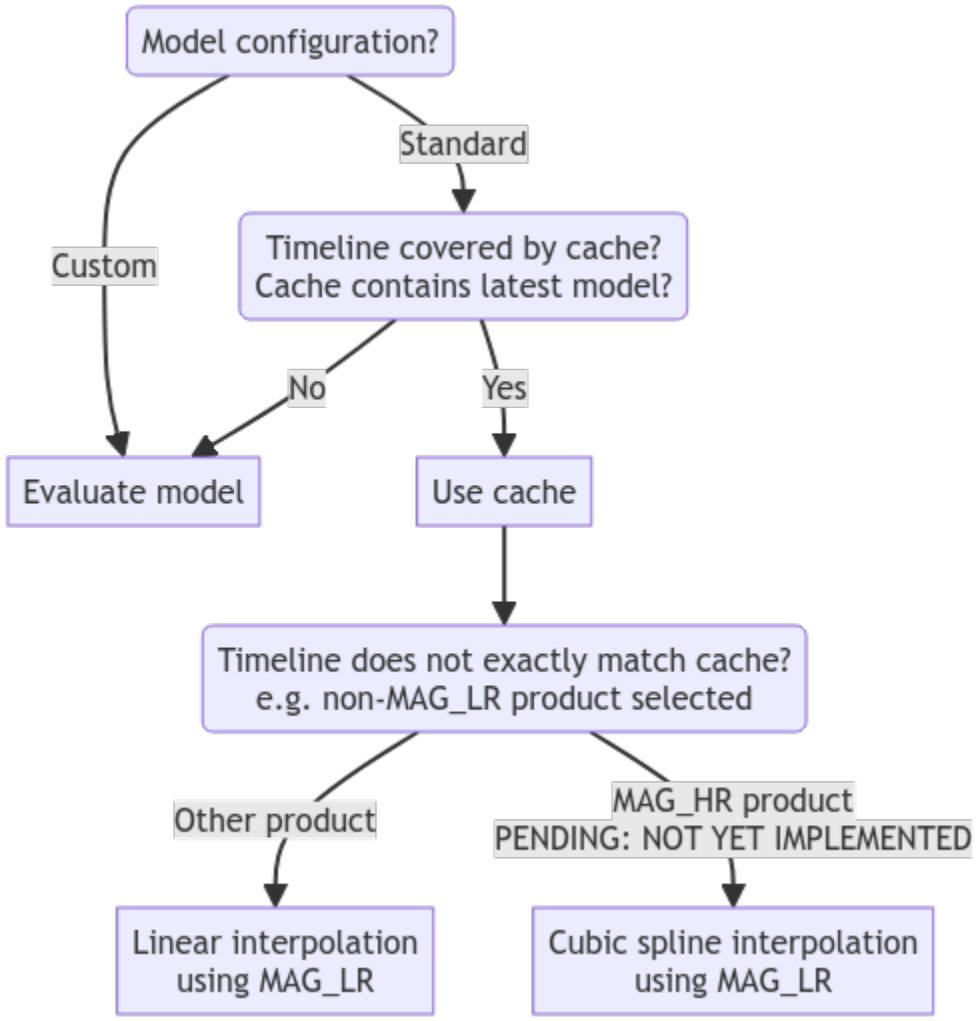
Cached models (these are chosen as they are both expensive and commonly used):

```
CHAOS-Static
MIO_SHA_2C-Primary
MIO_SHA_2C-Secondary
MLI_SHA_2C
```

The predictions for these models are cached only at the positions and times defined by the following products (i.e. low resolution magnetic products):

```
SW_OPER_MAGx_LR_1B (x=A,B,C)
GRACE_x_MAG (x=A,B)
GFx_OPER_FGM_ACAL_CORR (x=1,2)
GO_MAG_ACAL_CORR
GO_MAG_ACAL_CORR_ML
CS_OPER_MAG
```

The logic describing when the cache is used is as follows:



Custom configured models, e.g. `CHAOS-Static(max_degree=80)`, are not cached and must be evaluated directly.

Composed models, i.e. `Model = Model1 + Model2`, will use the cache for sub-models where available. For example, choosing `CHAOS-Core + CHAOS-Static` will make use of the cache for `CHAOS-Static` (an expensive model), but will directly evaluate `CHAOS-Core` (a cheap model), and combine the result. The same is true for *alias* models such as `CHAOS` (which equates to `CHAOS-Core + CHAOS-Static + CHAOS-MMA`).

When the source products or model are updated, the cache needs to be re-generated accordingly. This means means there is some delay before the cache is available again (while the changes are still being processed). In cases where the cache has been obsoleted, the system falls back to evaluating the model directly. In short, the caching mechanism prefers model consistency over performance.

1.7.5 Model values through HAPI

What is HAPI? See https://notebooks.vires.services/notebooks/02h1_hapi

When accessing magnetic datasets, there are additional HAPI parameters available:

```
B_NEC_Model
F_Model
B_NEC_res_Model
F_res_Model
```

These give, respectively, vector and scalar magnetic model values and data-model residuals using the full CHAOS model (core + lithosphere + magnetosphere). These are provided through the cache as described above.

1.8 Introduction to notebooks

Jupyter notebooks are a convenient tool for interactive data exploration, rapid prototyping, and producing reports. The Virtual Research Environment (VRE) provides free JupyterLab instances with persistent storage where you can run notebooks.

The VRE is a whole analysis suite, including many other libraries from the Python ecosystem. We provide recipes in the form of noteboooks compiled into a "Jupyter Book" to show how viresclient can be blended together with other libraries to access, visualise, and analyse data.

Table 1: Notebook repositories

Name / GitHub	View	Interact
Swarm-DISC/Swarm_notebooks		
ESA-VirES/Aeolus-notebooks		

Note: The *nbgitpuller* links above perform *git* operations for you, applying updates when you re-click the link using special automatic merging behaviour. Sometimes it may be necessary to perform the git operations directly instead.

To clear any changes you made and fetch the latest version, from within `Swarm_notebooks` or `Aeolus-notebooks` run from a terminal:

```
git fetch
git reset --hard origin/master
```

1.9 Release notes

1.9.1 Change log

Changes from 0.11.6 to 0.11.7

- Added auxiliaries `["F107_avg81d", "F107_avg81d_count"]`
- Updated and added ML-calibrated variants of GOCE & GRACE-FO magnetic datasets, `GO_MAG_ACAL_CORR_ML` & `GFx_MAG_ACAL_CORR_ML`

Changes from 0.11.5 to 0.11.6

- Fix to allow multiple VOBS to be fetched in one request

Changes from 0.11.4 to 0.11.5

- Fix *viresclient.SwarmRequest.available_times()* usage with pandas 2.x

Changes from 0.11.3 to 0.11.4

- Support for (L1B) FAST data

Changes from 0.11.2 to 0.11.3

- Added `ignore_cached_models=True` in *viresclient.SwarmRequest.set_products()*
- Added description of model handling in docs

Changes from 0.11.1 to 0.11.2

- Fix support for cdflib version 1.0

Changes from 0.11.0 to 0.11.1

- `viresclient` package now available through conda-forge
- Added parameter to Swarm `MAG` collections: `dF_Sun`
- Added GOCE ML magnetic dataset: `GO_MAG_ACAL_CORR_ML`

Changes from 0.10.3 to 0.11.0

- **Breaking change:**
  - *viresclient.ReturnedData()* property `.range_filters` changed to `.data_filters`
  - Xarray datasets attributes (`.attrs` property) have `"RangeFilters"` changed to `"AppliedFilters"`
- Added new arbitrary data filter functionality, with new methods:
  - *viresclient.SwarmRequest.set_range_filter()*
  - *viresclient.SwarmRequest.set_choice_filter()*
  - *viresclient.SwarmRequest.set_bitmask_filter()*
  - *viresclient.SwarmRequest.add_filter()*
- Added new collections for Swarm:
  - `SW_PREL_EFIxIDM_2_` (type `EFI_IDM`: ion drift velocities & effective masses, SLIDEM project)
  - `GO_MAG_ACAL_CORR` (type `MAG_GOCE`: magnetic data from the GOCE mission)
- Added new collections for Aeolus:
  - `ALD_U_N_1A`

- Fixed bug in merging multi-file datasets when loading as xarray

Changes from 0.10.2 to 0.10.3

- Added new collections:
  - `SW_OPER_EFIxTIE_2_` (type `EFI_TIE`: ion temperatrues)
  - `SW_EXPT_EFIx_TCT02` & `SW_EXPT_EFIx_TCT16` (types `EFI_TCT02`, `EFI_TCT16`: cross-track ion flows)

Changes from 0.10.1 to 0.10.2

- Removed upper version limits for dependencies

Changes from 0.10.0 to 0.10.1

- Update Jinja2 dependency

Changes from 0.9.1 to 0.10.0

- Added functionality to support VirES for Aeolus. See https://notebooks.aeolus.services
- Added dependency: netCDF4

Changes from 0.9.0 to 0.9.1

- Added *viresclient.SwarmRequest.get_conjunctions()* to fetch Swarm A/B conjunctions
- Fixed compatibility with xarray v0.19 of `reshape` kwarg in *viresclient.ReturnedData.as_xarray()*

Changes from 0.8.0 to 0.9.0

- Added support for:
  - PRISM products (`SW_OPER_MITx_LP_2F`, `SW_OPER_MITxTEC_2F`, `SW_OPER_PPIxFAC_2F`)
  - Multi-mission magnetic products (`CS_OPER_MAG`, `GRACE_x_MAG`, `GFx_OPER_FGM_ACAL_CORR`)
  - Swarm spacecraft positions (`SW_OPER_MODx_SC_1B`)
- Fixed missing auxiliary "dDst"
- Fixed fetching longer time series of hourly observatory products
- Added new progress bar that tracks processing of chunks in long requests

Changes from 0.7.2 to 0.8.0

- Added support for:
  - VOBS products (Virtual Observatories), e.g. collection `SW_OPER_VOBS_1M_2_`
  - AUX_OBSH products (hourly ground observatory data)
- Added *viresclient.SwarmRequest.available_times()* to query temporal availability of any collection
- Added new `reshape=True` kwarg to *viresclient.ReturnedData.as_xarray()* to enable optional reshaping of xarray datasets loaded from VOBS and AUX_OBS collections to higher-dimensional objects containing a new dimension (`IAGA_code` for AUX_OBS and `SiteCode` for VOBS)
- Added command line tool, `viresclient clear_credentials`, to help delete the stored credentials
- Changed tqdm progress bars to use `tqdm.notebook` when in Jupyter notebook (otherwise still uses plain tqdm)
- Dropped `"Timestamp"` variable attribute `"units"` (i.e. `ds["Timestamp"].attrs["units"]`) when loading as `xarray.Dataset`, for compatibility with xarray 0.17 when saving as netcdf

Changes from 0.7.1 to 0.7.2

- Fix usage of cdflib v0.3.20

Changes from 0.7.0 to 0.7.1

- Fix use of `expand` in `.as_dataframe()` for `AUX_OBS`

Changes from 0.6.2 to 0.7.0

- Added support for:
  - AUX_OBS products
  - AEBS products
  - MLI_SHA_2E
- See *Available parameters for Swarm* for details of the collection and measurement names
- Added *viresclient.SwarmRequest.available_observatories()* to query the AUX_OBS collections to identify IAGA codes available within each collection

Changes from 0.6.1 to 0.6.2

- Added automatic initialisation of access token when running on VRE
- Added new composed model aliases (shortcuts)

Changes from 0.6.0 to 0.6.1

- Fix to support the new EEFxTMS_2F baseline 02:
  - Product now available for Swarm Charlie (`C`)
  - `EEF` unit changed from `V/m` to `mV/m`
  - New measurement, `EEJ`
  - Variable renamed: `flag` to `Flag`

Changes from 0.5.0 to 0.6.0

- Provides access to `MAGx_HR` collections (50Hz magnetic measurements)
- Allows pandas v1.0+
- Dataframe index name is now set to "Timestamp" (fixes regression in a previous version)

Changes from 0.4.3 to 0.5.0

- IGRF model series have changed name: `IGRF-12` is dropped in favour of `IGRF` which now provides the latest IGRF (currently IGRF-13)
- `request.available_collections("MAG")` can now be called to filter by collection groups, *and now returns a dict instead of a list*
- Improvements for `xarray.Dataset` support:
  - NEC now provided as named coordinates for `B_NEC`-type variables
  - Similarly (VFM, quaternion, WGS84) coordinates also provided for the variables ["B_VFM", "dB_Sun", "dB_AOCS", "dB_other", "B_error"], ["q_NEC_CRF"], ["GPS_Position", "LEO_Position"] respectively
  - Metadata (units and description) are now set for each variable
  - (With xarray 0.14+, try `xarray.set_options(display_style="html")` for nicer output)

Changes from 0.4.2 to 0.4.3

- AMPS is now accessible as a regular model on the DISC server, see:

```
request = SwarmRequest("https://staging.viresdisc.vires.services/ows")
request.get_model_info(["AMPS"])
```

- xarray.Dataset objects now contain dimension names for all variables. Variables containing `B_NEC` get the `NEC` dimension name.
- CHAOS model series have changed name: `CHAOS-6-Core` etc. is dropped for `CHAOS-Core` etc. which provides the latest version of the CHAOS models (currently CHAOS-7)
- Better error message when authentication with server fails.
- When in notebooks: Detect empty or invalid credentials (e.g. on first usage), direct user to the token generation page, and prompt for token input.
- Added `request.list_jobs()` to give info on previous two jobs on the server (failed/running/succeeded).

Changes from 0.4.1 to 0.4.2

- Fixed orbit number queries (get_orbit_number)
- Added model sources to model info

Changes from 0.4.0 to 0.4.1

- Added low level data upload API and CLI
- Added set_token convenience function for quick configuration
- Changed list of accessible models:
  - Removed `MCO_SHA_2F`, `SIFM`
  - Added `MF7`, `LCS-1`

Changes from 0.3.0 to 0.4.0

- Fixed issues with running on Windows
- Enforcing Python v3.5+ for installation
- Allowing higher versions of cdflib, pandas, and xarray
- Added CLI configuration for setting server address and token
- Metadata for source lineage is now easier to access (names of original ESA data files, details of models used, and filters applied). These are set as properties of *viresclient.ReturnedData()* (i.e. `data`) and as metadata (`.attrs`) in the `Dataset` returned from `.as_xarray()`:

```
data.sources
data.magnetic_models
data.range_filters

ds = data.as_xarray()
ds.Sources
ds.MagneticModels
ds.RangeFilters
```

- Added access to collections `SW_OPER_IPDxIRR_2F`
- Added auxiliary data `F107` which is the hourly F10.7 value. This is in addition to `F10_INDEX` which was already present, which is a daily average.
- Added possibility of accessing multiple collections simultaneously, e.g.:

```
request.set_collection("SW_OPER_MAGA_LR_1B", "SW_OPER_MAGC_LR_1B")
```

- Added optional "expansion" of dataframes with:

```
data.as_dataframe(expand=True)
```

  This expands columns which contain vectors (e.g. `B_NEC`) into separate columns named like: `B_NEC_N`, `B_NEC_E`, `B_NEC_C`. This is recommended so that numpy operations will work on the columns. The default is `expand=False` to preserve the older behaviour.

Changes from v0.2.6 to 0.3.0 ^^^^^^^^^^^^^^^^^^^^^^^^^^^^_

- Service officially open to public through self-registration on https://vires.services
- Token-based authentication added

Changes from v0.2.5 to 0.2.6 ^^^^^^^^^^^^^^^^^^^^^^^^^^^-

- New model composition behaviour is implemented, extending what is possible with the `models` kwarg in *viresclient.SwarmRequest.set_products()* (with backwards compatibility). See demo in https://github.com/smithara/viresclient_examples/blob/master/model_residuals_and_cartopy.ipynb
- New method *viresclient.SwarmRequest.get_model_info()* to fetch model details from server.
- *viresclient.SwarmRequest.available_models()* is updated with these details.
- New parameters in TEC collections: `Elevation_Angle`, `Absolute_VTEC`.
- New parameters in auxiliaries: `OrbitDirection`, `QDOrbitDirection`.
- The auxiliary `Kp` is now provided as the proper Kp value, and `Kp10` is provided with the old behaviour with the value of Kp*10.
- Updated dependency on cdflib to v0.3.9, and xarray to allow both v0.10.x and v0.11.x.

Changes from v0.2.4 to 0.2.5 ^^^^^^^^^^^^^^^^^^^^^^^^^^^-

- EFI collections have changed from `SW_OPER_EFIx_PL_1B` to `SW_OPER_EFIx_LP_1B`, with different measurement variables
- Added support for user-defined models by providing a .shc file path as the `custom_model` in *viresclient.SwarmRequest.set_products()*. Model evaluations and residuals will then be returned, named as "Custom_Model", in the same way as other models behave.
- Added alternative input start and end times as ISO-8601 strings to *viresclient.SwarmRequest.get_between()*
- Minor bug fixes

Changes from v0.2.1 to v0.2.4 ^^^^^^^^^^^^^^^^^^^^^^^^^^^_

- Added models CHAOS-6-MMA-Primary and CHAOS-6-MMA-Secondary

Changes from v0.2.0 to v0.2.1 ^^^^^^^^^^^^^^^^^^^^^^^^^^^_

- Improved performance of pandas and xarray loading from cdf.
- Added `nrecords_limit` option to *viresclient.SwarmRequest.get_between()* to override the default maximum number of records in each request. Use this if a request is failing with a server error that the maximum allowable number of records has been exceeded - but this means that there is probably duplicate data on the server (old and new versions), so check the data that gets returned:

```
data = request.get_between(start_time, end_time, nrecords_limit=3456000)
ds = data.as_xarray()
# Identify negative time jumps
np.where(np.diff(ds["Timestamp"]).astype(float) < 0)
# e.g [2519945, 5284745, 5481414]
for i in [2519945, 5284745, 5481414]:
    print(ds.isel(Timestamp=i))
# Length of day should be 86400
ds.sel(Timestamp='2014-02-02')
```

- Added `tmpdir` option to *viresclient.SwarmRequest.get_between()* to override the default temporary file directory. The default is selected automatically according to https://docs.python.org/3/library/tempfile.html#tempfile.mkstemp (usually /tmp). This may not be suitable when fetching large amounts of data as some machines may have limited space available in /tmp or there may be a higher performance or preferred location.

Changes from v0.1.0 to v0.2.0 ^^^^^^^^^^^^^^^^^^^^^^^^^^_

- Now use `SwarmRequest` instead of `ClientRequest`.
- kwarg `subsample` changed to `sampling_step`.
- Added references to .available_collections() and .available_models().
- User credentials are automatically stored in a configuration file `~/.viresclient.ini`.
- Downloads are streamed to temporary files instead of being held in memory.
- Any size request is now supported. Large requests are automatically chunked up.
- Added download progress bar indicating size in MB.
- xarray added as a dependency and `.as_xarray()` method added.

1.10 API Reference

1.10.1 SwarmRequest

class viresclient.**SwarmRequest**(*url=None*, *token=None*, *config=None*, *logging_level='NO_LOGGING'*)

Bases: `viresclient._client.ClientRequest`

Handles the requests to and downloads from the server.

Examples

Retrieve data:

```
from viresclient import SwarmRequest
# Set up connection with server
request = SwarmRequest("https://vires.services/ows")
# Set collection to use
request.set_collection("SW_OPER_MAGA_LR_1B")
# Set mix of products to fetch:
#  measurements (variables from the given collection)
#  models (magnetic model predictions at spacecraft sampling points)
#  auxiliaries (variables available with any collection)
request.set_products(
    measurements=["F", "B_NEC"],
    models=["CHAOS-Core"],
    auxiliaries=["QDLat", "QDLon"],
    sampling_step="PT10S"
)
# Fetch data from a given time interval
data = request.get_between(
    start_time="2014-01-01T00:00",
    end_time="2014-01-01T01:00"
)
# Load the data as an xarray.Dataset
ds = data.as_xarray()
```

Check what data are available:

```
request.available_collections(details=False)
request.available_measurements("MAG")
request.available_auxiliaries()
request.available_models(details=False)
```

Parameters

- **url** (*str*) –
- **token** (*str*) –
- **config** (*str or ClientConfig*) –
- **logging_level** (*str*) –

add_filter(*filter_*)

Add an arbitrary data filter.

Parameters **filter** (*str*) – string defining the filter, as shown below

Filter grammar:

```
filter: predicate
predicate:
     variable == literal |
     variable != literal |
     variable < number |
     variable > number |
     variable <= number |
     variable >= number |
     variable & unsigned-integer == unsigned-integer |
     variable & unsigned-integer != unsigned-integer |
     (predicate AND predicate [AND predicate ...]) |
     (predicate OR predicate [OR predicate ...]) |
     NOT predicate
literal: boolean | integer | float | string
number: integer | float
variable: identifier | identifier[index]
index: integer[, integer ...]

Both single- and double quoted strings are allowed.
NaN values are matched by the ==/!= operators, i.e., the predicates
are internally converted to a proper "IS NaN" or "IS NOT NaN"
comparison.
```

Examples

“Flags & 128 == 0” Match records with Flag bit 7 set to 0.

“Elevation >= 15” Match values with values greater than or equal to 15.

“(Label == “D” OR Label == “N” OR Label = “X”)” Match records with Label set to D, N or X.

“(Type != 1 AND Type != 34) NOT (Type == 1 OR Type == 34)” Exclude records with Type set to 1 or 34.

“(Vector[2] <= -0.1 OR Vector[2] >= 0.5)” Match records with Vector[2] values outside of the (-0.1, 0.5) range.

`applied_filters`()

Print currently applied filters.

`available_auxiliaries`()

Returns a list of the available auxiliary parameters.

`available_collections`(*groupname=None*, *details=True*)

Show details of available collections.

Parameters

- **groupname** (*str*) – one of: (“MAG”, “EFI”, etc.)
- **details** (*bool*) – If True then print a nice output. If False then return a dict of available collections.

`available_measurements`(*collection=None*)

Returns a list of the available measurements for the chosen collection.

Parameters **collection** (*str*) – one of: (“MAG”, “EFI”, “IBI”, “TEC”, “FAC”, “EEF”)

`available_models`(*param=None*, *details=True*, *nice_output=True*)

Show details of avalable models.

If details is True, return a dictionary of model names and details. If nice_output is True, the dictionary is printed nicely. If details is False, return a list of model names. If param is set, filter to only return entries including this

---

Note:

F = Fast-Track Products
C = Comprehensive Inversion
D = Dedicated Chain
MCO = Core / main
MLI = Lithosphere
MMA = Magnetosphere
MIO = Ionosphere

Parameters

- **param** (*str*) – one of “F C D MCO MLI MMA MIO”
- **details** (*bool*) – True for a dict of details, False for a brief list
- **nice_output** (*bool*) – If True, just print the dict nicely

available_observatories(*collection*, *start_time=None*, *end_time=None*, *details=False*, *verbose=True*)

Get list of available observatories from server.

Search availability by collection, one of:

```
"SW_OPER_AUX_OBSH2_"
"SW_OPER_AUX_OBSM2_"
"SW_OPER_AUX_OBSS2_"
```

Examples

```
from viresclient import SwarmRequest
request = SwarmRequest()
# For a list of observatories available:
request.available_observatories("SW_OPER_AUX_OBSM2_")
# For a DataFrame also containing availability start and end times:
request.available_observatories("SW_OPER_AUX_OBSM2_", details=True)
# For available observatories during a given time period:
request.available_observatories(
    "SW_OPER_AUX_OBSM2_", "2013-01-01", "2013-02-01"
)
```

Parameters

- **collection** (*str*) – OBS collection name, e.g. "SW_OPER_AUX_OBSM2_"
- **start_time** (*datetime / ISO_8601 string*) –
- **end_time** (*datetime / ISO_8601 string*) –
- **details** (*bool*) – returns DataFrame if True
- **verbose** (*bool*) – Notify with special data terms

Returns IAGA codes (and start/end times)

Return type list or DataFrame

available_times(*collection*, *start_time=None*, *end_time=None*)

Returns temporal availability for a given collection

Parameters

- **(str)** – collection name
- **start_time** (*datetime / ISO_8601 string*) –
- **end_time** (*datetime / ISO_8601 string*) –

Returns DataFrame

clear_filters()

Remove all applied filters.

clear_range_filter()

Remove all applied filters.

get_between(*start_time=None*, *end_time=None*, *filetype='cdf'*, *asynchronous=True*, *show_progress=True*, *show_progress_chunks=True*, *leave_intermediate_progress_bars=True*, *nrecords_limit=None*, *tmpdir=None*)

Make the server request and download the data.

Parameters

- **start_time** (*datetime / ISO_8601 string*) –
- **end_time** (*datetime / ISO_8601 string*) –
- **filetype** (*str*) – one of ('csv', 'cdf')
- **asynchronous** (*bool*) – True for asynchronous processing, False for synchronous
- **show_progress** (*bool*) – Set to False to remove progress bars
- **show_progress_chunks** (*bool*) – Set to False to remove progress bar for chunks
- **leave_intermediate_progress_bars** (*bool*) – Set to False to clean up the individual progress bars left when making chunked requests
- **nrecords_limit** (*int*) – Override the default limit per request (e.g. nrecords_limit=3456000)
- **tmpdir** (*str*) – Override the default temporary file directory

Return type *ReturnedData*

get_conjunctions(*start_time=None*, *end_time=None*, *threshold=1.0*, *spacecraft1='A'*, *spacecraft2='B'*, *mission1='Swarm'*, *mission2='Swarm'*, *grade='OPER'*)

Get times of the spacecraft conjunctions.

Currently available for the following spacecraft pairs:

- Swarm-A/Swarm-B

Parameters

- **start_time** (*datetime / ISO_8601 string*) – optional start time
- **end_time** (*datetime / ISO_8601 string*) – optional end time
- **threshold** (*float*) – optional maximum allowed angular separation in degrees; by default set to 1; allowed values are [0, 180]
- **spacecraft1** – identifier of the first spacecraft, default to 'A'
- **spacecraft2** – identifier of the second spacecraft, default to 'B'
- **mission1** (*str*) – mission of the first spacecraft, defaults to 'Swarm'
- **mission2** (*str*) – mission of the first spacecraft, defaults to 'Swarm'
- **grade** (*str*) – products grade, possible values "OPER" or "FAST"

Return type *ReturnedData*

get_model_info(*models=None*, *custom_model=None*, *original_response=False*)

Get model info from server.

Handles the same models input as .set_products(), and returns a dict like:

{'IGRF12': { 'expression': 'IGRF12(max_degree=13,min_degree=0)', 'validity': {'start': '1900-01-01T00:00:00Z', 'end': '2020-01-01T00:00:00Z' }, ... }

If original_response=True, return the list of dicts like:

{'expression': 'MCO_SHA_2C(max_degree=16,min_degree=0)', 'name': 'MCO_SHA_2C', 'validity': {'start': '2013-11-30T14:38:24Z', 'end': '2018-01-01T00:00:00Z'}}, ...

Parameters

- **models** (*list/dict*) – as with set_products
- **custom_model** (*str*) – as with set_products
- **original_response** (*bool*) –

Returns dict or list

get_orbit_number(*spacecraft*, *input_time*, *mission='Swarm'*)

Translate a time to an orbit number.

Parameters

- **spacecraft** (*str*) – Swarm: one of ('A','B','C') or ("Alpha", "Bravo", "Charlie") GRACE: one of ('1','2') GRACE-FO: one of ('1','2') CryoSat-2: None
- **input_time** (*datetime*) – a point in time
- **mission** (*str*) – one of ('Swarm', 'GRACE', 'GRACE-FO', 'CryoSat-2')

Returns The current orbit number at the input_time

Return type int

get_times_for_orbits(*start_orbit*, *end_orbit*, *mission='Swarm'*, *spacecraft=None*)

Translate a pair of orbit numbers to a time interval.

Parameters

- **start_orbit** (*int*) – a starting orbit number
- **end_orbit** (*int*) – a later orbit number
- **spacecraft** (*str*) – Swarm: one of ('A','B','C') or ("Alpha", "Bravo", "Charlie") GRACE: one of ('1','2') GRACE-FO: one of ('1','2') CryoSat-2: None
- **mission** (*str*) – one of ('Swarm', 'GRACE', 'GRACE-FO', 'CryoSat-2')

Returns (start_time, end_time) The start time of the start_orbit and the ending time of the end_orbit. (Based on ascending nodes of the orbits)

Return type tuple (datetime)

list_jobs()

Return job information from the server.

Returns dict

set_bitmask_filter(*parameter*, *selection=0*, *mask=- 1*, *negate=False*)

Set a bitmask filter to apply.

Filters data for *parameter & mask == selection & mask*, or *parameter & mask != selection & mask* if negated.

Note: See *SwarmRequest.add_filter()* for arbitrary filters.

Parameters

- **parameter** (*str*) –
- **mask** (*integer*) –
- **selection** (*integer*) –

Examples

request.set_bitmask_filter("Flags_F", 0, 1) to set "Flags_F & 1 == 0" (i.e. bit 1 is set to 0)

set_choice_filter(*parameter*, **values*, *negate=False*)

Set a choice filter to apply.

Filters data for *parameter in values*, or *parameter not in values* if negated.

Note: See *SwarmRequest.add_filter()* for arbitrary filters.

Parameters

- **parameter** (*str*) –
- **values** (*float or integer or string*) –

Examples

request.set_choice_filter("Flags_F", 0, 1) to set "(Flags_F == 0 OR Flags_F == 1)"

request.set_choice_filter("Flags_F", 0, 1, negate=True) to set "(Flags_F != 0 AND Flags_F != 1)"

set_collection(**args*, *verbose=True*)

Set the collection(s) to use.

Parameters

- **(str)** – one or several from .available_collections()
- **verbose** (*bool*) – Notify if special data terms

set_products(*measurements=None*, *models=None*, *custom_model=None*, *auxiliaries=None*, *residuals=False*, *sampling_step=None*, *ignore_cached_models=False*)

Set the combination of products to retrieve.

If residuals=True then just get the measurement-model residuals, otherwise get both measurement and model values.

Parameters

- **measurements** (*list(str)*) – from .available_measurements(collection_key)
- **models** (*list(str)/dict*) – from .available_models() or defineable with custom expressions
- **custom_model** (*str*) – path to a custom model in .shc format
- **auxiliaries** (*list(str)*) – from .available_auxiliaries()
- **residuals** (*bool*) – True if only returning measurement-model residual

- **sampling_step** (*str*) – ISO_8601 duration, e.g. 10 seconds: PT10S, 1 minute: PT1M
- **ignore_cached_models** (*bool*) – True if cached models should be ignored and calculated on-the-fly

set_range_filter(*parameter*, *minimum=None*, *maximum=None*, *negate=False*)

Set a range filter to apply.

Filters data for minimum parameter maximum, or parameter < minimum OR parameter > maximum if negated.

Note:

- Apply multiple filters with successive calls to `.set_range_filter()`
- See *SwarmRequest.add_filter()* for arbitrary filters.

Parameters

- **parameter** (*str*) –
- **minimum** (*float or integer*) –
- **maximum** (*float or integer*) –

Examples

request.set_range_filter("Latitude", 0, 90) to set "Latitude >= 0 AND Latitude <= 90"

request.set_range_filter("Latitude", 0, 90, negate=True) to set "(Latitude < 0 OR Latitude > 90)"

1.10.2 AeolusRequest

class viresclient.**AeolusRequest**(*url=None*, *token=None*, *config=None*, *logging_level='NO_LOGGING'*)

Bases: viresclient._client.ClientRequest

Handles the requests to and downloads from the server.

Parameters

- **url** (*str*) –
- **username** (*str*) –
- **password** (*str*) –
- **token** (*str*) –
- **config** (*str or ClientConfig*) –
- **logging_level** (*str*) –

available_collections(*collection=None*, *field_type=None*, *like=None*, *details=True*)

available_times(*collection*, *start_time=None*, *end_time=None*)

Returns temporal availability for a given collection

Parameters

- **(str)** – collection name
- **start_time** (*datetime / ISO_8601 string*) –
- **end_time** (*datetime / ISO_8601 string*) –

Returns DataFrame

clear_range_filter()

Remove all applied filters.

get_between(*start_time=None*, *end_time=None*, *filetype='cdf'*, *asynchronous=True*, *show_progress=True*, *show_progress_chunks=True*, *leave_intermediate_progress_bars=True*, *nrecords_limit=None*, *tmpdir=None*)

Make the server request and download the data.

Parameters

- **start_time** (*datetime / ISO_8601 string*) –
- **end_time** (*datetime / ISO_8601 string*) –
- **filetype** (*str*) – one of ('csv', 'cdf')
- **asynchronous** (*bool*) – True for asynchronous processing, False for synchronous
- **show_progress** (*bool*) – Set to False to remove progress bars
- **show_progress_chunks** (*bool*) – Set to False to remove progress bar for chunks
- **leave_intermediate_progress_bars** (*bool*) – Set to False to clean up the individual progress bars left when making chunked requests
- **nrecords_limit** (*int*) – Override the default limit per request (e.g. nrecords_limit=3456000)
- **tmpdir** (*str*) – Override the default temporary file directory

Return type *ReturnedData*

get_from_file(*path=None*, *filetype='nc'*)

Get VirES ReturnedData object from file path

Allows loading of locally saved netCDF file (e.g. using to_file method) providing access to data manipulation methods such as as_xarray

Parameters

- **path** (*str*) –
- **filetype** (*str*) –

list_jobs()

Return job information from the server.

Returns dict

print_available_collections(*collection=None*, *field_type=None*, *regex=None*, *details=True*, *path=False*)

set_bbox(*bbox=None*)

Set a bounding box to apply as filter.

Note: Dictionary argument has to contain n, e, s, w keys for north, east, south and west values as EPSG 4326 coordinates

Parameters **bbox** (*dict*) –

set_collection(*collection*)

set_fields(*observation_fields=None*, *measurement_fields=None*, *ica_fields=None*, *sca_fields=None*, *mca_fields=None*, *mie_profile_fields=None*, *rayleigh_profile_fields=None*, *rayleigh_wind_fields=None*, *mie_wind_fields=None*, *rayleigh_grouping_fields=None*, *mie_grouping_fields=None*, *group_fields=None*, *fields=None*)

set_range_filter(*parameter=None*, *minimum=None*, *maximum=None*)

Set a filter to apply.

Filters data for minimum parameter maximum

Note: Apply multiple filters with successive calls to set_range_filter()

Parameters

- **parameter** (*str*) –
- **minimum** (*float*) –
- **maximum** (*float*) –

set_variables(*aux_type=None*, *fields=None*, *dsd_info=False*)

1.10.3 ReturnedData

class viresclient.**ReturnedData**(*filetype=None*, *N=1*, *tmpdir=None*)

Bases: object

Flexible object for working with data returned from the server

Holds a list of ReturnedDataFile objects under self.contents

Example usage:

```
...
data = request.get_between(..., ...)
data.sources
data.data_filters
data.magnetic_models
data.as_xarray()
data.as_xarray_dict()
data.as_dataframe(expand=True)
data.to_file()
```

as_dataframe(*expand=False*)

Convert the data to a pandas DataFrame.

If expand is True, expand some columns, e.g.:

B_NEC -> B_NEC_N, B_NEC_E, B_NEC_C

B_VFM -> B_VFM_i, B_VFM_j, B_VFM_k

Parameters **expand** (*bool*) –

Returns pandas.DataFrame

as_xarray(*reshape=False*)

Convert the data to an xarray Dataset.

Parameters **reshape** (*bool*) – Reshape to a convenient higher dimensional form

Returns xarray.Dataset

as_xarray_dict()

Convert the data to a dict containing an xarray per group.

Returns dict of xarray.Dataset

property contents

List of ReturnedDataFile objects

property data_filters

Get list of filters applied.

property filetype

Filetype string

property magnetic_models

Get list of magnetic models used.

property sources

Get list of source product identifiers.

to_file(*path*, *overwrite=False*)

Saves the data to the specified file, when data is only in one file.

Only write to file if it does not yet exist, or if overwrite=True. Currently handles CSV and CDF formats.

Note: This is currently only implemented for smaller data when the request has not been split into multiple requests - the limit is the equivalent of 50 days of 1Hz measurements. In these situations, you can still load the data as pandas/xarray objects (the contents of each file is automatically concatenated) and save them as a different file type. Or use `.to_files()` to save the split data directly.

Parameters

- **path** (*str*) – path to the file to save as
- **overwrite** (*bool*) – Will overwrite existing file if True

to_files(*paths*, *overwrite=False*)

Saves the data to the specified files.

Only write to file if it does not yet exist, or if overwrite=True. Currently handles CSV and CDF formats.

Parameters

- **paths** (*list of str*) – paths to the files to save as
- **overwrite** (*bool*) – Will overwrite existing file if True

class viresclient.**ReturnedDataFile**(*filetype=None*, *tmpdir=None*)

Bases: object

For handling individual files returned from the server.

Holds the data returned from the server and the data type. Data is held in a NamedTemporaryFile, which is automatically closed and destroyed when it goes out of scope. Provides output to different file types and data objects.

as_dataframe(*expand=False*)

Convert the data to a pandas DataFrame.

Returns pandas.DataFrame

as_xarray(*group=None*, *reshape=False*)

Convert the data to an xarray Dataset.

Note: Does not support csv

Only supports scalar and 3D vectors (currently)

Returns xarray.Dataset

as_xarray_dict()

Convert the data to an xarray Dataset.

Note: Only supports netCDF format

Returns dict of xarray.Dataset

property data_filters

property filetype

Filetype is one of (“csv”, “cdf”, “nc”)

property magnetic_models

open_cdf()

Returns the opened file as cdflib.CDF

property sources

to_file(*path*, *overwrite=False*)

Saves the data to the specified file.

Only write to file if it does not yet exist, or if overwrite=True. Currently handles CSV and CDF formats.

Parameters

- **path** (*str*) – path to the file to save as
- **overwrite** (*bool*) – Will overwrite existing file if True

to_netcdf(*path*, *overwrite=False*)

Saves the data as a netCDF4 file (this is compatible with HDF5)

Extension should be .nc

1.10.4 ClientConfig

class viresclient.**ClientConfig**(*path=None*)

Bases: object

Client configuration.

Example usage:

```
cc = ClientConfig()      # use default configuration file
cc = ClientConfig("./viresconf.ini")  # use custom configuration file

print(cc.path)           # print path
print(cc)                # print the whole configuration

cc.default_url = "https://foo.bar/ows"  # set default server

# access to credentials configuration ...
cc.set_site_config("https://foo2.bar/ows", token="...")

cc.save()    # save configuration
```

property default_url

Get default URL or None if not set.

get_site_config(*url*)

Get configuration for the given URL.

init(*env_var_name='VIRES_ACCESS_CONFIG'*)

Initialize client configuration.

property path

Get path of the configuration file.

save()

Save the configuration file.

set_site_config(*url*, ***options*)

Set configuration for the given URL.

1.10.5 set_token

viresclient.**set_token**(*url='https://vires.services/ows'*, *token=None*, *set_default=False*)

Set the access token for a given URL, using user input.

Get an access token at https://vires.services/accounts/tokens/

See https://viresclient.readthedocs.io/en/latest/config_details.html

This will create a configuration file if not already present, and input a token configuration for a given URL, replacing the current token. It sets the given URL as the default if one is not already set. It uses getpass to hide the token from view.

Example usage:

```
set_token()
# user prompted for input of token, for https://vires.services/ows

set_token(url="https://vires.services/ows")
# user prompted for input of token, for given url

set_token(url="https://vires.services/ows", token="...")
# set a given url and token (no prompting)
```

1.10.6 DataUpload

class viresclient.**DataUpload**(*url*, *token*, ***kwargs*)

Bases: object

VirES for Swarm data upload API proxy.

Example usage:

```
from viresclient import ClientConfig, DataUpload

du = DataUpload("https://vires.services", token="...")

cc = ClientConfig()
url = cc.default_url
du = DataUpload(url, **cc.get_site_config(url))

# upload file
info = du.post("example.csv")
print(info)

# get information about the uploaded files
info = du.get()
print(info)

# remove any uploaded files
du.clear()

# check if the upload is valid and get list of missing mandatory parameters
info = du.post("example.cdf")
is_valid = info.get('is_valid', True)
missing_fields = info.get('missing_fields', {}).keys()
print(is_valid, missing_fields)

# get constant parameters
id = info['identifier']
parameters = du.get_constant_parameters(id)
print(parameters)
```

(continues on next page)

(continued from previous page)

```
# set new constant parameters
parameters = du.set_constant_parameters(id, {'Radius': 7000000, 'Latitude': 24.0})
print(parameters)

# clear all constant parameters
parameters = du.set_constant_parameters(id, {}, replace=True)
print(parameters)
```

For more information about the supported file format see the file format specification

exception Error

Bases: Exception

Data upload error exception.

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

PATH_OWS = '/ows'

PATH_UPLOAD = '/custom_data/'

clear()

Remove all uploaded items.

delete(*identifier*)

REST/API DELETE request. Delete item of the given identifier.

get(*identifier=None*)

REST/API GET If an identifier provided, get info about the uploaded item. If no identifier provided, list all uploaded items.

classmethod get_api_url(*url*)

Translate WPS URL path to the upload REST/API URL path.

get_constant_parameters(*identifier*)

Get dictionary of the currently set constant parameters.

classmethod get_ows_url(*url*)

Translate REST/API URL path to the upload WPS URL path.

property ids

Get list of identifiers.

patch(*identifier*, *data*)

REST/API PATCH Update metadata of the uploaded dataset.

post(*file*, *filename=None*)

HTTP POST multipart/form-data Upload file to the server and get info about the uploaded file.

set_constant_parameters(*identifier*, *parameters*, *replace=False*)

Set constant parameters form from give key value dictionary. Set replace to True if you prefer to replace the already set parameters rather then update them.

1.11 Command Line Interface (CLI)

Get list of the available CLI commands:

```
$ viresclient --help
```

Get help on a specific CLI command:

```
$ viresclient <command> --help
```

1.11.1 1. Configuration

Set default server:

```
$ viresclient set_default_server <url>
```

Remove default server:

```
$ viresclient remove_default_server <url>
```

Set access token configuration (aka site configuration):

```
$ viresclient set_token <url>
Enter access token: r-8-mlkP_RBx4mDv0di5Bzt3UZ52NGg-
```

Remove site configuration configuration:

```
$ viresclient set_token remove_server <url>
```

Show current configuration:

```
$ viresclient show_configuration
...
```

Remove stored configuration:

```
$ viresclient clear_credentials
```

1.11.2 2. Data Upload

Upload CDF or CSV data files to VirES for Swarm server:

```
$ viresclient upload_file <url> <filename>
```

e.g.:

```
$ viresclient upload_file https://vires.services/ows ./test.csv
69f91765-ce9f-4be3-8475-76f7e0eb1d92[test.csv] uploaded
```

Show information about the currently uploaded file:

```
$ viresclient show_uploads <url>
```

e.g.:

```
$ viresclient show_uploads https://vires.services/ows
69f91765-ce9f-4be3-8475-76f7e0eb1d92
  filename:      test.csv
  is valid:      True
  data start:    2016-01-01T00:30:00Z
  data end:      2016-01-01T04:30:00Z
  uploaded on:   2019-09-04T08:50:30.592982Z
  content type:  text/csv
  size:          421
  MD5 checksum:  8afe55c47385d1556117e81bf5ba8c34
  fields:
    B_C
    B_E
    B_N
    B_NEC
    Latitude
    Longitude
    Observatory
    Radius
    Timestamp
```

or:

```
$ viresclient show_uploads https://staging.vires.services/ows
eff3bd47-0098-45d3-ba9e-158bea0fae12
  filename:      test_incomplete.cdf
  is valid:      False
  data start:    2019-02-02T23:59:59Z
  data end:      2019-02-03T23:59:58Z
  uploaded on:   2019-09-11T08:04:58.055294Z
  content type:  application/x-cdf
  size:          959888
  MD5 checksum:  778cfe6e60568b08750187dfc917b3e8
  missing mandatory fields:
    Latitude
    Longitude
  constant fields:
    Radius=6371200.0
  fields:
    B_NEC
    F
    Radius
    Timestamp
```

Note the `is valid: False` flag, missing mandatory fields `Latitude` and `Longitude`, and extra constant field `Radius`.

Remove any uploaded file:

```
$ viresclient clear_uploads <url>
```

e.g.:

```
$ viresclient clear_uploads https://vires.services/ows
69f91765-ce9f-4be3-8475-76f7e0eb1d92[test.csv] removed
```

Setting extra constant parameters:

```
$ viresclient set_upload_parameters https://staging.vires.services/ows -p "Latitude=43.78
↪" -p "Longitude=12.34"
eff3bd47-0098-45d3-ba9e-158bea0fae12: parameters updated

$ viresclient show_uploads https://staging.vires.services/ows
eff3bd47-0098-45d3-ba9e-158bea0fae12
  filename:      test_incomplete.cdf
  is valid:      True
  data start:    2019-02-02T23:59:59Z
  data end:      2019-02-03T23:59:58Z
  uploaded on:   2019-09-11T08:04:58.055294Z
  content type:  application/x-cdf
  size:          959888
  MD5 checksum:  778cfe6e60568b08750187dfc917b3e8
  constant fields:
    Latitude=43.78
    Longitude=12.34
    Radius=6371200.0
  fields:
    B_NEC
    F
    Latitude
    Longitude
    Radius
    Timestamp
```

Removing constant parameters:

```
$ viresclient clear_upload_parameters https://staging.vires.services/ows
eff3bd47-0098-45d3-ba9e-158bea0fae12: parameters removed

$ viresclient show_uploads https://staging.vires.services/ows
eff3bd47-0098-45d3-ba9e-158bea0fae12
  filename:      test_tt2000.cdf
  is valid:      False
  data start:    2019-02-02T23:59:59Z
  data end:      2019-02-03T23:59:58Z
  uploaded on:   2019-09-11T08:04:58.055294Z
  content type:  application/x-cdf
  size:          959888
  MD5 checksum:  778cfe6e60568b08750187dfc917b3e8
  missing mandatory fields:
    Latitude
    Longitude
  fields:
    B_NEC
    F
    Timestamp
```

INDEX

M

O

P

R

S

T

W