ESGF Pyclient Documentation

Release 0.3.1

Ag Stephens

Nov 16, 2022

CONTENTS

ESGF PyClient is a Python package designed for interacting with the Earth System Grid Federation system. Currently this package contains API code for calling the ESGF Search API within client code.

You can try it online using Binder, or view the notebooks on NBViewer.

Please submit bugs and feature requests through the bug tracker on GitHub. Pull requests are always welcome.

Full documentation is available on ReadTheDocs or in the docs directory.

```
$ conda create -n esgf-pyclient -c conda-forge esgf-pyclient
$ conda activate esgf-pyclient
```

Once installed you import the package as the name `pyesgf`:

```
from pyesgf.search import SearchConnection
conn = SearchConnection('https://esgf.ceda.ac.uk/esg-search',
                        distrib=True)
ctx = conn.new_context(project='CMIP5', query='humidity')
ctx.hit_count
```

See the *Examples*.

CHAPTER

ONE

EXAMPLES

You can try these notebook online using Binder, or view the notebooks on NBViewer.

1.1 Basic Usage

1.1.1 Examples of pyesgf.search usage

Prelude:

[ ]:
```
from pyesgf.search import SearchConnection
conn = SearchConnection('https://esgf.ceda.ac.uk/esg-search',
                        distrib=True)
```

Warning: don't use default search with `facets=*`.

This behavior is kept for backward-compatibility, but ESGF indexes might not successfully perform a distributed search when this option is used, so some results may be missing. For full results, it is recommended to pass a list of facets of interest when instantiating a context object. For example,

```
ctx = conn.new_context(facets='project,experiment_id')
```

Only the facets that you specify will be present in the `facets_counts` dictionary.

This warning is displayed when a distributed search is performed while using the `facets=*` default, a maximum of once per context object. To suppress this warning, set the environment variable `ESGF_PYCLIENT_NO_FACETS_STAR_WARNING` to any value or explicitly use `conn.new_context(facets='*')`

[ ]:
```
facets='project,experiment_family'
```

Find how many datasets containing *humidity* in a given experiment family:

[ ]:
```
ctx = conn.new_context(project='CMIP5', query='humidity', facets=facets)
ctx.hit_count
```

[ ]:
```
ctx.facet_counts['experiment_family']
```

Search using a partial ESGF dataset ID (and get first download URL):

```
[ ]: conn = SearchConnection('https://esgf.ceda.ac.uk/esg-search', distrib=False)
     ctx = conn.new_context(facets=facets)
     dataset_id_pattern = "cmip5.output1.MOHC.HadGEM2-CC.historical.mon.atmos.Amon.*"
     results = ctx.search(query="id:%s" % dataset_id_pattern)
     len(results)
```

```
[ ]: files = results[0].file_context().search()
     len(files)
```

```
[ ]: download_url = files[0].download_url
     print(download_url)
```

Find the OpenDAP URL for an aggregated dataset:

```
[ ]: conn = SearchConnection('http://esgf-data.dkrz.de/esg-search', distrib=False)
     ctx = conn.new_context(project='CMIP5', model='MPI-ESM-LR', experiment='decadal2000',␣
     ↪time_frequency='day')
     print('Hits: {}, Realms: {}, Ensembles: {}'.format(
         ctx.hit_count,
         ctx.facet_counts['realm'],
         ctx.facet_counts['ensemble']))
```

```
[ ]: ctx = ctx.constrain(realm='atmos', ensemble='r1i1p1')
     ctx.hit_count
```

```
[ ]: result = ctx.search()[0]
     agg_ctx = result.aggregation_context()
     agg = agg_ctx.search()[0]
     print(agg.opendap_url)
```

Find download URLs for all files in a dataset:

```
[ ]: conn = SearchConnection('http://esgf-data.dkrz.de/esg-search', distrib=False)
     ctx = conn.new_context(project='obs4MIPs')
     ctx.hit_count
```

```
[ ]: ds = ctx.search()[0]
     files = ds.file_context().search()
     len(files)
```

```
[ ]: for f in files:
         print(f.download_url)
```

Define a search for datasets that includes a temporal range:

```
[ ]: conn = SearchConnection('https://esgf.ceda.ac.uk/esg-search', distrib=False)
     ctx = conn.new_context(
         project="CMIP5", model="HadGEM2-ES",
```

(continues on next page)

(continued from previous page)

```
    time_frequency="mon", realm="atmos", ensemble="r1i1p1", latest=True,
    from_timestamp="2100-12-30T23:23:59Z", to_timestamp="2200-01-01T00:00:00Z")
ctx.hit_count
```

Or do the same thing by searching without temporal constraints and then applying the constraint:

```
[ ]: ctx = conn.new_context(
    project="CMIP5", model="HadGEM2-ES",
    time_frequency="mon", realm="atmos", ensemble="r1i1p1", latest=True)
ctx.hit_count
```

```
[ ]: ctx = ctx.constrain(from_timestamp = "2100-12-30T23:23:59Z", to_timestamp = "2200-01-
↪01T00:00:00Z")
ctx.hit_count
```

1.1.2 Examples of pyesgf.logon usage

NOTE: For the logon module you need to install the latest `myproxyclient` from pypi:

```
$ conda create -c conda-forge -n esgf-pyclient python=3.6 pip esgf-pyclient
$ conda activate esgf-pyclient
(esgf-pyclient) pip install myproxyclient
```

Obtain MyProxy credentials to allow downloading files or using secured OpenDAP:

```
[ ]: from pyesgf.logon import LogonManager
lm = LogonManager()
lm.logoff()
lm.is_logged_on()
```

NOTE: When you run it for the first time you need to set `bootstrap=True`.

```
[ ]: OPENID = 'https://esgf-data.dkrz.de/esgf-idp/openid/USERNAME'
lm.logon_with_openid(openid=OPENID, password=None, bootstrap=True)
lm.is_logged_on()
```

NOTE: you may be prompted for your username if not available via your OpenID.

Obtain MyProxy credentials from the MyProxy host in *interactive* mode asking you for *username* and *password*:

```
[ ]: myproxy_host = 'esgf-data.dkrz.de'
lm.logon(hostname=myproxy_host, interactive=True, bootstrap=True)
lm.is_logged_on()
```

NOTE: See the `pyesgf.logon` module documentation for details of how to use myproxy username instead of OpenID.

1.1.3 Examples of pyesgf download usage

Obtain MyProxy credentials to allow downloading files:

```
[ ]: from pyesgf.logon import LogonManager
     lm = LogonManager()
     lm.logoff()
     lm.is_logged_on()
```

```
[ ]: myproxy_host = 'esgf-data.dkrz.de'
     lm.logon(username=None, password=None, hostname=myproxy_host)
     lm.is_logged_on()
```

Now download a file using the ESGF wget script extracted from the server:

```
[ ]: from pyesgf.search import SearchConnection
     conn = SearchConnection('https://esgf-data.dkrz.de/esg-search', distrib=False)
     ctx = conn.new_context(project='obs4MIPs', institute='FUB-DWD')
     ds = ctx.search()[0]

     import tempfile
     fc = ds.file_context()
     wget_script_content = fc.get_download_script()
     script_path = tempfile.mkstemp(suffix='.sh', prefix='download-')[1]
     with open(script_path, "w") as writer:
         writer.write(wget_script_content)

     import os, subprocess
     os.chmod(script_path, 0o750)
     download_dir = os.path.dirname(script_path)
     subprocess.check_output("{}".format(script_path), cwd=download_dir)
```

... and the files will be downloaded to a temporary directory:

```
[ ]: print(download_dir)
```

If you are doing batch searching and things are running slow, you might be able to achieve a considerable speed up by sending the following argument to the search call:

```
[ ]: ctx.search(ignore_facet_check=True)
```

This cuts out an extra call that typically takes 2 seconds to return a response. Note that it may mean some of the functionality is affected (such as being able to view the available facets and access the hit count) so use this feature with care.

You can also dictate how the search batches up its requests with:

```
[ ]: ctx.search(batch_size=250)
```

The `batch_size` argument does not affect the final result but may affect the speed of the response. The batch size can also be set as a default in the `pyesgf.search.consts` module.

1.2 Demo

1.2.1 Subset CMIP5 Datasets with xarray

xarray: http://xarray.pydata.org/en/stable/index.html

Search CMIP5 Dataset with ESGF pyclient

using: https://esgf-pyclient.readthedocs.io/en/latest/index.html

[ ]:
```
from pyesgf.search import SearchConnection
conn = SearchConnection('https://esgf-data.dkrz.de/esg-search', distrib=True)
```

[ ]:
```
ctx = conn.new_context(
    project='CMIP5',
    experiment='rcp45',
    model='HadCM3',
    ensemble='r1i1p1',
    time_frequency='mon',
    realm='atmos',
    data_node='esgf-data1.ceda.ac.uk',
    )
ctx.hit_count
```

[ ]:
```
result = ctx.search()[0]
result.dataset_id
```

[ ]:
```
files = result.file_context().search()
for file in files:
    if 'tasmax' in file.opendap_url:
        tasmax_url = file.opendap_url
        print(tasmax_url)
```

ESGF Logon

[ ]:
```
from pyesgf.logon import LogonManager
lm = LogonManager()
lm.logoff()
lm.is_logged_on()
```

[ ]:
```
lm.logon(hostname='esgf-data.dkrz.de', interactive=True, bootstrap=True)
lm.is_logged_on()
```

Subset single dataset with xarray

Using OpenDAP: http://xarray.pydata.org/en/stable/io.html?highlight=opendap#opendap

```
[ ]: import xarray as xr
ds = xr.open_dataset(tasmax_url, chunks={'time': 120})
print(ds)
```

```
[ ]: da = ds['tasmax']
da = da.isel(time=slice(0, 1))
da = da.sel(lat=slice(-50, 50), lon=slice(0, 50))
```

```
[ ]: %matplotlib inline
da.plot()
```

Download to NetCDF

```
[ ]: da.to_netcdf('tasmax.nc')
```

1.2.2 Subset CMIP6 Datasets with xarray

xarray: http://xarray.pydata.org/en/stable/index.html

Search CMIP6 Dataset with ESGF pyclient

using: https://esgf-pyclient.readthedocs.io/en/latest/index.html

```
[ ]: from pyesgf.search import SearchConnection
conn = SearchConnection('https://esgf-data.dkrz.de/esg-search', distrib=True)
```

```
[ ]: ctx = conn.new_context(
    project='CMIP6',
    source_id='UKESM1-0-LL',
    experiment_id='historical',
    variable='tas',
    frequency='mon',
    variant_label='r1i1p1f2',
    data_node='esgf-data3.ceda.ac.uk')
ctx.hit_count
```

```
[ ]: result = ctx.search()[0]
result.dataset_id
```

```
[ ]: files = result.file_context().search()
for file in files:
    print(file.opendap_url)
```

Subset single dataset with xarray

Using OpenDAP: http://xarray.pydata.org/en/stable/io.html?highlight=opendap#opendap

[ ]:
```
import xarray as xr
ds = xr.open_dataset(files[0].opendap_url, chunks={'time': 120})
print(ds)
```

[ ]:
```
da = ds['tas']
da = da.isel(time=slice(0, 1))
da = da.sel(lat=slice(-50, 50), lon=slice(0, 50))
```

[ ]:
```
%matplotlib inline
da.plot()
```

Subset over multiple datasets

[ ]:
```
ds_agg = xr.open_mfdataset([files[0].opendap_url, files[1].opendap_url], chunks={'time':␣
↪120}, combine='nested', concat_dim='time')
print(ds_agg)
```

[ ]:
```
da = ds_agg['tas']
da = da.isel(time=slice(1200, 1201))
da = da.sel(lat=slice(-50, 50), lon=slice(0, 50))
```

[ ]:
```
da.plot()
```

Download dataset

[ ]:
```
da.to_netcdf('tas_africa_19500116.nc')
```

CHAPTER

TWO

INSTALLATION

2.1 Install from PyPI

```
$ pip install esgf-pyclient
```

2.2 Install from Anaconda

```
$ conda install -c conda-forge esgf-pyclient
```

2.3 Install from GitHub

Get esgf-pyclient source from GitHub:

```
$ git clone https://github.com/ESGF/esgf-pyclient.git
$ cd esgf-pyclient
```

Optionally create Conda environment named *esgf-pyclient*:

```
$ conda env create -f environment.yml
$ conda activate esgf-pyclient
```

Install esgf-pyclient:

```
$ pip install -e .
OR
$ make install
```

For development you can use this command:

```
$ pip install -e .[dev]
OR
$ make develop
```

CHAPTER

THREE

DEVELOPMENT

3.1 Get Started!

Check out code from the esgf-pyclient GitHub repo and start the installation:

```
$ git clone https://github.com/ESGF/esgf-pyclient.git
$ cd esgf-pyclient
$ conda env create -f environment.yml
$ pip install -e .[dev]
```

When you're done making changes, check that your changes pass *flake8* and the tests:

```
$ flake8
$ pytest
```

Or use the Makefile:

```
$ make lint
$ make test  # skip slow tests
$ make test-all
```

3.2 Write Documentation

You can find the documentation in the *docs/source* folder. To generate the Sphinx documentation locally you can use the *Makefile*:

```
$ make docs
```

3.3 Bump a new version

Make a new version of esgf-pyclient in the following steps:

- Make sure everything is commit to GitHub.
- Update `CHANGES.rst` with the next version.
- Dry Run: `bumpversion --dry-run --verbose --new-version 0.3.1 patch`
- Do it: `bumpversion --new-version 0.3.1 patch`

- … or: `bumpversion --new-version 0.4.0 minor`
- Push it: `git push --tags`

See the bumpversion documentation for details.

CHAPTER

FOUR

DESIGN CONCEPTS

4.1 Search Concepts

The *pyesgf.search* interface to ESGF search reflects the typical workflow of a user navigating through the sets of facets categorising available data.

4.1.1 Keyword classification

The keyword arguments described in the ESGF Search API have a wide veriety of roles within the search workflow. To reflect this *pyesgf.search* classifies these keywords into system, spatiotemporal and facet keywords. Responsibility for these keywords are distributes across several classes.

System keywords

API keyword	class	Notes
limit	SearchConnection	Set in `SearchConnection:send_query()` method or transparently through `SearchContext`
offset	SearchConnection	Set in `SearchConnection:send_query()` method or transparently through `SearchContext`
shards	SearchConnection	Set in constructor
distrib	SearchConnection	Set in constructor
latest	SearchContext	Set in constructor
facets	SearchContext	Set in constructor
fields	SearchContext	Set in constructor
replica	SearchContext	Set in constructor
type	SearchContext	Create contexts with the right type using `ResultSet.file_context()`, etc.
from	SearchContext	Set in constructor. Use “from_timestamp” in the context API.
to	SearchContext	Set in constructor. Use “to_timestamp” in the context API.
fields	n/a	Managed internally
format	n/a	Managed internally
id	n/a	Managed internally

Temporal keywords

Temporal keywords are supported for Dataset search. The terms "from_timestamp" and "to_timestamp" should be used with values following the format "YYYY-MM-DDThh:mm:ssZ".

Spatial keywords

Spatial keywords are not yet supported by *pyesgf.search* however the API does have placeholders for these keywords anticipating future implementation:

Facet keywords

All other keywords are considered to be search facets. The keyword "query" is dealt with specially as a freetext facet.

4.1.2 Main Classes

SearchConnection

`SearchConnection` instances represent a connection to an ESGF Search web service. This stores the service URL and also service-level parameters like distrib and shards.

SearchContext

`SearchContext` represents the constraints on a given search. This includes the type of records you are searching for (File or Dataset), the list of possible facets with or without facet counts (depending on how the instance is created), currently selected facets/search-terms. Instances can return the number of hits and facet-counts associated with the current search.

SearchContext objects can be created in several ways:

1. From a SearchConnection object using the method `SearchConnection.new_context()`
2. By further constraining an existing FacetContext object. E.g. new_context = context.constrain(institute='IPSL').
3. From a Result object using one of it's *foo_context()* methods to create a context for searching for results related to the Result.
4. Future development may implement project-specific factory. E.g. CMIP5FacetContext().

ResultSet

`ResultSet` instances are returned by the `SearchContext.search()` method and represent the results from a query. They supports transparent paging of results with a client-side cache.

Result

`Result` instances represent the result record in the SOLr response. They are subclassed to represent records of different types: `FileResult` and `DatasetResult`. Results have various properties exposing information about the objects they represent. e.g. dataset_id, checksum, filename, size, etc.

CHAPTER

FIVE

API REFERENCE

5.1 Search API

An interface to the ESGF Search API

5.1.1 Module `pyesgf.search.connection`

Defines the class representing connections to the ESGF Search API. To perform a search create a *SearchConnection* instance then use `new_context()` to create a search context.

Warning: Prior to v0.1.1 the *url* parameter expected the full URL of the search endpoint up to the query string. This has now been changed to expect *url* to ommit the final endpoint name, e.g. `https://esgf-node.llnl.gov/esg-search/search` should be changed to `https://esgf-node.llnl.gov/esg-search` in client code. The current implementation detects the presence of `/search` and corrects the URL to retain backward compatibility but this feature may not remain in future versions.

class pyesgf.search.connection.**SearchConnection**(*url*, *distrib=True*, *cache=None*, *timeout=120*, *expire_after=datetime.timedelta(seconds=3600)*, *session=None*, *verify=True*, *context_class=None*)

Variables

- **url** – The URL to the Search API service. This should be the URL of the ESGF search service excluding the final endpoint name. Usually this is http://<hostname>/esg-search
- **distrib** – Boolean stating whether searches through this connection are distributed. i.e. whether the Search service distributes the query to other search peers. See also the documentation for the `facets` argument to `pyesgf.search.context.SearchContext` in relation to distributed searches.
- **cache** – Path to *sqlite* cache file. Cache expires every hours.
- **timeout** – Time (in seconds) before query returns an error. Default: 120s.
- **expire_after** – Time delta after cache expires. Default: 1 hour.

- **session** – requests.session object. optional.
- **verify** – boolean, determines if query should be sent over a verified channel.

get_shard_list()

return the list of all available shards. A subset of the returned list can be supplied to 'send_query()' to limit the query to selected shards.

Shards are described by hostname and mapped to SOLr shard descriptions internally.

Returns
the list of available shards

new_context(*context_class=None*, *latest=None*, *facets=None*, *fields=None*, *from_timestamp=None*, *to_timestamp=None*, *replica=None*, *shards=None*, *search_type=None*, ***constraints*)

Returns a *pyesgf.search.context.SearchContext* class for performing faceted searches.

See `SearchContext.__init__()` for documentation on the arguments.

send_search(*query_dict*, *limit=None*, *offset=None*, *shards=None*)

Send a query to the "search" endpoint. See `send_query()` for details.

Returns
The json document for the search results

send_wget(*query_dict*, *shards=None*)

Send a query to the "search" endpoint. See `send_query()` for details.

Returns
A string containing the script.

pyesgf.search.connection.**create_single_session**(*cache=None*, *expire_after=datetime.timedelta(seconds=3600)*, ***kwargs*)

Simple helper function to start a requests or requests_cache session.

cache, if specified is a filename to a threadsafe sqlite database expire_after specifies how long the cache should be kept

pyesgf.search.connection.**query_keyword_type**(*keyword*)

Returns the keyword type of a search query keyword.

Possible values are 'system', 'freetext', 'facet', 'temporal' and 'geospatial'. If the keyword is unknown it is assumed to be a facet keyword

5.1.2 Module pyesgf.search.context

Defines the *SearchContext* class which represents each ESGF search query.

class pyesgf.search.context.**AggregationSearchContext**(*connection*, *constraints*, *search_type=None*, *latest=None*, *facets=None*, *fields=None*, *from_timestamp=None*, *to_timestamp=None*, *replica=None*, *shards=None*)

class pyesgf.search.context.**DatasetSearchContext**(*connection*, *constraints*, *search_type=None*, *latest=None*, *facets=None*, *fields=None*, *from_timestamp=None*, *to_timestamp=None*, *replica=None*, *shards=None*)

class pyesgf.search.context.**FileSearchContext**(*connection*, *constraints*, *search_type=None*, *latest=None*, *facets=None*, *fields=None*, *from_timestamp=None*, *to_timestamp=None*, *replica=None*, *shards=None*)

class pyesgf.search.context.**SearchContext**(*connection*, *constraints*, *search_type=None*, *latest=None*, *facets=None*, *fields=None*, *from_timestamp=None*, *to_timestamp=None*, *replica=None*, *shards=None*)

Instances of this class represent the state of a current search. It exposes what facets are available to select and the facet counts if they are available.

Subclasses of this class can restrict the search options. For instance FileSearchContext, DatasetSerachContext or CMIP5SearchContext

SearchContext instances are connected to SearchConnection instances. You normally create SearchContext instances via one of: 1. Calling SearchConnection.new_context() 2. Calling SearchContext.constrain()

Variables

- **constraints** – A dictionary of facet constraints currently in effect. `constraint[facet_name] = [value, value, ...]`
- **facets** – A string containing a comma-separated list of facets to be returned (for example `'source_id,ensemble_id'`). If set, this will be used to select which facet counts to include, as returned in the `facet_counts` dictionary. Defaults to including all available facets, but with distributed searches (where the SearchConnection instance was created with `distrib=True`), some results may be missing for server-side reasons when requesting all facets, so a warning message will be issued. This contains further details.

Property facet_counts

A dictionary of available hits with each facet value for the search as currently constrained. This property returns a dictionary of dictionaries where `facet_counts[facet][facet_value] == hit_count`

Property hit_count

The total number of hits available with current constraints.

constrain(***constraints*)

Return a *new* instance with the additional constraints.

get_download_script(***constraints*)

Download a script for downloading all files in the set of results.

Parameters

constraints – Further constraints for this query. Equivalent to calling `self.constrain(**constraints).get_download_script()`

Returns

A string containing the script

get_facet_options()

Return a dictionary of facet counts filtered to remove all facets that are completely constrained. This method is similar to the property `facet_counts` except facet values which are not relevant for further constraining are removed.

search(*batch_size=50*, *ignore_facet_check=False*, ***constraints*)

Perform the search with current constraints returning a set of results.

Batch_size

The number of results to get per HTTP request.

Ignore_facet_check
Do not make an extra HTTP request to populate `facet_counts` and `hit_count`.

Parameters
constraints – Further constraints for this query. Equivalent to calling `self.constrain(**constraints).search()`

Returns
A ResultSet for this query

5.1.3 Module pyesgf.search.results

Search results are retrieved through the *ResultSet* class. This class hides paging of large result sets behind a client-side cache. Subclasses of `Result` represent results of different SOLr record type.

`class pyesgf.search.results.AggregationResult(json, context)`

A result object for ESGF aggregations. Properties from *BaseResult* are inherited.

Property aggregation_id
The aggregation id

`class pyesgf.search.results.BaseResult(json, context)`

Base class for results.

Subclasses represent different search types such as File and Dataset.

Variables

- **json** – The original json representation of the result.
- **context** – The SearchContext which generated this result.

Property urls
a dictionary of the form `{service: [(url, mime_type), ...], ...}`

Property opendap_url
The url of an OPeNDAP endpoint for this result if available

Property las_url
The url of an LAS endpoint for this result if available

Property download_url
The url for downloading the result by HTTP if available

Property gridftp_url
The url for downloading the result by Globus if available

Property globus_url
The url for downloading the result by Globus if available (including endpoint)

Property index_node
The index node from where the metadata is stored. Calls to `*_context()` will optimise queries to only address this node.

`class pyesgf.search.results.DatasetResult(json, context)`

A result object for ESGF datasets.

Property dataset_id
The solr dataset_id which is unique throughout the system.

aggregation_context()

Return a SearchContext for searching for aggregations within this dataset.

file_context()

Return a SearchContext for searching for files within this dataset.

property number_of_files

Returns file count as reported by the dataset record.

class pyesgf.search.results.**FileResult**(*json*, *context*)

A result object for ESGF files. Properties from *BaseResult* are inherited.

Property file_id
The identifier for the file

Property checksum
The checksum of the file

Property checksum_type
The algorithm used for generating the checksum

Property filename
The filename

Property size
The file size in bytes

class pyesgf.search.results.**ResultSet**(*context*, *batch_size=50*, *eager=True*)

Variables
context – The search context object used to generate this resultset

Property batch_size
The number of results that will be requested from esgf-search as one call. This must be set on creation and cannot change.

5.2 ESGF Security API

pyesgf provides a simplified interface to obtaining ESGF credentials.

5.2.1 Module pyesgf.logon

Manage the client's interaction with ESGF's security system. Using this module requires installing the MyProxyClient library.

To obtain ESGF credentials create a *LogonManager* instance and supply it with logon details:

```
>>> lm = LogonManager()
>>> lm.is_logged_on()
False
>>> lm.logon(username, password, myproxy_hostname, bootstrap=True)
>>> lm.is_logged_on()
True
```

Logon parameters that aren't specified will be prompted for at the terminal by default. The *LogonManager* object also writes a `.httprc` file configuring OPeNDAP access through the NetCDF API.

The option `bootstrap=True` is needed on the first run.

You can use your OpenID to logon instead. The logon details will be deduced from the OpenID where possible:

```
>>> lm.logoff()
>>> lm.is_logged_on()
False
>>> lm.logon_with_openid(openid, password, bootstrap=True)
>>> lm.is_logged_on()
True
```

class pyesgf.logon.**LogonManager**(*esgf_dir='/home/docs/.esg'*, *dap_config='/home/docs/.dodsrc'*, *verify=True*)

Manages ESGF crendentials and security configuration files.

Also integrates with NetCDF's secure OPeNDAP configuration.

logoff(*clear_trustroots=False*)

Remove any obtained credentials from the ESGF environment.

Parameters
clear_trustroots – If True also remove trustroots.

logon(*username=None*, *password=None*, *hostname=None*, *bootstrap=False*, *update_trustroots=True*, *interactive=True*)

Obtain ESGF credentials from the specified MyProxy service.

If `interactive == True` then any missing parameters of `password`, `username` or `hostname` will be prompted for at the terminal.

Parameters

- **interactive** – Whether to ask for input at the terminal for any missing information. I.e. username, password or hostname.
- **bootstrap** – Whether to bootstrap the trustroots for this MyProxy service.
- **update_trustroots** – Whether to update the trustroots for this MyProxy service.

logon_with_openid(*openid*, *password=None*, *bootstrap=False*, *update_trustroots=True*, *interactive=True*)

Obtains ESGF credentials by detecting the MyProxy parameters from the users OpenID. Some ESGF compatible OpenIDs do not contain enough information to obtain credentials. In this case the user is prompted for missing information if `interactive == True`, otherwise an exception is raised.

Parameters
openid – OpenID to login with See *logon()* for parameters `interactive`, `bootstrap` and `update_trustroots`.

CHAPTER

SIX

RELEASE NOTES

6.1 0.3.1 (2022-02-25)

- Fix: fix tests and merge conflicts (#79).
- Fix #75: ignore_facet_check search option appears to be broken (#76).
- Fix #74: Add warnings when default facets=* used on distributed search (#77).
- Fix #78: Updates for requests_cache API (#68).
- Fix: Improvements to tests (#73).

6.2 0.3.0 (2021-02-08)

- Added test for batch size (#66).
- Test notebooks (#65).
- Replaced Travis CI by GitHub actions CI (#64).
- Return globus urls in search (#63).
- Fixed build of Search API (#61).
- Remove unused code (#49).
- Cleaned up tests (#45).
- Using `webob.multidict` (#43).
- Cleaned up docs (#39).
- Marked slow tests (#38).
- Added notebook examples (#37, #46, #48).
- Fixed the usage of the `input()` method in `logon.py` (#36).
- Skip Python 2.x (#35).

6.3 0.2.2 (2019-07-19)

- Fixed test suite (#33)
- Added badges for RTD and Travis CI to Readme.

6.4 0.2.1 (2018-03-14)

This release includes the following features:

- The library now supports **Python 3**.
- `verify` option in `LogonManager` has been added.
- Works with Python 3 version of `MyProxyClient`.
- Testing structure with `pytest` has been improved.

6.5 0.1.8 (2017-01-03)

This release includes the following changes:

- The tests have been updated and various fixes made to make them match the up-to-date ESGF Search API.
- Following problems with the search being slow in certain scenarios an extra call to the Search service was made optional through the `SearchContext.search()` method. If you send the argument and value of `ignore_facet_check=True` then this hidden call to the service will be avoided. This typically saves 2 seconds of wait time which can be very important in some iterative search scenarios.
- The `SearchContext.search()` method was also extended so that the argument `batch_size` could be directly sent to it in order to manage how the calls to the API would be separated out into batches. This does not affect the final result but may affect the speed of the response. The batch size can also be set as a default in the `pyesgf.search.consts` module.
- Searches at the file-level now return a `gridftp_url` property along with other existing properties such as `download_url`.

6.6 0.1.6 (2016-05-16)

This release includes a fix for Issue #4 to cope with ESGF Search end points being given with or without a ":port" component in the host address.

6.7 0.1.1 (2013-04-15)

This release will include wget script download support.

Warning: The expected value of the *url* parameter to `SearchConnection()` has changed in this release. Prior to v0.1.1 the *url* parameter expected the full URL of the search endpoint up to the query string. This has now

been changed to expect *url* to omit the final endpoint name, e.g. `https://esgf-node.llnl.gov/esg-search/search` should be changed to `https://esgf-node.llnl.gov/esg-search` in client code. The current implementation detects the presence of `/search` and corrects the URL to retain backward compatibility but this feature may not remain in future versions.

This release changes the call signature of `SearchConnection.send_query()` and introduces the additional methods `send_search()` and `send_wget()`. When upgrading code to work with this new API simly:

1. Change the *url* parameter to `SearchConnection()` to not include the `/search` suffix
2. Change any occurrence of `send_query()` to `send_search()`

6.8 0.1b1 (2013-01-19)

This release marks the start of the 0.1 series which is considered beta-quality. API changes in this series will be clearly marked in the documentation and backward-compatible releases will be maintained on pypi.

The 0.1b1 release includes integrated MyProxy logon support in the `pyesgf.logon` module. This release also includes optimisations to the search system to avoid querying multiple shards when requesting the files from a dataset.

PYTHON MODULE INDEX

p

INDEX