ESA CCI Toolbox

Release 0.4.2.dev0

Brockmann Consult GmbH

May 05, 2024

TABLE OF CONTENTS

The CCI Toolbox is a python package that provides access and operations to CCI data. It is available on GitHub and can be installed with Conda.

CHAPTER

ONE

DATA STORES

The CCI Toolbox comes with three pre-configured data stores, built with the xcube Python package. The CCI Open Data Portal data store (esa-cci) provides programmatic access to all CCI data. The Zarr data store (esa-cci-zarr) provides access to selected CCI data in Zarr format for faster performance. The kerchunk Data Store (esa-cci-kc) gives access to CCI datasets through a reference file, thereby allowing similar performance as the Zarr Store

CHAPTER

TWO

DATASETS

Datasets are accessed through Data Stores. By providing a dataset identifier the CCI Toolbox loads only the metadata and structure of the dataset, with the full dataset loaded only when needed for operations. Opened datasets are represented through data structures defined by Python packages xarray, pandas, and geopandas.

CHAPTER

THREE

OPERATIONS

The CCI Toolbox provides climate analyses operations geared to CCI data for *Coregistration*, *Resampling*, spatial and temporal *Subsetting*, *Aggregation*, *Anomalies*, *Arithmetics*, selected *Data Frame Operations* and more. In addition, the Python packages xarray, pandas, and geopandas provide a rich and powerful low-level data processing interface for datasets opened through the CCI Toolbox. See the *API Reference* for details.

CHAPTER

FOUR

INSTALLATION

Method 1 - Install Conda and then run the following

```
$ conda create --name ect --channel conda-forge esa-climate-toolbox
$ conda activate ect
```

Method 2 - If you already have an existing Conda environment

```
$ conda install --channel conda-forge esa-climate-toolbox
```

Method 3 - Install directly from the GitHub repository

```
$ git clone https://github.com/esa-cci/esa-climate-toolbox.git
$ cd esa-climate-toolbox
$ conda env create
$ conda activate ect
$ pip install -e .
```

CHAPTER

FIVE

GETTING STARTED

Try our *Jupyter Notebooks* on exploring CCI data, accessing data from the Open Data Portal Store, from the Zarr Store, and from the Kerchunk Store; on opening data subsets and opening data as dataframes, and on finding and using operations on CCI data.

CHAPTER

SIX

HELPDESK

For support with the CCI Toolbox, please visit our Helpdesk.

6.1 Introduction

6.1.1 Project Background

ESA's Climate Change Initiative is a major research and development effort that generates global, decades-long satellite data records to track and understand key aspects of the Earth climate system, and known as Essential Climate Variables.

The Programme enables a community of over 500 experts from across Europe to exploit the Earth observation archive and data from operating satellite missions to craft high-quality data products that strengthen scientific understanding climate and underpin the models and climate services used to inform support ESA Member States to take climate action and report progress towards the Paris Agreement goals.

These data records support the United Nations Framework Convention on Climate Change and the International Panel on Climate Change to monitor, assess and address changes to Earth's climate system.

The CCI Toolbox and the CCI Open Data Portal are the two main technical support projects within the programme. The CCI Open Data Portal provides a single point of harmonised access to a subset of mature and validated ECV-related data products.

The CCI Toolbox provides tools that support visualisation, analysis and processing across CCI and other climate data products. With these two technical cross-cutting activities ESA is providing an interface between its CCI projects and the ECVs generated there, and the wider climate change user community.

6.1.2 Key Objectives

The four key objectives of the CCI Toolbox are:

- Provide to climate users an intuitive software application that is capable of **ingesting data from all CCI projects** and synergistically use this data in a uniform tooling environment. This requires the application to abstract from the various data types used to represent the different ECVs (vector data, n-D raster data), and from data formats used (NetCDF, Shapefiles), and also from various data sources (remote services, local files).
- Provide to users a rich set of **data processing operations** that implement commonly used climate algorithms. Processors can be used to build processing chains that represent typical climate workflows.
- Provide to users various **visualisation and analysis** operations. The majority of visualisation and analysis functions are applicable to multiple ECVs while others may only work if certain constraints are met. Some of these functions may be implemented as processors and be used in processing chains.

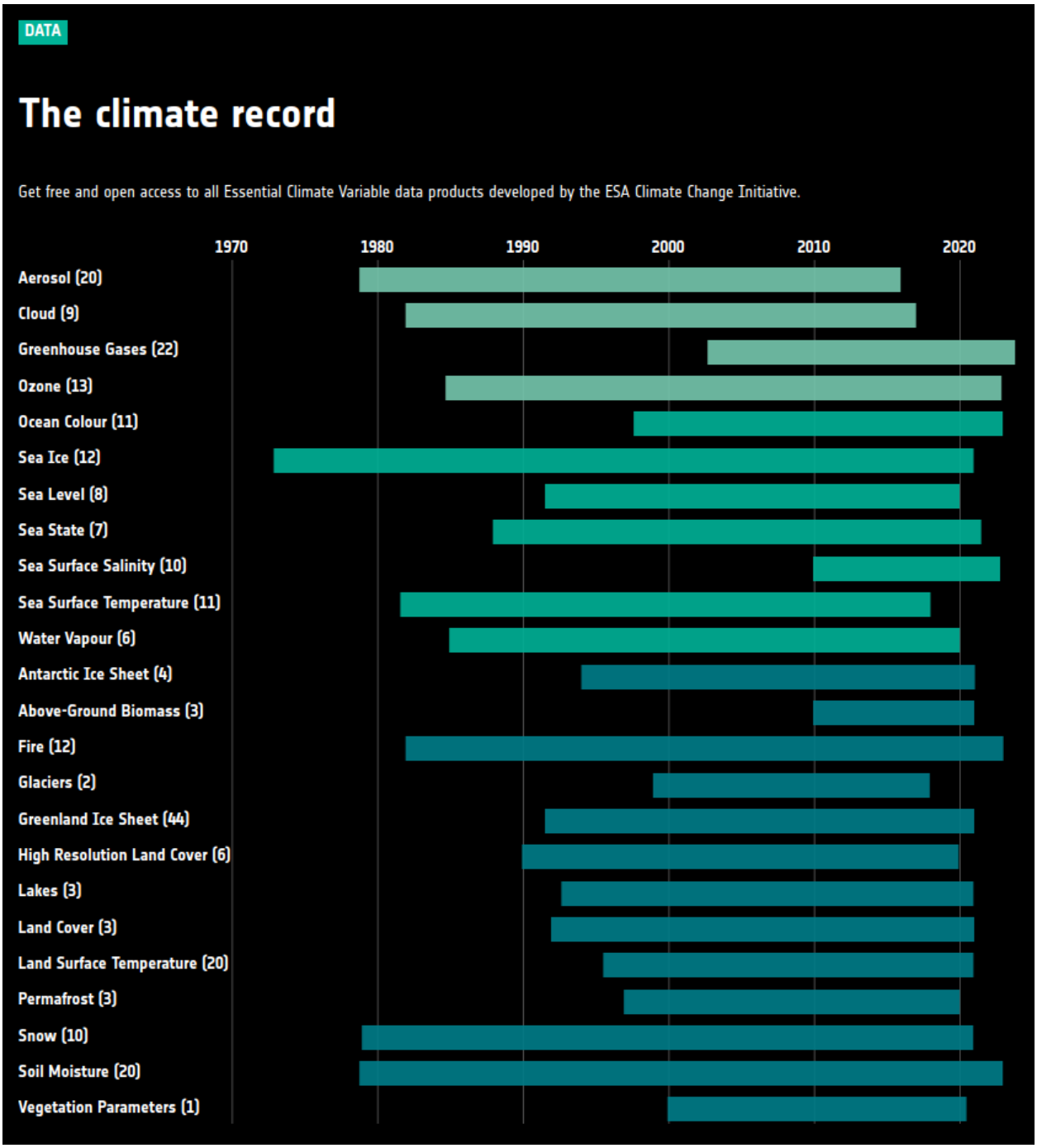


Fig. 1: Snapshot from CCI Open Data Portal with the available ECVs

- Provide the **architecture** of the CCI Toolbox so that it **can be extended by new climate operations** and that it also allows for **reuse of existing or planned software tools and libraries**. Furthermore allow other scientists and tool developers to use the underlying CCI Toolbox algorithms and libraries in their own programs.

6.2 About the ESA CCI Toolbox

The ESA CCI Toolbox is a software developed to facilitate processing and analysis of all the data products generated by the ESA Climate Change Initiative Programme (CCI). It supports analysis and interactive visualisation of these data products using its Python interface. The ESA CCI Toolbox **Python API** allows using the functions of the ESA CCI Toolbox in Python programs and may also be used to build extensions.

6.2.1 Concepts

The ESA CCI Toolbox software is based on a few simple concepts. To get the most out of using the toolbox, it makes sense to make oneself familiar with them before using the Toolbox.

Data Stores

The ESA CCI Toolbox uses the concept of data stores as provided by the xcube software package. This has the advantage that users may easily define their own stores to combine their own third-party data with the CCI data. Users may read more about stores in the documentation of the xcube data store framework.

The CCI toolbox comes with three pre-defined stores: The first one is the CCI Open Data Portal store, which is denoted by the handle *esa-cci*. It provides access to any data from the Open Data Portal.

The second data store is the Zarr store (*esa-cci-zarr*), which allows to access datasets from the Open Data Portal that have been converted to the Zarr format. This has been done for selected datasets which were either frequently used or large in terms of data volume. Providing the data as zarr files allows for a more performant data access. The number of Zarr datasets will be constantly increased.

The third data store is the Kerchunk Store (*esa-cci-kc*), which accesses datasets that are offered by the Open Data Portal via the references format. This format allows to access the files with a similar performance as the Zarr data store.

Additionally, the CCI Toolbox allows to define output stores, to which operation results may be written.

You can find detailed listings of the provided functionality in the *API Reference*.

Datasets

Data Stores provide access to datasets. You may open datasets from a data store by providing the dataset's identifier. The ESA CCI Toolbox will only read in a dataset's metadata and basic structure, but not actual data until explicitly requested (e.g., during the application of an operation or saving). That way, the Toolbox can deal with datasets that don't fit into your computer's memory. The ESA CCI Toolbox allows for *out-of-core* and *multi-core* processing. However, you can always read datasets directly from your local storage, . e.g., NetCDF files or ESRI Shapefiles.

The ESA CCI Toolbox does not invent new data structures for representing datasets in memory. Instead, opened datasets are represented by data structures defined by the popular Python packages xarray, pandas, and geopandas:

- Gridded and raster datasets (based on NetCDF/CF or Zarr) are represented by *xarray.Dataset* objects. Dataset variables are represented by NumPy-compatible *xarray.DataArray* objects.
- Vector datasets (from ESRI Shapefiles, GeoJSON files) are represented by *geopandas.GeoDataFrame* objects. Dataset variables are represented by pandas-compatible *geopandas.GeoSeries* objects.
- Tabular data (from CSV, Excel files) are represented by *pandas.DataFrame* objects.

Note that all remote CCI data set identifiers are prefixed by "esacci.", for example `esacci.SST.day.L4.SSTdepth.multi-sensor.multi-platform.OSTIA.1-0.r1`.

Functions and Operations

The ESA CCI Toolbox provides numerous I/O, analysis, and processing **operations** that address typical climate analyses. These *operations* are Python functions. The ESA CCI Toolbox has an operation registry where functions are registered. In addition to operations provided by the ESA CII Toolbox, the Python packages xarray, pandas, and geopandas provide a rich and powerful low-level data processing interface for the datasets opened through the Toolbox.

You can find detailed listings of the provided functionality in *API Reference*.

6.3 Installation Guide

6.3.1 Installation into a new environment

The ESA CCI Toolbox and all necessary dependencies are available on conda-forge and can be installed using the conda package manager. The conda package manager itself can be obtained in the miniconda distribution. Once conda is installed, the ESA CCI Toolbox can be installed like this:

```
$ conda create --name ect --channel conda-forge esa-climate-toolbox
$ conda activate ect
```

The name of the environment may be freely chosen.

6.3.2 Installation into an existing environment

The ESA CCI Toolbox can also be installed into an existing conda environment. To do so, execute this command with the existing environment activated:

```
$ conda install --channel conda-forge esa-climate-toolbox
```

In case you encounter any problems, try installing these packages first:

```
$ conda install --channel conda-forge aiohttp lxml nest-asyncio xcube "pydap==3.3"
```

6.3.3 Installation into a new environment from the repository

If you want to install the ESA CCI Toolbox directly from the git repository (for example in order to use an unreleased version or to modify the code), you can do so as follows:

```
$ git clone https://github.com/esa-cci/esa-climate-toolbox.git
$ cd esa-climate-toolbox
$ conda env create
$ conda activate ect
$ pip install -e .
```

6.4 Quick Start

After executing the instructions given in *Installation Guide* the ESA CCI Toolbox is ready to use. To show how to make first steps, a variety of notebooks has been compiled. You may find them in the *notebooks* section in the Github Repository, but we have also assembled them here to provide you with an overview.

6.4.1 ESA CCI Toolbox Notebooks Setup

Before using Jupyter Lab for the first time install the `jupyterlab` and `jupyterlab-geojson` packages.

```
(ect) conda install -c conda-forge jupyterlab jupyterlab-geojson
```

Start Jupyter Lab:

```
(ect) $ jupyter-lab
```

6.4.2 Jupyter Notebooks

ESA CCI Toolbox Finding Data

This notebook shows the usage of some covenience functions that have been setup to further facilitate the handling of the stores and the use of the ESA CCI Toolbox. A thorough description can be found here: https://esa-climate-toolbox.readthedocs.io/en/latest/api_reference.html#datasets-and-datastores. To see how you can actually open data using these stores, see notebooks 2-5.

To run this Notebook, make sure the ESA CCI Toolbox is setup correctly.

The functions provided here serve the purpose to get an overview about data and from which data store to get it from. They also serve to look for Essential Climate Variables. We may start by listing all that are provided by the ESA CCI Toolbox.

[1]:
```
from esa_climate_toolbox.core import list_ecvs

list_ecvs()
```

[1]:
```
['OC',
 'SOILMOISTURE',
 'LC',
 'SEAICE',
 'PERMAFROST',
 'VEGETATION',
 'SEASURFACESALINITY',
 'LST',
 'SST',
 'SEASTATE',
 'CLOUD',
 'AEROSOL',
 'GHG',
 'BIOMASS',
 'SNOW',
 'OZONE',
 'WATERVAPOUR',
```

(continues on next page)

(continued from previous page)

```
 'LAKES',
 'SEALEVEL',
 'ICESHEETS',
 'FIRE']
```

Now that we see what the toolbox can offer, we can find datasets for one ecv, e.g., WATERVAPOUR.

[2]:
```
from esa_climate_toolbox.core import list_ecv_datasets

list_ecv_datasets("WATERVAPOUR")
```

[2]:
```
[('esacci.WATERVAPOUR.day.L3S.TCWV.multi-sensor.multi-platform.TCWV_land_005deg.3-2.r1',
  'esa-cci'),
 ('esacci.WATERVAPOUR.day.L3S.TCWV.multi-sensor.multi-platform.TCWV_land_05deg.3-2.r1',
  'esa-cci'),
 ('esacci.WATERVAPOUR.mon.L3S.TCWV.multi-sensor.multi-platform.TCWV_land_005deg.3-2.r1',
  'esa-cci'),
 ('esacci.WATERVAPOUR.mon.L3S.TCWV.multi-sensor.multi-platform.TCWV_land_05deg.3-2.r1',
  'esa-cci')]
```

The result consists of a list of tuples, where the first value is a data id and the second value the name of the store that provides access to the data. For information on how to access the stores specifically, using the xcube store framework, see notebooks 2 to 5. Here, we continue with listing all the stores that are registered in the system.

[3]:
```
from esa_climate_toolbox.core import list_stores

list_stores()
```

[3]:
```
['esa-cci', 'esa-cci-kc', 'esa-cci-zarr', 'local']
```

You will see default stores (and maybe some you have added yourself). You can also ask for a store by asking for a specific dataset id.

[4]:
```
from esa_climate_toolbox.core import find_data_store

find_data_store(ds_id="esacci.WATERVAPOUR.mon.L3S.TCWV.multi-sensor.multi-platform.TCWV_
↪land_05deg.3-2.r1")
```

[4]:
```
('esa-cci',
 <esa_climate_toolbox.ds.dataaccess.CciCdcDataStore at 0x7f432ff48450>)
```

The result consists of the name of the store as well as the python object itself. We could also get it by just asking for its name.

[5]:
```
from esa_climate_toolbox.core import get_store

get_store("esa-cci")
```

[5]:
```
<esa_climate_toolbox.ds.dataaccess.CciCdcDataStore at 0x7f432ff48450>
```

We can also ask for the data of a store:

[6]:
```
from esa_climate_toolbox.core import list_datasets

list_datasets("esa-cci-zarr")
```

[6]:
```
['ESACCI-BIOMASS-L4-AGB-MERGED-100m-2010-2018-fv2.0.zarr',
 'ESACCI-BIOMASS-L4-AGB-MERGED-100m-2010-2020-fv4.0.zarr',
 'ESACCI-GHG-L2-CH4-SCIAMACHY-WFMD-2002-2011-fv1.zarr',
 'ESACCI-GHG-L2-CO2-OCO-2-FOCAL-2014-2021-v10.zarr',
 'ESACCI-GHG-L2-CO2-SCIAMACHY-WFMD-2002-2012-fv1.zarr',
 'ESACCI-ICESHEETS_Antarctica_GMB-2002-2016-v1.1.zarr',
 'ESACCI-ICESHEETS_Greenland_GMB-2003-2016-v1.1.zarr',
 'ESACCI-L3C_CLOUD-CLD_PRODUCTS-AVHRR_NOAA-1982-2016-fv3.0.zarr',
 'ESACCI-L3C_SNOW-SWE-1979-2018-fv1.0.zarr',
 'ESACCI-L3C_SNOW-SWE-1979-2020-fv2.0.zarr',
 'ESACCI-L4_GHRSST-SST-GMPE-GLOB_CDR2.0-1981-2016-v02.0-fv01.0.zarr',
 'ESACCI-LAKES-L3S-LK_PRODUCTS-MERGED-1992-09-fv2.0.1.zarr',
 'ESACCI-LC-L4-LCCS-Map-300m-P1Y-1992-2015-v2.0.7b.zarr',
 'ESACCI-LST-L3C-LST-MODISA-0.01deg_1DAILY_DAY-2002-2018-fv3.00.zarr',
 'ESACCI-LST-L3C-LST-MODISA-0.01deg_1DAILY_NIGHT-2002-2018-fv3.00.zarr',
 'ESACCI-LST-L3S-LST-IRCDR_-0.01deg_1DAILY_DAY-1995-2020-fv3.00.zarr',
 'ESACCI-LST-L3S-LST-IRCDR_-0.01deg_1DAILY_NIGHT-1995-2020-fv3.00.zarr',
 'ESACCI-LST-L3S-LST-IRCDR_-0.01deg_1MONTHLY_DAY-1995-2020-fv3.00.zarr',
 'ESACCI-LST-L3S-LST-IRCDR_-0.01deg_1MONTHLY_NIGHT-1995-2020-fv3.00.zarr',
 'ESACCI-OC-L3S-IOP-MERGED-1M_MONTHLY_4km_GEO_PML_OCx-QAA-1997-2022-fv6.0.zarr',
 'ESACCI-OC-L3S-IOP-MERGED-1M_MONTHLY_4km_GEO_PML_OCx_QAA-1997-2020-fv5.0.zarr',
 'ESACCI-OC-L3S-IOP-MERGED-1Y_YEARLY_4km_GEO_PML_OCx_QAA-1997-2022-fv6.0.zarr',
 'ESACCI-OC-L3S-IOP-MERGED-1Y_YEARLY_4km_GEO_PML_RRS-1997-2022-fv6.0.zarr',
 'ESACCI-OC-L3S-IOP-MERGED-8D_DAILY_4km_GEO_PML_OCx_QAA-1997-2022-fv6.0.zarr',
 'ESACCI-OC-L3S-OC_PRODUCTS-MERGED-1M_MONTHLY_4km_GEO_PML_OCx_QAA-1997-2022-fv6.0.zarr',
 'ESACCI-OC-L3S-OC_PRODUCTS-MERGED-1Y_YEARLY_4km_GEO_PML_OCx_QAA-1997-2022-fv6.0.zarr',
 'ESACCI-OC-L3S-RRS-MERGED-1M_MONTHLY_4km_GEO_PML_RRS-1997-2022-fv6.0.zarr',
 'ESACCI-PERMAFROST-L4-ALT-MODISLST-AREA4_PP-1997-2018-fv02.0.zarr',
 'ESACCI-SEAICE-L3C-SITHICK-RA2_ENVISAT-NH25KMEASE2-2002-2012-fv2.0.zarr',
 'ESACCI-SEAICE-L3C-SITHICK-SIRAL_CRYOSAT2-NH25KMEASE2-2010-2017-fv2.0.zarr',
 'ESACCI-SEAICE-L4-SICONC-AMSR_50.0kmEASE2-NH-2002-2017-fv2.1.zarr',
 'ESACCI-SEALEVEL-IND-MSLTR-MERGED-1993-2016-fv02.zarr',
 'ESACCI-SEALEVEL-L4-MSLA-MERGED-1993-2015-fv02.zarr',
 'ESACCI-SOILMOISTURE-L3S-SSMV-COMBINED-1978-2020-fv05.3.zarr',
 'ESACCI-SOILMOISTURE-L3S-SSMV-COMBINED-1978-2021-fv07.1.zarr']
```

Adding your own stores

You may also define your own stores. This can come in handy when you want to combine your own data with the data from the toolbox. You may add store to locally hosted data like this:

[7]:
```
from esa_climate_toolbox.core import add_local_store

add_local_store(
    root="/path/to/my/data/",
    store_id="my_real_local_store",
    includes="aero*.nc",
    read_only=True,
    persist=False
)
```

[7]:
```
'my_real_local_store'
```

The `root` specifies the path to the data that shall be served by the store. The `store_id` defines by which name the store is registered in the system. With `includes` you can specify a pattern to determine what data shall be served, `read_only` causes that the store does not permit write operations, and if you would set `persist` to true, the store will be saved and available in your next session. Of course, we can always also remove a store. Don't worry, this only deletes the registry entry. No real data is deleted.

[8]:
```
from esa_climate_toolbox.core import remove_store

remove_store("my_real_local_store")
```

Writing data to stores

When you have created a new dataset, you might want to save it somewhere. For that, you can use this command:

[9]:
```
from esa_climate_toolbox.core import write_data

# write_data(data, store_id="my_real_local_store")
```

We leave this open for now, as we don't want to actually write data now. Note that to write data, you need to define a store and make sure it is not read_only (see above). If you know you want your output in the same place, you can set a default output store:

[10]:
```
from esa_climate_toolbox.core import set_output_store

add_local_store(
    root="/path/to/my/data/",
    store_id="my_real_local_store",
    includes="aero*.nc",
    read_only=True,
    persist=False
)
set_output_store("my_real_local_store")
```

Let's check this is really the output store:

[11]:
```
from esa_climate_toolbox.core import get_output_store_id

get_output_store_id()
```

[11]:
```
'my_real_local_store'
```

Great! You are set to use these functions for making use of the toolbox. See the other notebooks to find out how to open datasets and use operations. Also, see the API for more detailed information on the functions described here: https://esa-climate-toolbox.readthedocs.io/en/latest/api_reference.html#datasets-and-datastores.

ESA CCI Toolbox Data Access

The aim of this notebook is to show how to query for all the data sets provided by the ESA CCI Toolbox. To run this Notebook, make sure the ESA CCI Toolbox is setup correctly.

The ESA CCI Toolbox is based on xcube. It provides dedicated stores to access data from the ESA Open Data Portal (ODP). To access the store, we can use the following import.

[1]:
```
from xcube.core.store import new_data_store
```

Now we can acces the ESA Open Data Portal store. We may request it by 'esa-cci'.

[2]:
```
cci_store = new_data_store('esa-cci')
```

We can check what types of data it provides.

[3]:
```
cci_store.get_data_types()
```

[3]:
```
('dataset', 'geodataframe')
```

All provided data are datasets. So, let's have a look what data sets are available.

[4]:
```
datasets = cci_store.get_data_ids()
list(datasets)
```

[4]:
```
['esacci.AEROSOL.5-days.L3C.AEX.GOMOS.Envisat.AERGOM.3-00.r1',
 'esacci.AEROSOL.climatology.L3.AAI.multi-sensor.multi-platform.MSAAI.1-7.r1',
 'esacci.AEROSOL.day.L3.AAI.multi-sensor.multi-platform.MSAAI.1-7.r1',
 'esacci.AEROSOL.day.L3C.AER_PRODUCTS.AATSR.Envisat.ADV.2-31.r1',
 'esacci.AEROSOL.day.L3C.AER_PRODUCTS.AATSR.Envisat.ORAC.04-01-.r1',
 'esacci.AEROSOL.day.L3C.AER_PRODUCTS.AATSR.Envisat.ORAC.04-01_seg-.r1',
 'esacci.AEROSOL.day.L3C.AER_PRODUCTS.AATSR.Envisat.SU.4-3.r1',
 'esacci.AEROSOL.day.L3C.AER_PRODUCTS.ATSR-2.ERS-2.ADV.2-31.r1',
 'esacci.AEROSOL.day.L3C.AER_PRODUCTS.ATSR-2.ERS-2.ORAC.04-01-.r1',
 'esacci.AEROSOL.day.L3C.AER_PRODUCTS.ATSR-2.ERS-2.ORAC.04-01_seg-.r1',
 'esacci.AEROSOL.day.L3C.AER_PRODUCTS.ATSR-2.ERS-2.SU.4-3.r1',
 'esacci.AEROSOL.day.L3C.AER_PRODUCTS.multi-sensor.multi-platform.AATSR-ENVISAT-ENS_
↪DAILY.v2-6.r1',
 'esacci.AEROSOL.day.L3C.AER_PRODUCTS.multi-sensor.multi-platform.ATSR2-ENVISAT-ENS_
↪DAILY.v2-6.r1',
 'esacci.AEROSOL.day.L3C.AOD.MERIS.Envisat.MERIS_ENVISAT.2-2.r1',
 'esacci.AEROSOL.mon.L3.AAI.multi-sensor.multi-platform.MSAAI.1-7.r1',
 'esacci.AEROSOL.mon.L3C.AER_PRODUCTS.AATSR.Envisat.ADV.2-31.r1',
 'esacci.AEROSOL.mon.L3C.AER_PRODUCTS.AATSR.Envisat.ORAC.04-01-.r1',
 'esacci.AEROSOL.mon.L3C.AER_PRODUCTS.AATSR.Envisat.ORAC.04-01_seg-.r1',
 'esacci.AEROSOL.mon.L3C.AER_PRODUCTS.AATSR.Envisat.SU.4-3.r1',
 'esacci.AEROSOL.mon.L3C.AER_PRODUCTS.ATSR-2.ERS-2.ADV.2-31.r1',
 'esacci.AEROSOL.mon.L3C.AER_PRODUCTS.ATSR-2.ERS-2.ORAC.04-01-.r1',
 'esacci.AEROSOL.mon.L3C.AER_PRODUCTS.ATSR-2.ERS-2.ORAC.04-01_seg-.r1',
 'esacci.AEROSOL.mon.L3C.AER_PRODUCTS.multi-sensor.multi-platform.AATSR-ENVISAT-ENS_
↪MONTHLY.v2-6.r1',
 'esacci.AEROSOL.mon.L3C.AER_PRODUCTS.multi-sensor.multi-platform.ATSR2-ENVISAT-ENS_
↪MONTHLY.v2-6.r1',
 'esacci.AEROSOL.mon.L3C.AOD.MERIS.Envisat.MERIS_ENVISAT.2-2.r1',
 'esacci.AEROSOL.satellite-orbit-frequency.L2P.AER_PRODUCTS.AATSR.Envisat.AATSR-ENVISAT-
```

(continues on next page)

(continued from previous page)

```
↪ENS.v2-6.r1',
 'esacci.AEROSOL.satellite-orbit-frequency.L2P.AER_PRODUCTS.AATSR.Envisat.ADV.2-31.r1',
 'esacci.AEROSOL.satellite-orbit-frequency.L2P.AER_PRODUCTS.AATSR.Envisat.ORAC.04-01.r1',
 'esacci.AEROSOL.satellite-orbit-frequency.L2P.AER_PRODUCTS.AATSR.Envisat.SU.4-3.r1',
 'esacci.AEROSOL.satellite-orbit-frequency.L2P.AER_PRODUCTS.ATSR-2.ERS-2.ADV.2-31.r1',
 'esacci.AEROSOL.satellite-orbit-frequency.L2P.AER_PRODUCTS.ATSR-2.ERS-2.ORAC.04-01.r1',
 'esacci.AEROSOL.satellite-orbit-frequency.L2P.AER_PRODUCTS.ATSR-2.ERS-2.SU.4-3.r1',
 'esacci.AEROSOL.satellite-orbit-frequency.L2P.AER_PRODUCTS.multi-sensor.multi-platform.
↪ATSR2-ENVISAT-ENS.v2-6.r1',
 'esacci.AEROSOL.satellite-orbit-frequency.L2P.AOD.MERIS.Envisat.MERIS_ENVISAT.2-2.r1',
 'esacci.AEROSOL.yr.L3C.AER_PRODUCTS.AATSR.Envisat.AATSR-ENVISAT-ENS_ANNUAL.v2-6.r1',
 'esacci.AEROSOL.yr.L3C.AER_PRODUCTS.ATSR-2.Envisat.ATSR2-ENVISAT-ENS_ANNUAL.v2-6.r1',
 'esacci.BIOMASS.yr.L4.AGB.multi-sensor.multi-platform.CHANGE.4-0.r1',
 'esacci.BIOMASS.yr.L4.AGB.multi-sensor.multi-platform.MERGED.2-0.r1',
 'esacci.BIOMASS.yr.L4.AGB.multi-sensor.multi-platform.MERGED.3-0.r1',
 'esacci.BIOMASS.yr.L4.AGB.multi-sensor.multi-platform.MERGED.4-0.r1',
 'esacci.CLOUD.mon.L3C.CLD_PRODUCTS.MODIS.Aqua.MODIS_AQUA.2-0.r1',
 'esacci.CLOUD.mon.L3C.CLD_PRODUCTS.MODIS.Terra.MODIS_TERRA.2-0.r1',
 'esacci.CLOUD.mon.L3C.CLD_PRODUCTS.multi-sensor.multi-platform.ATSR2-AATSR.3-0.r1',
 'esacci.CLOUD.mon.L3C.CLD_PRODUCTS.multi-sensor.multi-platform.AVHRR-AM.3-0.r1',
 'esacci.CLOUD.mon.L3C.CLD_PRODUCTS.multi-sensor.multi-platform.AVHRR-PM.3-0.r1',
 'esacci.CLOUD.mon.L3C.CLD_PRODUCTS.multi-sensor.multi-platform.MERIS-AATSR.2-0.r1',
 'esacci.FIRE.mon.L3S.BA.MODIS.Terra.MODIS_TERRA.v5-1.pixel',
 'esacci.FIRE.mon.L3S.BA.MSI-(Sentinel-2).Sentinel-2A.MSI.2-0.pixel',
 'esacci.FIRE.mon.L3S.BA.MSI-(Sentinel-2).Sentinel-2A.MSI.v1-1.pixel',
 'esacci.FIRE.mon.L3S.BA.multi-sensor.multi-platform.SYN.v1-0.pixel',
 'esacci.FIRE.mon.L3S.BA.multi-sensor.multi-platform.SYN.v1-1.pixel',
 'esacci.FIRE.mon.L4.BA.MODIS.Terra.MODIS_TERRA.v5-1.grid',
 'esacci.FIRE.mon.L4.BA.MSI-(Sentinel-2).Sentinel-2A.MSI.2-0.grid',
 'esacci.FIRE.mon.L4.BA.MSI-(Sentinel-2).Sentinel-2A.MSI.v1-1.grid',
 'esacci.FIRE.mon.L4.BA.multi-sensor.multi-platform.SYN.v1-0.grid',
 'esacci.FIRE.mon.L4.BA.multi-sensor.multi-platform.SYN.v1-1.grid',
 'esacci.GHG.satellite-orbit-frequency.L2.CH4.TANSO-FTS-2.GOSAT-2.SRFP.v2-0-2.r1',
 'esacci.GHG.satellite-orbit-frequency.L2.CH4.TANSO-FTS-2.GOSAT-2.SRPR.v2-0-2.r1',
 'esacci.GHG.satellite-orbit-frequency.L2.CH4.TROPOMI.Sentinel-5P.WFMD.v1-8.r1',
 'esacci.GHG.satellite-orbit-frequency.L2.CO2.OCO.OCO-2.FOCAL.v10-1.r1',
 'esacci.GHG.satellite-orbit-frequency.L2.CO2.TANSO-FTS-2.GOSAT-2.SRFP.v2-0-2.r1',
 'esacci.GHG.satellite-orbit-frequency.L2.CO2.TANSO-FTS.GOSAT.OCFP.v7-0-1.r1',
 'esacci.GHG.satellite-orbit-frequency.L2.CO2.multi-sensor.multi-platform.EMMA.v2-2a.r1',
 'esacci.GHG.satellite-orbit-frequency.L2.CO2.multi-sensor.multi-platform.EMMA.v2-2b.r1',
 'esacci.ICESHEETS.mon.IND.GMB.GRACE-instrument.GRACE.VARIOUS.1-3.greenland_gmb_time_
↪series',
 'esacci.ICESHEETS.unspecified.L4.IV.SAR-C-(Sentinel-1).multi-platform.UNSPECIFIED.1-1.
↪greenland_s1_250m_20150610_20170321_Helheim',
 'esacci.ICESHEETS.unspecified.L4.SEC.multi-sensor.multi-platform.UNSPECIFIED.0-1.
↪greenland_sec_saral_altika',
 'esacci.ICESHEETS.unspecified.Unspecified.CFL.multi-sensor.multi-platform.UNSPECIFIED.
↪v3-0.greenland',
 'esacci.ICESHEETS.unspecified.Unspecified.GLL.multi-sensor.multi-platform.UNSPECIFIED.
↪v1-3.greenland',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.AMI-SAR.ERS-1.UNSPECIFIED.1-1.greenland_
↪northern_drainage_basin_winter_1991_1992',
```

(continues on next page)

(continued from previous page)

```
 'esacci.ICESHEETS.unspecified.Unspecified.IV.AMI-SAR.ERS-2.UNSPECIFIED.1-1.greenland_
↪margin_1995_1996',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.MSI-(Sentinel-2).Sentinel-2A.UNSPECIFIED.1-
↪0.greenland_s2_50m_20160508_20160518_docker_smith',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.MSI-(Sentinel-2).multi-platform.
↪UNSPECIFIED.1-1.greenland_s2_50m_seasonal_20170501_20170829_Helheim',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.MSI-(Sentinel-2).multi-platform.
↪UNSPECIFIED.1-1.greenland_s2_50m_seasonal_20170501_20170914_Petermann',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.MSI-(Sentinel-2).multi-platform.
↪UNSPECIFIED.1-1.greenland_s2_50m_seasonal_20170603_20170908_Jakobshavn',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.MSI-(Sentinel-2).multi-platform.
↪UNSPECIFIED.1-1.greenland_s2_50m_seasonal_20170625_20170810_79Fjord',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.MSI-(Sentinel-2).multi-platform.
↪UNSPECIFIED.1-1.greenland_s2_50m_seasonal_20170625_20170810_Zachariae',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.MSI-(Sentinel-2).multi-platform.
↪UNSPECIFIED.1-1.greenland_s2_50m_seasonal_20170630_20170814_Hagen',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.MSI-(Sentinel-2).multi-platform.
↪UNSPECIFIED.1-1.greenland_s2_50m_seasonal_20170715_20170814_Upernavik',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.MSI-(Sentinel-2).multi-platform.
↪UNSPECIFIED.1-1.greenland_s2_50m_seasonal_20170721_20170820_Kangerdlugssuaq',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.MSI-(Sentinel-2).multi-platform.
↪UNSPECIFIED.1-1.greenland_s2_50m_timeseries_20170501_20170829_Helheim',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.MSI-(Sentinel-2).multi-platform.
↪UNSPECIFIED.1-1.greenland_s2_50m_timeseries_20170501_20170914_Petermann',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.MSI-(Sentinel-2).multi-platform.
↪UNSPECIFIED.1-1.greenland_s2_50m_timeseries_20170603_20170908_Jakobshavn',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.MSI-(Sentinel-2).multi-platform.
↪UNSPECIFIED.1-1.greenland_s2_50m_timeseries_20170625_20170810_79Fjord',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.MSI-(Sentinel-2).multi-platform.
↪UNSPECIFIED.1-1.greenland_s2_50m_timeseries_20170625_20170810_Zachariae',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.MSI-(Sentinel-2).multi-platform.
↪UNSPECIFIED.1-1.greenland_s2_50m_timeseries_20170630_20170814_Hagen',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.MSI-(Sentinel-2).multi-platform.
↪UNSPECIFIED.1-1.greenland_s2_50m_timeseries_20170715_20170814_Upernavik',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.MSI-(Sentinel-2).multi-platform.
↪UNSPECIFIED.1-1.greenland_s2_50m_timeseries_20170721_20170820_Kangerdlugssuaq',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.SAR-(RadarSat-2).RadarSat-2.UNSPECIFIED.1-
↪0.greenland_map_winter_2013_2014',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.SAR-2000.multi-platform.UNSPECIFIED.1-0.
↪greenland_csk_250m_timeseries_20120604_20141223_Jakobshavn',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.SAR-C-(Sentinel-1).Sentinel-1A.UNSPECIFIED.
↪1-0.greenland_map_winter_2014_2015',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.SAR-C-(Sentinel-1).Sentinel-1A.UNSPECIFIED.
↪1-2.greenland_map_winter_2015_2016',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.SAR-C-(Sentinel-1).multi-platform.
↪UNSPECIFIED.1-0.greenland_map_winter_2016_2017',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.SAR-C-(Sentinel-1).multi-platform.
↪UNSPECIFIED.1-0.greenland_map_winter_2017_2018',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.SAR-C-(Sentinel-1).multi-platform.
↪UNSPECIFIED.1-1.greenland_s1_250m_20141010_20170317_Upernavik',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.SAR-C-(Sentinel-1).multi-platform.
↪UNSPECIFIED.1-1.greenland_s1_250m_20141011_20170317_20150122_20170322_Hagen',
```

(continues on next page)

(continued from previous page)

```
 'esacci.ICESHEETS.unspecified.Unspecified.IV.SAR-C-(Sentinel-1).multi-platform.
↪UNSPECIFIED.1-1.greenland_s1_250m_20141011_20170317_Jakobshavn',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.SAR-C-(Sentinel-1).multi-platform.
↪UNSPECIFIED.1-1.greenland_s1_250m_20150118_20170321_Kangerlussuaq',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.SAR-C-(Sentinel-1).multi-platform.
↪UNSPECIFIED.1-1.greenland_s1_250m_20150122_20170319_Petermann',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.SAR-C-(Sentinel-1).multi-platform.
↪UNSPECIFIED.1-1.greenland_s1_250m_20150122_20170322_79-Fjord',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.SAR-C-(Sentinel-1).multi-platform.
↪UNSPECIFIED.1-1.greenland_s1_250m_20150124_20170322_Storstroemmen',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.SAR-C-(Sentinel-1).multi-platform.
↪UNSPECIFIED.1-1.greenland_s1_250m_20150126_20170322_Zachariae',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.multi-sensor.multi-platform.UNSPECIFIED.1-
↪0.greenland_timeseries_Kangerlussuaq',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.multi-sensor.multi-platform.UNSPECIFIED.1-
↪1.greenland_timeseries_2002_2010_Jakobshavn',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.multi-sensor.multi-platform.UNSPECIFIED.1-
↪1.greenland_timeseries_Hagen',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.multi-sensor.multi-platform.UNSPECIFIED.1-
↪1.greenland_timeseries_Helheim',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.multi-sensor.multi-platform.UNSPECIFIED.1-
↪1.greenland_timeseries_Petermann',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.multi-sensor.multi-platform.UNSPECIFIED.1-
↪1.greenland_timeseries_Storstrommen',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.multi-sensor.multi-platform.UNSPECIFIED.1-
↪1.greenland_timeseries_Zachariae_79Fjord',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.multi-sensor.multi-platform.UNSPECIFIED.1-
↪2.greenland_timeseries_1992_2010_Jakobshavn',
 'esacci.ICESHEETS.unspecified.Unspecified.IV.multi-sensor.multi-platform.UNSPECIFIED.1-
↪2.greenland_timeseries_Upernarvik',
 'esacci.ICESHEETS.yr.Unspecified.GMB.GRACE-instrument.GRACE.UNSPECIFIED.1-2.greenland_
↪gmb_mass_trends',
 'esacci.ICESHEETS.yr.Unspecified.GMB.GRACE-instrument.GRACE.UNSPECIFIED.1-2.greenland_
↪gmb_timeseries',
 'esacci.ICESHEETS.yr.Unspecified.GMB.GRACE-instrument.GRACE.UNSPECIFIED.1-3.greenland_
↪gmb_mass_trends',
 'esacci.ICESHEETS.yr.Unspecified.GMB.GRACE-instrument.GRACE.UNSPECIFIED.1-4.greenland_
↪gmb_mass_trends',
 'esacci.ICESHEETS.yr.Unspecified.GMB.GRACE-instrument.GRACE.UNSPECIFIED.1-4.greenland_
↪gmb_time_series',
 'esacci.ICESHEETS.yr.Unspecified.GMB.GRACE-instrument.GRACE.UNSPECIFIED.1-5.greenland_
↪gmb_mass_trends',
 'esacci.ICESHEETS.yr.Unspecified.GMB.GRACE-instrument.GRACE.UNSPECIFIED.1-5.greenland_
↪gmb_time_series',
 'esacci.ICESHEETS.yr.Unspecified.IV.PALSAR.ALOS.UNSPECIFIED.1-1.greenland_margin_2006_
↪2011',
 'esacci.ICESHEETS.yr.Unspecified.SEC.SIRAL.CryoSat-2.UNSPECIFIED.2-2.greenland_sec_
↪cryosat_2yr',
 'esacci.ICESHEETS.yr.Unspecified.SEC.SIRAL.CryoSat-2.UNSPECIFIED.2-2.greenland_sec_
↪cryosat_5yr',
 'esacci.ICESHEETS.yr.Unspecified.SEC.multi-sensor.multi-platform.UNSPECIFIED.1-2.r1',
 'esacci.LAKES.day.L3S.LK_PRODUCTS.multi-sensor.multi-platform.MERGED.v1-0.r1',
```

(continues on next page)

(continued from previous page)

```
'esacci.LAKES.day.L3S.LK_PRODUCTS.multi-sensor.multi-platform.MERGED.v1-1.r1',
'esacci.LAKES.day.L3S.LK_PRODUCTS.multi-sensor.multi-platform.MERGED.v2-0-2.r1',
'esacci.LC.13-yrs.L4.WB.ASAR.Envisat.Map.4-0.r1',
'esacci.LC.yr.L4.LCCS.multi-sensor.multi-platform.Map.2-0-7.r1',
'esacci.LC.yr.L4.PFT.Unspecified.Unspecified.Map.2-0-8.r1',
'esacci.LST.3-hours.L3S.LST.multi-sensor.multi-platform.IRMGP.1-00.r1',
'esacci.LST.day.L3C.LST.AATSR.Envisat.ATSR_3.3-00.DAY',
'esacci.LST.day.L3C.LST.AATSR.Envisat.ATSR_3.3-00.NIGHT',
'esacci.LST.day.L3C.LST.ATSR-2.ERS-2.ATSR_2.3-00.DAY',
'esacci.LST.day.L3C.LST.ATSR-2.ERS-2.ATSR_2.3-00.NIGHT',
'esacci.LST.day.L3C.LST.MODIS.Aqua.MODISA.3-00.DAY',
'esacci.LST.day.L3C.LST.MODIS.Aqua.MODISA.3-00.NIGHT',
'esacci.LST.day.L3C.LST.MODIS.Terra.MODIST.3-00.DAY',
'esacci.LST.day.L3C.LST.MODIS.Terra.MODIST.3-00.NIGHT',
'esacci.LST.day.L3C.LST.SLSTR.Sentinel-3A.SLSTRA.3-00.DAY',
'esacci.LST.day.L3C.LST.SLSTR.Sentinel-3A.SLSTRA.3-00.NIGHT',
'esacci.LST.day.L3C.LST.SLSTR.Sentinel-3B.SLSTRB.3-00.DAY',
'esacci.LST.day.L3C.LST.SLSTR.Sentinel-3B.SLSTRB.3-00.NIGHT',
'esacci.LST.day.L3C.LST.multi-sensor.multi-platform.SSMI_SSMIS.v2-33.ASC',
'esacci.LST.day.L3C.LST.multi-sensor.multi-platform.SSMI_SSMIS.v2-33.DES',
'esacci.LST.day.L3S.LST.multi-sensor.multi-platform.IRCDR.2-00.DAY',
'esacci.LST.day.L3S.LST.multi-sensor.multi-platform.IRCDR.2-00.NIGHT',
'esacci.LST.mon.L3C.LST.AATSR.Envisat.ATSR_3.3-00.DAY',
'esacci.LST.mon.L3C.LST.AATSR.Envisat.ATSR_3.3-00.NIGHT',
'esacci.LST.mon.L3C.LST.ATSR-2.ERS-2.ATSR_2.3-00.DAY',
'esacci.LST.mon.L3C.LST.ATSR-2.ERS-2.ATSR_2.3-00.NIGHT',
'esacci.LST.mon.L3C.LST.MODIS.Aqua.MODISA.3-00.DAY',
'esacci.LST.mon.L3C.LST.MODIS.Aqua.MODISA.3-00.NIGHT',
'esacci.LST.mon.L3C.LST.MODIS.Terra.MODIST.3-00.DAY',
'esacci.LST.mon.L3C.LST.MODIS.Terra.MODIST.3-00.NIGHT',
'esacci.LST.mon.L3C.LST.SLSTR.Sentinel-3A.SLSTRA.3-00.DAY',
'esacci.LST.mon.L3C.LST.SLSTR.Sentinel-3A.SLSTRA.3-00.NIGHT',
'esacci.LST.mon.L3C.LST.SLSTR.Sentinel-3B.SLSTRB.3-00.DAY',
'esacci.LST.mon.L3C.LST.SLSTR.Sentinel-3B.SLSTRB.3-00.NIGHT',
'esacci.LST.mon.L3C.LST.multi-sensor.multi-platform.SSMI_SSMIS.v2-33.ASC',
'esacci.LST.mon.L3C.LST.multi-sensor.multi-platform.SSMI_SSMIS.v2-33.DES',
'esacci.LST.mon.L3S.LST.multi-sensor.multi-platform.IRCDR.2-00.DAY',
'esacci.LST.mon.L3S.LST.multi-sensor.multi-platform.IRCDR.2-00.NIGHT',
'esacci.LST.mon.L3S.LST.multi-sensor.multi-platform.IRMGP.1-00.00:00UTC',
'esacci.LST.mon.L3S.LST.multi-sensor.multi-platform.IRMGP.1-00.03:00UTC',
'esacci.LST.mon.L3S.LST.multi-sensor.multi-platform.IRMGP.1-00.06:00UTC',
'esacci.LST.mon.L3S.LST.multi-sensor.multi-platform.IRMGP.1-00.09:00UTC',
'esacci.LST.mon.L3S.LST.multi-sensor.multi-platform.IRMGP.1-00.12:00UTC',
'esacci.LST.mon.L3S.LST.multi-sensor.multi-platform.IRMGP.1-00.15:00UTC',
'esacci.LST.mon.L3S.LST.multi-sensor.multi-platform.IRMGP.1-00.18:00UTC',
'esacci.LST.mon.L3S.LST.multi-sensor.multi-platform.IRMGP.1-00.21:00UTC',
'esacci.LST.yr.L3C.LST.multi-sensor.multi-platform.SSMI_SSMIS.v2-33.ASC',
'esacci.LST.yr.L3C.LST.multi-sensor.multi-platform.SSMI_SSMIS.v2-33.DES',
'esacci.OC.5-days.L3S.CHLOR_A.multi-sensor.multi-platform.MERGED.6-0.geographic',
'esacci.OC.5-days.L3S.CHLOR_A.multi-sensor.multi-platform.MERGED.6-0.sinusoidal',
'esacci.OC.5-days.L3S.IOP.multi-sensor.multi-platform.MERGED.6-0.geographic',
'esacci.OC.5-days.L3S.IOP.multi-sensor.multi-platform.MERGED.6-0.sinusoidal',
```

(continues on next page)

(continued from previous page)

```
'esacci.OC.5-days.L3S.K_490.multi-sensor.multi-platform.MERGED.6-0.geographic',
'esacci.OC.5-days.L3S.K_490.multi-sensor.multi-platform.MERGED.6-0.sinusoidal',
'esacci.OC.5-days.L3S.OC_PRODUCTS.multi-sensor.multi-platform.MERGED.6-0.geographic',
'esacci.OC.5-days.L3S.OC_PRODUCTS.multi-sensor.multi-platform.MERGED.6-0.sinusoidal',
'esacci.OC.5-days.L3S.RRS.multi-sensor.multi-platform.MERGED.6-0.geographic',
'esacci.OC.5-days.L3S.RRS.multi-sensor.multi-platform.MERGED.6-0.sinusoidal',
'esacci.OC.8-days.L3S.CHLOR_A.multi-sensor.multi-platform.MERGED.6-0.geographic',
'esacci.OC.8-days.L3S.CHLOR_A.multi-sensor.multi-platform.MERGED.6-0.sinusoidal',
'esacci.OC.8-days.L3S.IOP.multi-sensor.multi-platform.MERGED.6-0.geographic',
'esacci.OC.8-days.L3S.IOP.multi-sensor.multi-platform.MERGED.6-0.sinusoidal',
'esacci.OC.8-days.L3S.K_490.multi-sensor.multi-platform.MERGED.6-0.geographic',
'esacci.OC.8-days.L3S.K_490.multi-sensor.multi-platform.MERGED.6-0.sinusoidal',
'esacci.OC.8-days.L3S.OC_PRODUCTS.multi-sensor.multi-platform.MERGED.6-0.geographic',
'esacci.OC.8-days.L3S.OC_PRODUCTS.multi-sensor.multi-platform.MERGED.6-0.sinusoidal',
'esacci.OC.8-days.L3S.RRS.multi-sensor.multi-platform.MERGED.6-0.geographic',
'esacci.OC.8-days.L3S.RRS.multi-sensor.multi-platform.MERGED.6-0.sinusoidal',
'esacci.OC.day.L3S.CHLOR_A.multi-sensor.multi-platform.MERGED.6-0.geographic',
'esacci.OC.day.L3S.CHLOR_A.multi-sensor.multi-platform.MERGED.6-0.sinusoidal',
'esacci.OC.day.L3S.IOP.multi-sensor.multi-platform.MERGED.6-0.geographic',
'esacci.OC.day.L3S.IOP.multi-sensor.multi-platform.MERGED.6-0.sinusoidal',
'esacci.OC.day.L3S.K_490.multi-sensor.multi-platform.MERGED.6-0.geographic',
'esacci.OC.day.L3S.K_490.multi-sensor.multi-platform.MERGED.6-0.sinusoidal',
'esacci.OC.day.L3S.OC_PRODUCTS.multi-sensor.multi-platform.MERGED.6-0.geographic',
'esacci.OC.day.L3S.OC_PRODUCTS.multi-sensor.multi-platform.MERGED.6-0.sinusoidal',
'esacci.OC.day.L3S.RRS.multi-sensor.multi-platform.MERGED.6-0.geographic',
'esacci.OC.day.L3S.RRS.multi-sensor.multi-platform.MERGED.6-0.sinusoidal',
'esacci.OC.mon.L3S.CHLOR_A.multi-sensor.multi-platform.MERGED.6-0.geographic',
'esacci.OC.mon.L3S.CHLOR_A.multi-sensor.multi-platform.MERGED.6-0.sinusoidal',
'esacci.OC.mon.L3S.IOP.multi-sensor.multi-platform.MERGED.6-0.geographic',
'esacci.OC.mon.L3S.IOP.multi-sensor.multi-platform.MERGED.6-0.sinusoidal',
'esacci.OC.mon.L3S.K_490.multi-sensor.multi-platform.MERGED.6-0.geographic',
'esacci.OC.mon.L3S.K_490.multi-sensor.multi-platform.MERGED.6-0.sinusoidal',
'esacci.OC.mon.L3S.OC_PRODUCTS.multi-sensor.multi-platform.MERGED.6-0.geographic',
'esacci.OC.mon.L3S.OC_PRODUCTS.multi-sensor.multi-platform.MERGED.6-0.sinusoidal',
'esacci.OC.mon.L3S.RRS.multi-sensor.multi-platform.MERGED.6-0.geographic',
'esacci.OC.mon.L3S.RRS.multi-sensor.multi-platform.MERGED.6-0.sinusoidal',
'esacci.OC.yr.L3S.CHLOR_A.multi-sensor.multi-platform.MERGED.6-0.geographic',
'esacci.OC.yr.L3S.CHLOR_A.multi-sensor.multi-platform.MERGED.6-0.sinusoidal',
'esacci.OC.yr.L3S.IOP.multi-sensor.multi-platform.MERGED.6-0.geographic',
'esacci.OC.yr.L3S.IOP.multi-sensor.multi-platform.MERGED.6-0.sinusoidal',
'esacci.OC.yr.L3S.K_490.multi-sensor.multi-platform.MERGED.6-0.geographic',
'esacci.OC.yr.L3S.K_490.multi-sensor.multi-platform.MERGED.6-0.sinusoidal',
'esacci.OC.yr.L3S.OC_PRODUCTS.multi-sensor.multi-platform.MERGED.6-0.geographic',
'esacci.OC.yr.L3S.OC_PRODUCTS.multi-sensor.multi-platform.MERGED.6-0.sinusoidal',
'esacci.OC.yr.L3S.RRS.multi-sensor.multi-platform.MERGED.6-0.geographic',
'esacci.OC.yr.L3S.RRS.multi-sensor.multi-platform.MERGED.6-0.sinusoidal',
'esacci.OZONE.day.L3S.TC.multi-sensor.multi-platform.MERGED.fv0100.r1',
'esacci.OZONE.mon.L3.LP.GOMOS.Envisat.GOMOS_ENVISAT.v0001.r1',
'esacci.OZONE.mon.L3.LP.MIPAS.Envisat.MIPAS_ENVISAT.v0001.r1',
'esacci.OZONE.mon.L3.LP.OSIRIS.ODIN.OSIRIS_ODIN.v0001.r1',
'esacci.OZONE.mon.L3.LP.SCIAMACHY.Envisat.SCIAMACHY_ENVISAT.v0001.r1',
'esacci.OZONE.mon.L3.LP.SMR.ODIN.MZM.v0001.r1',
```

(continues on next page)

(continued from previous page)

```
 'esacci.OZONE.mon.L3.LP.SMR.ODIN.SMR_ODIN.v0001.r1',
 'esacci.OZONE.mon.L3.NP.multi-sensor.multi-platform.MERGED.fv0002.r1',
 'esacci.PERMAFROST.yr.L4.ALT.multi-sensor.multi-platform.ERA5_MODISLST_BIASCORRECTED.03-
↪0.r1',
 'esacci.PERMAFROST.yr.L4.ALT.multi-sensor.multi-platform.MODISLST_CRYOGRID.03-0.r1',
 'esacci.PERMAFROST.yr.L4.GTD.multi-sensor.multi-platform.ERA5_MODISLST_BIASCORRECTED.03-
↪0.r1',
 'esacci.PERMAFROST.yr.L4.GTD.multi-sensor.multi-platform.MODISLST_CRYOGRID.03-0.r1',
 'esacci.PERMAFROST.yr.L4.PFR.multi-sensor.multi-platform.ERA5_MODISLST_BIASCORRECTED.03-
↪0.r1',
 'esacci.PERMAFROST.yr.L4.PFR.multi-sensor.multi-platform.MODISLST_CRYOGRID.03-0.r1',
 'esacci.SEAICE.day.L3C.SICONC.ESMR-(Nimbus-5).Nimbus-5.NIMBUS5_ESMR-EASE2_NH.1-0.NH',
 'esacci.SEAICE.day.L3C.SICONC.ESMR-(Nimbus-5).Nimbus-5.NIMBUS5_ESMR-EASE2_SH.1-0.SH',
 'esacci.SEAICE.day.L4.SICONC.multi-sensor.multi-platform.AMSR_25kmEASE2.2-1.NH',
 'esacci.SEAICE.day.L4.SICONC.multi-sensor.multi-platform.AMSR_25kmEASE2.2-1.SH',
 'esacci.SEAICE.day.L4.SICONC.multi-sensor.multi-platform.AMSR_50kmEASE2.2-1.NH',
 'esacci.SEAICE.day.L4.SICONC.multi-sensor.multi-platform.AMSR_50kmEASE2.2-1.SH',
 'esacci.SEAICE.day.L4.SICONC.multi-sensor.multi-platform.RE_SSMI_12-5kmEASE2-NH.3-0.NH',
 'esacci.SEAICE.day.L4.SICONC.multi-sensor.multi-platform.RE_SSMI_12-5kmEASE2-SH.3-0.SH',
 'esacci.SEAICE.mon.L3C.SITHICK.RA-2.Envisat.NH25KMEASE2.2-0.r1',
 'esacci.SEAICE.mon.L3C.SITHICK.RA-2.Envisat.SH50KMEASE2.2-0.r1',
 'esacci.SEAICE.mon.L3C.SITHICK.SIRAL.CryoSat-2.NH25KMEASE2.2-0.r1',
 'esacci.SEAICE.mon.L3C.SITHICK.SIRAL.CryoSat-2.SH50KMEASE2.2-0.r1',
 'esacci.SEALEVEL.mon.IND.MSL.multi-sensor.multi-platform.MERGED.2-0.r1',
 'esacci.SEALEVEL.mon.IND.MSLAMPH.multi-sensor.multi-platform.MERGED.2-0.r1',
 'esacci.SEALEVEL.mon.IND.MSLTR.multi-sensor.multi-platform.MERGED.2-0.r1',
 'esacci.SEALEVEL.mon.IND.MSLTR.multi-sensor.multi-platform.MERGED.2-2.ASA',
 'esacci.SEALEVEL.mon.IND.MSLTR.multi-sensor.multi-platform.MERGED.2-2.BENGUELA',
 'esacci.SEALEVEL.mon.IND.MSLTR.multi-sensor.multi-platform.MERGED.2-2.CARIBBEAN',
 'esacci.SEALEVEL.mon.IND.MSLTR.multi-sensor.multi-platform.MERGED.2-2.GULFSTREAM',
 'esacci.SEALEVEL.mon.IND.MSLTR.multi-sensor.multi-platform.MERGED.2-2.HUMBOLDT',
 'esacci.SEALEVEL.mon.IND.MSLTR.multi-sensor.multi-platform.MERGED.2-2.MED_SEA',
 'esacci.SEALEVEL.mon.IND.MSLTR.multi-sensor.multi-platform.MERGED.2-2.NE_ATL',
 'esacci.SEALEVEL.mon.IND.MSLTR.multi-sensor.multi-platform.MERGED.2-2.N_INDIAN',
 'esacci.SEALEVEL.mon.IND.MSLTR.multi-sensor.multi-platform.MERGED.2-2.SE_AFRICA',
 'esacci.SEALEVEL.mon.IND.MSLTR.multi-sensor.multi-platform.MERGED.2-2.SE_ASIA',
 'esacci.SEALEVEL.mon.IND.MSLTR.multi-sensor.multi-platform.MERGED.2-2.S_AUSTRALIA',
 'esacci.SEALEVEL.mon.IND.MSLTR.multi-sensor.multi-platform.MERGED.2-2.WAFRICA',
 'esacci.SEALEVEL.mon.IND.MSLTR.multi-sensor.multi-platform.MERGED.2-2.r1',
 'esacci.SEALEVEL.mon.L4.MSLA.multi-sensor.multi-platform.MERGED.2-0.r1',
 'esacci.SEASTATE.mon.L4.SWH.multi-sensor.multi-platform.MULTI_1M.3-0.r1',
 'esacci.SEASURFACESALINITY.15-days.L4.SSS.multi-sensor.multi-platform.GLOBAL-MERGED_OI_
↪Monthly_CENTRED_15Day_0-25deg.4-41.r1',
 'esacci.SEASURFACESALINITY.15-days.L4.SSS.multi-sensor.multi-platform.MERGED_OI_Monthly_
↪CENTRED_15Day_25km.2-31.r1',
 'esacci.SEASURFACESALINITY.15-days.L4.SSS.multi-sensor.multi-platform.MERGED_OI_Monthly_
↪CENTRED_15Day_25km.3-21.r1',
 'esacci.SEASURFACESALINITY.15-days.L4.SSS.multi-sensor.multi-platform.POLAR-MERGED_OI_
↪7DAY_RUNNINGMEAN_DAILY_25kmEASE2-SH.4-41.r1',
 'esacci.SEASURFACESALINITY.15-days.L4.SSS.multi-sensor.multi-platform.POLAR-MERGED_OI_
↪Monthly_CENTRED_15Day_25kmEASE2-NH.4-41.r1',
 'esacci.SEASURFACESALINITY.15-days.L4.SSS.multi-sensor.multi-platform.POLAR-MERGED_OI_
```

(continues on next page)

(continued from previous page)

```
↪Monthly_CENTRED_15Day_25kmEASE2-SH.4-41.r1',
 'esacci.SEASURFACESALINITY.day.L4.SSS.multi-sensor.multi-platform.GLOBAL-MERGED_OI_7DAY_
↪RUNNINGMEAN_DAILY_0-25deg.4-41.r1',
 'esacci.SEASURFACESALINITY.day.L4.SSS.multi-sensor.multi-platform.MERGED_OI_7DAY_
↪RUNNINGMEAN_DAILY_25km.2-31.r1',
 'esacci.SEASURFACESALINITY.day.L4.SSS.multi-sensor.multi-platform.MERGED_OI_7DAY_
↪RUNNINGMEAN_DAILY_25km.3-21.r1',
 'esacci.SEASURFACESALINITY.day.L4.SSS.multi-sensor.multi-platform.POLAR-MERGED_OI_7DAY_
↪RUNNINGMEAN_DAILY_25kmEASE2-NH.4-41.r1',
 'esacci.SNOW.day.L3C.SCFG.AATSR.Envisat.AATSR_ENVISAT.v1-0.r1',
 'esacci.SNOW.day.L3C.SCFG.ATSR-2.ERS-2.ATSR2_ERS2.v1-0.r1',
 'esacci.SNOW.day.L3C.SCFG.MODIS.Terra.MODIS_TERRA.2-0.r1',
 'esacci.SNOW.day.L3C.SCFG.multi-sensor.multi-platform.AVHRR_MERGED.2-0.r1',
 'esacci.SNOW.day.L3C.SCFV.AATSR.Envisat.AATSR_ENVISAT.v1-0.r1',
 'esacci.SNOW.day.L3C.SCFV.ATSR-2.ERS-2.ATSR2_ERS2.v1-0.r1',
 'esacci.SNOW.day.L3C.SCFV.MODIS.Terra.MODIS_TERRA.2-0.r1',
 'esacci.SNOW.day.L3C.SCFV.multi-sensor.multi-platform.AVHRR_MERGED.2-0.r1',
 'esacci.SNOW.day.L3C.SWE.multi-sensor.multi-platform.MERGED.2-0.r1',
 'esacci.SNOW.day.L3S.SCFG.multi-sensor.multi-platform.MERGED.1-0.r1',
 'esacci.SOILMOISTURE.day.L3S.SSMS.multi-sensor.multi-platform.ACTIVE.05-2.r1',
 'esacci.SOILMOISTURE.day.L3S.SSMS.multi-sensor.multi-platform.ACTIVE.v05-3.r1',
 'esacci.SOILMOISTURE.day.L3S.SSMS.multi-sensor.multi-platform.ACTIVE.v06-1.r1',
 'esacci.SOILMOISTURE.day.L3S.SSMS.multi-sensor.multi-platform.ACTIVE.v06-2.r1',
 'esacci.SOILMOISTURE.day.L3S.SSMS.multi-sensor.multi-platform.ACTIVE.v07-1.r1',
 'esacci.SOILMOISTURE.day.L3S.SSMS.multi-sensor.multi-platform.ACTIVE.v08-1.r1',
 'esacci.SOILMOISTURE.day.L3S.SSMV.multi-sensor.multi-platform.COMBINED.05-2.r1',
 'esacci.SOILMOISTURE.day.L3S.SSMV.multi-sensor.multi-platform.COMBINED.v05-3.r1',
 'esacci.SOILMOISTURE.day.L3S.SSMV.multi-sensor.multi-platform.COMBINED.v06-1.r1',
 'esacci.SOILMOISTURE.day.L3S.SSMV.multi-sensor.multi-platform.COMBINED.v06-2.r1',
 'esacci.SOILMOISTURE.day.L3S.SSMV.multi-sensor.multi-platform.COMBINED.v07-1.r1',
 'esacci.SOILMOISTURE.day.L3S.SSMV.multi-sensor.multi-platform.COMBINED.v08-1.r1',
 'esacci.SOILMOISTURE.day.L3S.SSMV.multi-sensor.multi-platform.COMBINED_ADJUSTED.v07-1.r1
↪',
 'esacci.SOILMOISTURE.day.L3S.SSMV.multi-sensor.multi-platform.PASSIVE.05-2.r1',
 'esacci.SOILMOISTURE.day.L3S.SSMV.multi-sensor.multi-platform.PASSIVE.v05-3.r1',
 'esacci.SOILMOISTURE.day.L3S.SSMV.multi-sensor.multi-platform.PASSIVE.v06-1.r1',
 'esacci.SOILMOISTURE.day.L3S.SSMV.multi-sensor.multi-platform.PASSIVE.v06-2.r1',
 'esacci.SOILMOISTURE.day.L3S.SSMV.multi-sensor.multi-platform.PASSIVE.v07-1.r1',
 'esacci.SOILMOISTURE.day.L3S.SSMV.multi-sensor.multi-platform.PASSIVE.v08-1.r1',
 'esacci.SST.day.L3C.SSTskin.AATSR.Envisat.AATSR.2-1.day',
 'esacci.SST.day.L3C.SSTskin.AATSR.Envisat.AATSR.2-1.night',
 'esacci.SST.day.L3C.SSTskin.ATSR-2.ERS-2.ATSR2.2-1.day',
 'esacci.SST.day.L3C.SSTskin.ATSR-2.ERS-2.ATSR2.2-1.night',
 'esacci.SST.day.L3C.SSTskin.ATSR.ERS-1.ATSR1.2-1.day',
 'esacci.SST.day.L3C.SSTskin.ATSR.ERS-1.ATSR1.2-1.night',
 'esacci.SST.day.L3C.SSTskin.AVHRR-2.NOAA-11.AVHRR11_G.2-1.day',
 'esacci.SST.day.L3C.SSTskin.AVHRR-2.NOAA-11.AVHRR11_G.2-1.night',
 'esacci.SST.day.L3C.SSTskin.AVHRR-2.NOAA-12.AVHRR12_G.2-1.day',
 'esacci.SST.day.L3C.SSTskin.AVHRR-2.NOAA-12.AVHRR12_G.2-1.night',
 'esacci.SST.day.L3C.SSTskin.AVHRR-2.NOAA-14.AVHRR14_G.2-1.day',
 'esacci.SST.day.L3C.SSTskin.AVHRR-2.NOAA-14.AVHRR14_G.2-1.night',
 'esacci.SST.day.L3C.SSTskin.AVHRR-2.NOAA-7.AVHRR07_G.2-1.day',
```

(continues on next page)

(continued from previous page)

```
'esacci.SST.day.L3C.SSTskin.AVHRR-2.NOAA-7.AVHRR07_G.2-1.night',
'esacci.SST.day.L3C.SSTskin.AVHRR-2.NOAA-9.AVHRR09_G.2-1.day',
'esacci.SST.day.L3C.SSTskin.AVHRR-2.NOAA-9.AVHRR09_G.2-1.night',
'esacci.SST.day.L3C.SSTskin.AVHRR-3.Metop-A.AVHRRMTA_G.2-1.day',
'esacci.SST.day.L3C.SSTskin.AVHRR-3.Metop-A.AVHRRMTA_G.2-1.night',
'esacci.SST.day.L3C.SSTskin.AVHRR-3.NOAA-15.AVHRR15_G.2-1.day',
'esacci.SST.day.L3C.SSTskin.AVHRR-3.NOAA-15.AVHRR15_G.2-1.night',
'esacci.SST.day.L3C.SSTskin.AVHRR-3.NOAA-16.AVHRR16_G.2-1.day',
'esacci.SST.day.L3C.SSTskin.AVHRR-3.NOAA-16.AVHRR16_G.2-1.night',
'esacci.SST.day.L3C.SSTskin.AVHRR-3.NOAA-17.AVHRR17_G.2-1.day',
'esacci.SST.day.L3C.SSTskin.AVHRR-3.NOAA-17.AVHRR17_G.2-1.night',
'esacci.SST.day.L3C.SSTskin.AVHRR-3.NOAA-18.AVHRR18_G.2-1.day',
'esacci.SST.day.L3C.SSTskin.AVHRR-3.NOAA-18.AVHRR18_G.2-1.night',
'esacci.SST.day.L3C.SSTskin.AVHRR-3.NOAA-19.AVHRR19_G.2-1.day',
'esacci.SST.day.L3C.SSTskin.AVHRR-3.NOAA-19.AVHRR19_G.2-1.night',
'esacci.SST.day.L4.SSTdepth.multi-sensor.multi-platform.OSTIA.1-1.r1',
'esacci.SST.day.L4.SSTdepth.multi-sensor.multi-platform.OSTIA.2-1.anomaly',
'esacci.SST.day.L4.SSTdepth.multi-sensor.multi-platform.OSTIA.2-1.sst',
'esacci.SST.day.L4.SSTskin.Unspecified.Unspecified.GMPE.2-0.r1',
'esacci.SST.satellite-orbit-frequency.L2P.SSTskin.AATSR.Envisat.AATSR.2-1.r1',
'esacci.SST.satellite-orbit-frequency.L2P.SSTskin.ATSR-2.ERS-2.ATSR2.2-1.r1',
'esacci.SST.satellite-orbit-frequency.L2P.SSTskin.ATSR.ERS-1.ATSR1.2-1.r1',
'esacci.SST.satellite-orbit-frequency.L2P.SSTskin.AVHRR-2.NOAA-11.AVHRR11_G.2-1.r1',
'esacci.SST.satellite-orbit-frequency.L2P.SSTskin.AVHRR-2.NOAA-12.AVHRR12_G.2-1.r1',
'esacci.SST.satellite-orbit-frequency.L2P.SSTskin.AVHRR-2.NOAA-14.AVHRR14_G.2-1.r1',
'esacci.SST.satellite-orbit-frequency.L2P.SSTskin.AVHRR-2.NOAA-7.AVHRR07_G.2-1.r1',
'esacci.SST.satellite-orbit-frequency.L2P.SSTskin.AVHRR-2.NOAA-9.AVHRR09_G.2-1.r1',
'esacci.SST.satellite-orbit-frequency.L2P.SSTskin.AVHRR-3.Metop-A.AVHRRMTA_G.2-1.r1',
'esacci.SST.satellite-orbit-frequency.L2P.SSTskin.AVHRR-3.NOAA-15.AVHRR15_G.2-1.r1',
'esacci.SST.satellite-orbit-frequency.L2P.SSTskin.AVHRR-3.NOAA-16.AVHRR16_G.2-1.r1',
'esacci.SST.satellite-orbit-frequency.L2P.SSTskin.AVHRR-3.NOAA-17.AVHRR17_G.2-1.r1',
'esacci.SST.satellite-orbit-frequency.L2P.SSTskin.AVHRR-3.NOAA-18.AVHRR18_G.2-1.r1',
'esacci.SST.satellite-orbit-frequency.L2P.SSTskin.AVHRR-3.NOAA-19.AVHRR19_G.2-1.r1',
'esacci.SST.satellite-orbit-frequency.L3U.SSTskin.AATSR.Envisat.AATSR.2-1.r1',
'esacci.SST.satellite-orbit-frequency.L3U.SSTskin.ATSR-2.ERS-2.ATSR2.2-1.r1',
'esacci.SST.satellite-orbit-frequency.L3U.SSTskin.ATSR.ERS-1.ATSR1.2-1.r1',
'esacci.SST.satellite-orbit-frequency.L3U.SSTskin.AVHRR-2.NOAA-11.AVHRR11_G.2-1.r1',
'esacci.SST.satellite-orbit-frequency.L3U.SSTskin.AVHRR-2.NOAA-12.AVHRR12_G.2-1.r1',
'esacci.SST.satellite-orbit-frequency.L3U.SSTskin.AVHRR-2.NOAA-14.AVHRR14_G.2-1.r1',
'esacci.SST.satellite-orbit-frequency.L3U.SSTskin.AVHRR-2.NOAA-7.AVHRR07_G.2-1.r1',
'esacci.SST.satellite-orbit-frequency.L3U.SSTskin.AVHRR-2.NOAA-9.AVHRR09_G.2-1.r1',
'esacci.SST.satellite-orbit-frequency.L3U.SSTskin.AVHRR-3.Metop-A.AVHRRMTA_G.2-1.r1',
'esacci.SST.satellite-orbit-frequency.L3U.SSTskin.AVHRR-3.NOAA-15.AVHRR15_G.2-1.r1',
'esacci.SST.satellite-orbit-frequency.L3U.SSTskin.AVHRR-3.NOAA-16.AVHRR16_G.2-1.r1',
'esacci.SST.satellite-orbit-frequency.L3U.SSTskin.AVHRR-3.NOAA-17.AVHRR17_G.2-1.r1',
'esacci.SST.satellite-orbit-frequency.L3U.SSTskin.AVHRR-3.NOAA-18.AVHRR18_G.2-1.r1',
'esacci.SST.satellite-orbit-frequency.L3U.SSTskin.AVHRR-3.NOAA-19.AVHRR19_G.2-1.r1',
'esacci.VEGETATION.5-days.L3S.VP_PRODUCTS.VEGETATION.SPOT-5.MERGED.v1-0.r1',
'esacci.VEGETATION.5-days.L3S.VP_PRODUCTS.VEGETATION.multi-platform.MERGED.v1-0.r1',
'esacci.VEGETATION.5-days.L3S.VP_PRODUCTS.Végétation-P.PROBA-V.MERGED.v1-0.r1',
'esacci.VEGETATION.5-days.L3S.VP_PRODUCTS.multi-sensor.multi-platform.MERGED.v1-0.r1',
'esacci.WATERVAPOUR.day.L3S.TCWV.multi-sensor.multi-platform.TCWV_land_005deg.3-2.r1',
```

(continues on next page)

(continued from previous page)

```
 'esacci.WATERVAPOUR.day.L3S.TCWV.multi-sensor.multi-platform.TCWV_land_05deg.3-2.r1',
 'esacci.WATERVAPOUR.mon.L3S.TCWV.multi-sensor.multi-platform.TCWV_land_005deg.3-2.r1',
 'esacci.WATERVAPOUR.mon.L3S.TCWV.multi-sensor.multi-platform.TCWV_land_05deg.3-2.r1',
 'esacci.GHG.satellite-orbit-frequency.L2.CH4.SCIAMACHY.Envisat.IMAP.v7-2.r1',
 'esacci.GHG.satellite-orbit-frequency.L2.CH4.SCIAMACHY.Envisat.WFMD.v4-0.r1',
 'esacci.GHG.satellite-orbit-frequency.L2.CH4.TANSO-FTS.GOSAT.EMMA.ch4_v1-2.r1',
 'esacci.GHG.satellite-orbit-frequency.L2.CH4.TANSO-FTS.GOSAT.OCFP.v2-1.r1',
 'esacci.GHG.satellite-orbit-frequency.L2.CH4.TANSO-FTS.GOSAT.OCPR.v7-0.r1',
 'esacci.GHG.satellite-orbit-frequency.L2.CH4.TANSO-FTS.GOSAT.SRFP.v2-3-8.r1',
 'esacci.GHG.satellite-orbit-frequency.L2.CH4.TANSO-FTS.GOSAT.SRPR.v2-3-8.r1',
 'esacci.GHG.satellite-orbit-frequency.L2.CO2.SCIAMACHY.Envisat.BESD.v02-01-02.r1',
 'esacci.GHG.satellite-orbit-frequency.L2.CO2.SCIAMACHY.Envisat.WFMD.v4-0.r1',
 'esacci.GHG.satellite-orbit-frequency.L2.CO2.TANSO-FTS.GOSAT.EMMA.v2-2c.r1',
 'esacci.GHG.satellite-orbit-frequency.L2.CO2.TANSO-FTS.GOSAT.SRFP.v2-3-8.r1',
 'esacci.SEAICE.satellite-orbit-frequency.L2P.SITHICK.RA-2.Envisat.NH.2-0.r1',
 'esacci.SEAICE.satellite-orbit-frequency.L2P.SITHICK.RA-2.Envisat.SH.2-0.r1',
 'esacci.SEAICE.satellite-orbit-frequency.L2P.SITHICK.SIRAL.CryoSat-2.NH.2-0.r1',
 'esacci.SEAICE.satellite-orbit-frequency.L2P.SITHICK.SIRAL.CryoSat-2.SH.2-0.r1',
 'esacci.SEALEVEL.satellite-orbit-frequency.L1.UNSPECIFIED.AltiKa.SARAL.UNSPECIFIED.v2-0.
↪r1',
 'esacci.SEALEVEL.satellite-orbit-frequency.L1.UNSPECIFIED.GFO-RA.GFO.UNSPECIFIED.v2-0.r1
↪',
 'esacci.SEALEVEL.satellite-orbit-frequency.L1.UNSPECIFIED.Poseidon-2.Jason-1.
↪UNSPECIFIED.v2-0.r1',
 'esacci.SEALEVEL.satellite-orbit-frequency.L1.UNSPECIFIED.Poseidon-3.Jason-2.
↪UNSPECIFIED.v2-0.r1',
 'esacci.SEALEVEL.satellite-orbit-frequency.L1.UNSPECIFIED.RA-2.Envisat.UNSPECIFIED.v2-0.
↪r1',
 'esacci.SEALEVEL.satellite-orbit-frequency.L1.UNSPECIFIED.RA.ERS-1.UNSPECIFIED.v2-0.r1',
 'esacci.SEALEVEL.satellite-orbit-frequency.L1.UNSPECIFIED.RA.ERS-2.UNSPECIFIED.v2-0.r1',
 'esacci.SEALEVEL.satellite-orbit-frequency.L1.UNSPECIFIED.SIRAL.CryoSat-2.UNSPECIFIED.
↪v2-0.r1',
 'esacci.SEALEVEL.satellite-orbit-frequency.L1.UNSPECIFIED.SSALT.Topex-Poseidon.
↪UNSPECIFIED.v2-0.r1']
```

This might have been a bit much. In case you are looking for particular data sets, you can search for them. You can list which search options are available:

[5]:
```
cci_store.get_search_params_schema()
```

[5]:
```
<xcube.util.jsonschema.JsonObjectSchema at 0x7fc53f88e590>
```

The parameters are listed under *properties*. *start_date*, *end_date* and *bbox* are standard search parameters that are also used by other stores. *cci_attrs* lists additional parameters that are specific for the cci store (again, listed under *properties*). Of these, let's use *ecv* and *frequency*.

[6]:
```
cci_attrs = dict(
    ecv='AEROSOL',
    frequency='month'
)

[descriptor.data_id for descriptor in cci_store.search_data(cci_attrs=cci_attrs)]
```

```
---------------------------------------------------------------------------
ConnectionAbortedError                    Traceback (most recent call last)
File ~/miniconda3/envs/ect/lib/python3.11/site-packages/aiohttp/connector.py:980, in␣
↪TCPConnector._wrap_create_connection(self, req, timeout, client_error, *args, **kwargs)
    979     async with ceil_timeout(timeout.sock_connect):
--> 980         return await self._loop.create_connection(*args, **kwargs)  # type:␣
↪ignore[return-value]  # noqa
    981 except cert_errors as exc:

File ~/miniconda3/envs/ect/lib/python3.11/asyncio/base_events.py:1112, in BaseEventLoop.
↪create_connection(self, protocol_factory, host, port, ssl, family, proto, flags, sock,␣
↪local_addr, server_hostname, ssl_handshake_timeout, ssl_shutdown_timeout, happy_
↪eyeballs_delay, interleave)
   1109         raise ValueError(
   1110             f'A Stream Socket was expected, got {sock!r}')
-> 1112 transport, protocol = await self._create_connection_transport(
   1113     sock, protocol_factory, ssl, server_hostname,
   1114     ssl_handshake_timeout=ssl_handshake_timeout,
   1115     ssl_shutdown_timeout=ssl_shutdown_timeout)
   1116 if self._debug:
   1117     # Get the socket from the transport because SSL transport closes
   1118     # the old socket and creates a new SSL socket

File ~/miniconda3/envs/ect/lib/python3.11/asyncio/base_events.py:1145, in BaseEventLoop._
↪create_connection_transport(self, sock, protocol_factory, ssl, server_hostname, server_
↪side, ssl_handshake_timeout, ssl_shutdown_timeout)
   1144 try:
-> 1145     await waiter
   1146 except:

File ~/miniconda3/envs/ect/lib/python3.11/asyncio/futures.py:287, in Future.__await__
↪(self)
    286     self._asyncio_future_blocking = True
--> 287     yield self  # This tells Task to wait for completion.
    288 if not self.done():

File ~/miniconda3/envs/ect/lib/python3.11/asyncio/tasks.py:349, in Task.__wakeup(self,␣
↪future)
    348 try:
--> 349     future.result()
    350 except BaseException as exc:
    351     # This may also be a cancellation.

File ~/miniconda3/envs/ect/lib/python3.11/asyncio/futures.py:203, in Future.result(self)
    202 if self._exception is not None:
--> 203     raise self._exception.with_traceback(self._exception_tb)
    204 return self._result

ConnectionAbortedError: SSL handshake is taking longer than 60.0 seconds: aborting the␣
↪connection

The above exception was the direct cause of the following exception:
```

(continues on next page)

(continued from previous page)

```
ClientConnectorError                          Traceback (most recent call last)
Cell In[6], line 6
      1 cci_attrs = dict(
      2     ecv='AEROSOL',
      3     frequency='month'
      4 )
----> 6 [descriptor.data_id for descriptor in cci_store.search_data(cci_attrs=cci_attrs)]

File ~/projects/esa-cci/esa-climate-toolbox/esa_climate_toolbox/ds/dataaccess.py:721, in␣
↪CciCdcDataStore.search_data(self, data_type, **search_params)
    719 desc_iterators = []
    720 for opener in openers:
--> 721     desc_iterators.extend(opener.search_data(**search_params))
    722 return desc_iterators

File ~/projects/esa-cci/esa-climate-toolbox/esa_climate_toolbox/ds/dataaccess.py:186, in␣
↪CciCdcDataOpener.search_data(self, **search_params)
    184 def search_data(self, **search_params) -> Iterator[DatasetDescriptor]:
    185     search_result = self._cci_cdc.search(**search_params)
--> 186     data_descriptors = self._describe_data(search_result)
    187     return iter(data_descriptors)

File ~/projects/esa-cci/esa-climate-toolbox/esa_climate_toolbox/ds/dataaccess.py:126, in␣
↪CciCdcDataOpener._describe_data(self, data_ids)
    125 def _describe_data(self, data_ids: List[str]) -> List[DatasetDescriptor]:
--> 126     ds_metadata_list = self._cci_cdc.get_datasets_metadata(data_ids)
    127     data_descriptors = []
    128     for i, ds_metadata in enumerate(ds_metadata_list):

File ~/projects/esa-cci/esa-climate-toolbox/esa_climate_toolbox/ds/ccicdc.py:560, in␣
↪CciCdc.get_datasets_metadata(self, dataset_ids)
    558 def get_datasets_metadata(self, dataset_ids: List[str]) -> List[dict]:
    559     assert isinstance(dataset_ids, list)
--> 560     self._run_with_session(
    561         self._ensure_all_info_in_data_sources, dataset_ids
    562     )
    563     metadata = []
    564     for dataset_id in dataset_ids:

File ~/projects/esa-cci/esa-climate-toolbox/esa_climate_toolbox/ds/ccicdc.py:484, in␣
↪CciCdc._run_with_session(self, async_function, *params)
    480 loop = asyncio.new_event_loop()
    481 coro = _run_with_session_executor(
    482     async_function, *params, headers=self._headers
    483 )
--> 484 result = loop.run_until_complete(coro)
    485 # Short sleep to allow underlying connections to close
    486 loop.run_until_complete(asyncio.sleep(1.))

File ~/miniconda3/envs/ect/lib/python3.11/site-packages/nest_asyncio.py:99, in _patch_
↪loop.<locals>.run_until_complete(self, future)
     96 if not f.done():
```

(continues on next page)

(continued from previous page)

```
     97     raise RuntimeError(
     98         'Event loop stopped before Future completed.')
---> 99 return f.result()

File ~/miniconda3/envs/ect/lib/python3.11/asyncio/futures.py:203, in Future.result(self)
    201 self.__log_traceback = False
    202 if self._exception is not None:
--> 203     raise self._exception.with_traceback(self._exception_tb)
    204 return self._result

File ~/miniconda3/envs/ect/lib/python3.11/asyncio/tasks.py:277, in Task.__step(***failed␣
↪resolving arguments***)
    273 try:
    274     if exc is None:
    275         # We use the `send` method directly, because coroutines
    276         # don't have `__iter__` and `__next__` methods.
--> 277         result = coro.send(None)
    278     else:
    279         result = coro.throw(exc)

File ~/projects/esa-cci/esa-climate-toolbox/esa_climate_toolbox/ds/ccicdc.py:130, in _
↪run_with_session_executor(async_function, headers, *params)
    124 async def _run_with_session_executor(async_function, *params, headers):
    125     async with aiohttp.ClientSession(
    126             connector=aiohttp.TCPConnector(limit=50),
    127             headers=headers,
    128             trust_env=True
    129     ) as session:
--> 130         return await async_function(session, *params)

File ~/projects/esa-cci/esa-climate-toolbox/esa_climate_toolbox/ds/ccicdc.py:693, in␣
↪CciCdc._ensure_all_info_in_data_sources(self, session, dataset_names)
    689 for dataset_name in dataset_names:
    690     all_info_tasks.append(
    691         self._ensure_all_info_in_data_source(session, dataset_name)
    692     )
--> 693 await asyncio.gather(*all_info_tasks)

File ~/miniconda3/envs/ect/lib/python3.11/asyncio/tasks.py:349, in Task.__wakeup(self,␣
↪future)
    347 def __wakeup(self, future):
    348     try:
--> 349         future.result()
    350     except BaseException as exc:
    351         # This may also be a cancellation.
    352         self.__step(exc)

File ~/miniconda3/envs/ect/lib/python3.11/asyncio/tasks.py:279, in Task.__step(***failed␣
↪resolving arguments***)
    277         result = coro.send(None)
    278     else:
--> 279         result = coro.throw(exc)
```

(continues on next page)

(continued from previous page)

```
    280 except StopIteration as exc:
    281     if self._must_cancel:
    282         # Task is cancelled right before coro stops.

File ~/projects/esa-cci/esa-climate-toolbox/esa_climate_toolbox/ds/ccicdc.py:702, in␣
↪CciCdc._ensure_all_info_in_data_source(self, session, dataset_name)
    700     return
    701 data_fid = await self._get_dataset_id(session, dataset_name)
--> 702 await self._set_variable_infos(
    703     self._opensearch_url, data_fid, dataset_name, session, data_source
    704 )

File ~/projects/esa-cci/esa-climate-toolbox/esa_climate_toolbox/ds/ccicdc.py:1275, in␣
↪CciCdc._set_variable_infos(self, opensearch_url, dataset_id, dataset_name, session,␣
↪data_source)
   1265 feature, time_dimension_size = \
   1266     await self._fetch_feature_and_num_nc_files_at(
   1267         session,
   (...)
   1271         1
   1272     )
   1273 if feature is not None:
   1274     variable_infos, attributes = \
-> 1275         await self._get_variable_infos_from_feature(feature, session)
   1276     for variable_info in variable_infos:
   1277         for index, dimension in enumerate(
   1278                 variable_infos[variable_info]['dimensions']
   1279         ):

File ~/projects/esa-cci/esa-climate-toolbox/esa_climate_toolbox/ds/ccicdc.py:1411, in␣
↪CciCdc._get_variable_infos_from_feature(self, feature, session)
   1409     _LOG.warning(f'Dataset is not accessible via Opendap')
   1410     return {}, {}
-> 1411 dataset = await self._get_opendap_dataset(session, opendap_url)
   1412 if not dataset:
   1413     _LOG.warning(f'Could not extract information about variables '
   1414                  f'and attributes from {opendap_url}')

File ~/projects/esa-cci/esa-climate-toolbox/esa_climate_toolbox/ds/ccicdc.py:1493, in␣
↪CciCdc._get_opendap_dataset(self, session, url)
   1492 async def _get_opendap_dataset(self, session, url: str):
-> 1493     res_dict = await self._get_result_dict(session, url)
   1494     if 'dds' not in res_dict or 'das' not in res_dict:
   1495         _LOG.warning(
   1496             'Could not open opendap url. No dds or das file provided.'
   1497         )

File ~/projects/esa-cci/esa-climate-toolbox/esa_climate_toolbox/ds/ccicdc.py:1481, in␣
↪CciCdc._get_result_dict(self, session, url)
   1475 tasks.append(self._get_content_from_opendap_url(
   1476     url, 'dds', res_dict, session)
   1477 )
```

(continues on next page)

(continued from previous page)

```
   1478 tasks.append(self._get_content_from_opendap_url(
   1479     url, 'das', res_dict, session)
   1480 )
-> 1481 await asyncio.gather(*tasks)
   1482 if 'das' in res_dict:
   1483     res_dict['das'] = res_dict['das'].replace(
   1484         '        Float32 valid_min -Infinity;\n', ''
   1485     )

File ~/miniconda3/envs/ect/lib/python3.11/asyncio/tasks.py:349, in Task.__wakeup(self,␣
↪future)
    347 def __wakeup(self, future):
    348     try:
--> 349         future.result()
    350     except BaseException as exc:
    351         # This may also be a cancellation.
    352         self.__step(exc)

File ~/miniconda3/envs/ect/lib/python3.11/asyncio/tasks.py:279, in Task.__step(***failed␣
↪resolving arguments***)
    277         result = coro.send(None)
    278     else:
--> 279         result = coro.throw(exc)
    280 except StopIteration as exc:
    281     if self._must_cancel:
    282         # Task is cancelled right before coro stops.

File ~/projects/esa-cci/esa-climate-toolbox/esa_climate_toolbox/ds/ccicdc.py:1544, in␣
↪CciCdc._get_content_from_opendap_url(self, url, part, res_dict, session)
   1542 scheme, netloc, path, query, fragment = urlsplit(url)
   1543 url = urlunsplit((scheme, netloc, path + f'.{part}', query, fragment))
-> 1544 resp = await self.get_response(session, url)
   1545 if resp:
   1546     res_dict[part] = await resp.read()

File ~/projects/esa-cci/esa-climate-toolbox/esa_climate_toolbox/ds/ccicdc.py:1585, in␣
↪CciCdc.get_response(self, session, url)
   1583 retry_backoff_base = self._retry_backoff_base
   1584 for i in range(num_retries):
-> 1585     resp = await session.request(method='GET', url=url)
   1586     if resp.status == 200:
   1587         return resp

File ~/miniconda3/envs/ect/lib/python3.11/site-packages/aiohttp/client.py:562, in␣
↪ClientSession._request(self, method, str_or_url, params, data, json, cookies, headers,␣
↪skip_auto_headers, auth, allow_redirects, max_redirects, compress, chunked, expect100,␣
↪raise_for_status, read_until_eof, proxy, proxy_auth, timeout, verify_ssl, fingerprint,␣
↪ssl_context, ssl, proxy_headers, trace_request_ctx, read_bufsize)
    560     async with ceil_timeout(real_timeout.connect):
    561         assert self._connector is not None
--> 562         conn = await self._connector.connect(
    563             req, traces=traces, timeout=real_timeout
```

(continues on next page)

(continued from previous page)

```
    564         )
    565 except asyncio.TimeoutError as exc:
    566     raise ServerTimeoutError(
    567         "Connection timeout " "to host {}".format(url)
    568     ) from exc

File ~/miniconda3/envs/ect/lib/python3.11/site-packages/aiohttp/connector.py:540, in␣
↪BaseConnector.connect(self, req, traces, timeout)
    537         await trace.send_connection_create_start()
    539 try:
--> 540     proto = await self._create_connection(req, traces, timeout)
    541     if self._closed:
    542         proto.close()

File ~/miniconda3/envs/ect/lib/python3.11/site-packages/aiohttp/connector.py:901, in␣
↪TCPConnector._create_connection(self, req, traces, timeout)
    899     _, proto = await self._create_proxy_connection(req, traces, timeout)
    900 else:
--> 901     _, proto = await self._create_direct_connection(req, traces, timeout)
    903 return proto

File ~/miniconda3/envs/ect/lib/python3.11/site-packages/aiohttp/connector.py:1209, in␣
↪TCPConnector._create_direct_connection(self, req, traces, timeout, client_error)
   1207 else:
   1208     assert last_exc is not None
-> 1209     raise last_exc

File ~/miniconda3/envs/ect/lib/python3.11/site-packages/aiohttp/connector.py:1178, in␣
↪TCPConnector._create_direct_connection(self, req, traces, timeout, client_error)
   1175 port = hinfo["port"]
   1177 try:
-> 1178     transp, proto = await self._wrap_create_connection(
   1179         self._factory,
   1180         host,
   1181         port,
   1182         timeout=timeout,
   1183         ssl=sslcontext,
   1184         family=hinfo["family"],
   1185         proto=hinfo["proto"],
   1186         flags=hinfo["flags"],
   1187         server_hostname=hinfo["hostname"] if sslcontext else None,
   1188         local_addr=self._local_addr,
   1189         req=req,
   1190         client_error=client_error,
   1191     )
   1192 except ClientConnectorError as exc:
   1193     last_exc = exc

File ~/miniconda3/envs/ect/lib/python3.11/site-packages/aiohttp/connector.py:988, in␣
↪TCPConnector._wrap_create_connection(self, req, timeout, client_error, *args, **kwargs)
    986 if exc.errno is None and isinstance(exc, asyncio.TimeoutError):
    987     raise
```

(continues on next page)

(continued from previous page)

```
--> 988 raise client_error(req.connection_key, exc) from exc

ClientConnectorError: Cannot connect to host data.cci.ceda.ac.uk:443 ssl:default [None]
```

'esacci.AEROSOL.mon.L3.AAI.multi-sensor.multi-platform.MSAAI.1-7.r1' sounds interesting. Let's get more information about that one.

[ ]:
```
cci_store.describe_data('esacci.AEROSOL.mon.L3.AAI.multi-sensor.multi-platform.MSAAI.1-7.
↪r1')
```

Finally, we can have a look at which openers are provided by this store.

[ ]:
```
cci_store.get_data_opener_ids()
```

Here is how you can view which parameters can be used when you want to open a particular dataset (both parameters are optional).

[ ]:
```
cci_store.get_open_data_params_schema(data_id='esacci.AEROSOL.mon.L3.AAI.multi-sensor.
↪multi-platform.MSAAI.1-7.r1', opener_id='dataset:zarr:esa-cdc')
```

Great! Now we are all set to open some cubes.

ESA CCI Toolbox Data Access with Parameters

This notebook shows how to open data cubes from the ESA Open Data Portal for a given time range and region:

A **temporarily regular cube** with aggregated CCI data that fall into equal-size time periods;

We start with necessary imports.

[1]:
```
from xcube.core.store import new_data_store
import IPython.display
import shapely
```

Now we can create the store.

[2]:
```
cci_store = new_data_store('esa-cci')
```

For this demo, we are using the dataset with the id 'esacci.AEROSOL.mon.L3.AAI.multi-sensor.multi-platform.MSAAI.1-7.r1'. Here is its metadata:

[3]:
```
cci_store.describe_data('esacci.AEROSOL.mon.L3.AAI.multi-sensor.multi-platform.MSAAI.1-7.
↪r1')
```

[3]:
```
<xcube.core.store.descriptor.DatasetDescriptor at 0x7f1837f9a0d0>
```

We are focussing on Africa, so we define a bounding box accordingly.

[4]:
```
x1 = -23.40  # degree
y1 = -40.40  # degree
x2 = 57.40  # degree
y2 = 40.40  # degree

bbox = (x1, y1, x2, y2)
```

Visualize the bounding box. If you don't see anything, please refer to Ex0-DCFS-Setup.

[5]:
```
IPython.display.GeoJSON(shapely.geometry.box(*bbox).__geo_interface__)
```

```
<IPython.display.GeoJSON object>
```

Our time range covers data from 1997: 1997-01-01 to 1997-12-01

Now to open the dataset. You may check its metadata first to make sure that your parameters are valid.

[6]:
```
cci_store.get_open_data_params_schema(data_id='esacci.AEROSOL.mon.L3.AAI.multi-sensor.
↪multi-platform.MSAAI.1-7.r1', opener_id='dataset:zarr:esa-cdc')
```

[6]:
```
<xcube.util.jsonschema.JsonObjectSchema at 0x7f183779f490>
```

Select any valid combination from the paramater in *properties* above. The only mandatory parameter is the data_id. If you leave any parameter out, all available values will be considered.

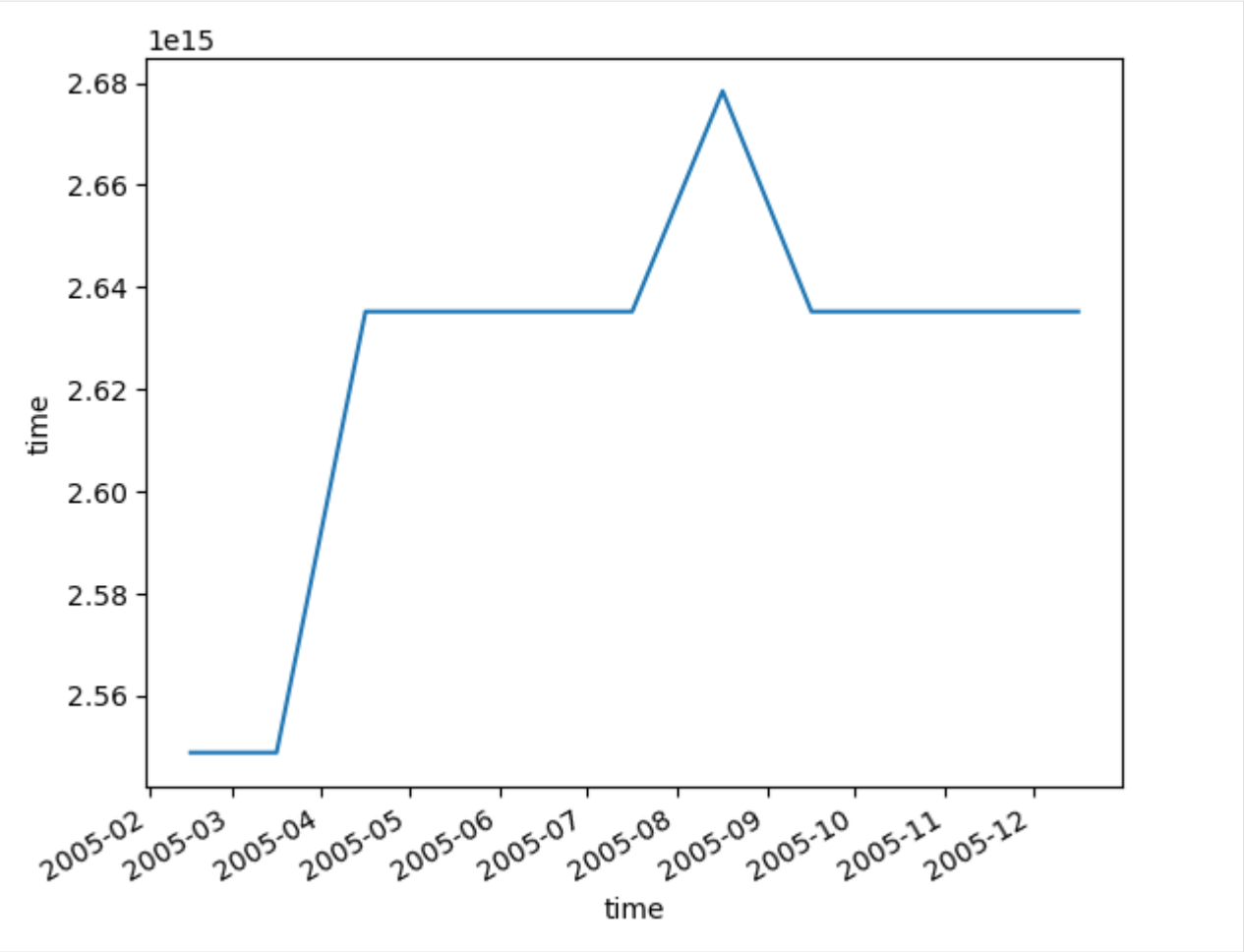
[7]:
```
cube = cci_store.open_data('esacci.AEROSOL.mon.L3.AAI.multi-sensor.multi-platform.MSAAI.
↪1-7.r1',
                           variable_names=['absorbing_aerosol_index'],
                           bbox=bbox,
                           time_range=['2005-01-01', '2005-12-31'])
cube
```

[7]:
```
<xarray.Dataset>
Dimensions:                  (time: 12, lat: 80, lon: 80, bnds: 2)
Coordinates:
  * lat                      (lat) float32 -39.5 -38.5 -37.5 ... 37.5 38.5 39.5
  * lon                      (lon) float32 -22.5 -21.5 -20.5 ... 54.5 55.5 56.5
  * time                     (time) datetime64[ns] 2005-01-16T12:00:00 ... 20...
    time_bnds                (time, bnds) datetime64[ns] dask.array<chunksize=(12, 2),␣
↪meta=np.ndarray>
Dimensions without coordinates: bnds
Data variables:
    absorbing_aerosol_index  (time, lat, lon) float32 dask.array<chunksize=(1, 80, 80),␣
↪meta=np.ndarray>
Attributes:
    Conventions:             CF-1.7
    title:                   esacci.AEROSOL.mon.L3.AAI.multi-sensor.multi-pla...
    date_created:            2024-02-29T21:53:26.841148
    processing_level:        L3
    time_coverage_start:     2005-01-01T00:00:00
    time_coverage_end:       2006-01-01T00:00:00
    time_coverage_duration:  P365DT0H0M0S
    history:                 [{'program': 'esa_climate_toolbox.ds.chunkstore...
```

We may visualize the cube's time coordinates:

[8]:
```
cube.time.diff(dim='time').plot.line()
```

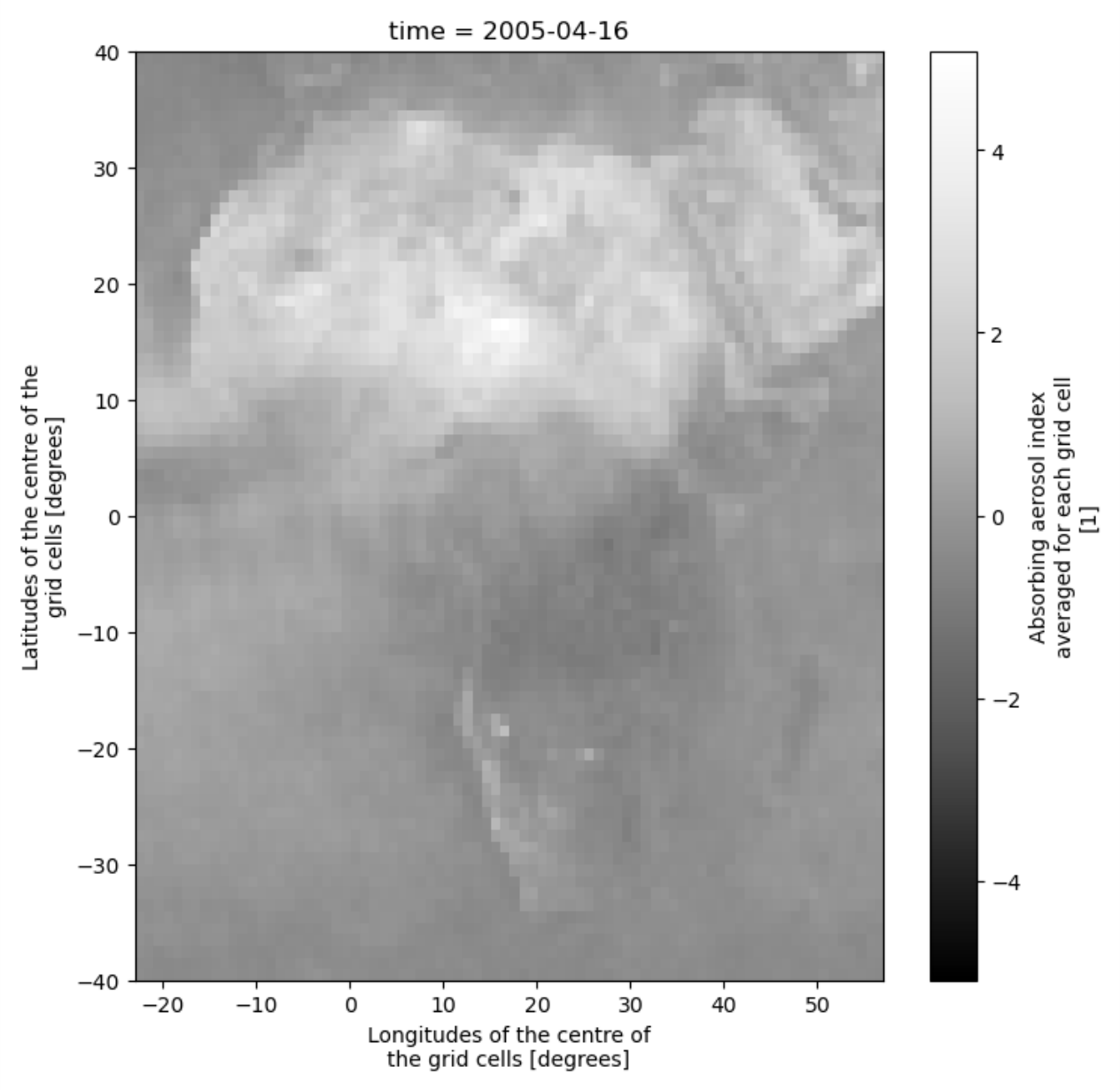
[8]:
```
[<matplotlib.lines.Line2D at 0x7f183610fed0>]
```



And finally, display the results:

[9]:
```
cube.absorbing_aerosol_index.sel(time='2005-04-15 12:00:00', method='nearest').plot.
↪imshow(cmap='Greys_r', figsize=(8, 8))
```

[9]:
```
<matplotlib.image.AxesImage at 0x7f18361b19d0>
```



[ ]:

ESA CCI Toolbox Vector Data Access

The ESA CCI Toolbox also provides access to data that is not provided on a structured grid. This data is provided in the form of geopandas geodataframes rather than xarray datasets.

To run this Notebook, make sure the ESA CCI Toolbox is setup correctly.

For this notebook, we start as before, by opening the standard `esa-cci` data store.

[1]:
```
from xcube.core.store import new_data_store

cci_store = new_data_store('esa-cci')
```

The data in question are GHG, SEALEVEL, and SEAICE datasets in `satellite-orbit-frequency`. Let's search for GHG datasets.

[2]:
```
descriptors = cci_store.search_data(
    data_type="geodataframe",
    cci_attrs=dict(
        ecv="GHG"
    )
)
[descriptor.data_id for descriptor in descriptors]
```

[2]:
```
['esacci.GHG.satellite-orbit-frequency.L2.CH4.SCIAMACHY.Envisat.IMAP.v7-2.r1',
 'esacci.GHG.satellite-orbit-frequency.L2.CH4.SCIAMACHY.Envisat.WFMD.v4-0.r1',
 'esacci.GHG.satellite-orbit-frequency.L2.CH4.TANSO-FTS.GOSAT.EMMA.ch4_v1-2.r1',
 'esacci.GHG.satellite-orbit-frequency.L2.CH4.TANSO-FTS.GOSAT.OCFP.v2-1.r1',
 'esacci.GHG.satellite-orbit-frequency.L2.CH4.TANSO-FTS.GOSAT.OCPR.v7-0.r1',
 'esacci.GHG.satellite-orbit-frequency.L2.CH4.TANSO-FTS.GOSAT.SRFP.v2-3-8.r1',
 'esacci.GHG.satellite-orbit-frequency.L2.CH4.TANSO-FTS.GOSAT.SRPR.v2-3-8.r1',
 'esacci.GHG.satellite-orbit-frequency.L2.CO2.SCIAMACHY.Envisat.BESD.v02-01-02.r1',
 'esacci.GHG.satellite-orbit-frequency.L2.CO2.SCIAMACHY.Envisat.WFMD.v4-0.r1',
 'esacci.GHG.satellite-orbit-frequency.L2.CO2.TANSO-FTS.GOSAT.EMMA.v2-2c.r1',
 'esacci.GHG.satellite-orbit-frequency.L2.CO2.TANSO-FTS.GOSAT.SRFP.v2-3-8.r1']
```

We have a closer look at the first one.

[3]:
```
descriptors[0]
```

[3]:
```
<xcube.core.store.descriptor.GeoDataFrameDescriptor at 0x7fa9e25a07d0>
```

It makes sense to not open the whole dataframe but read in a subset. We ask for two variables for one day (the variable names are listed under feature_schema:properties).

[4]:
```
var_names = ["xch4", "xco2_retrieved"]
time_range = ["2010-07-04", "2010-07-04"]

ghg_df = cci_store.open_data(
    "esacci.GHG.satellite-orbit-frequency.L2.CH4.SCIAMACHY.Envisat.IMAP.v7-2.r1",
    variable_names=var_names,
    time_range=time_range
)
ghg_df
```

[4]:
```
                        geometry         xch4  xco2_retrieved  \
0      POINT (21.12531 66.15704)  1835.639282      350.589783
1      POINT (17.33319 54.23273)  1848.246216      350.940857
2      POINT (21.32617 53.16351)  1834.837891      361.529053
3      POINT (17.02777 53.65680)  1767.532104      357.012085
4      POINT (15.31268 53.61804)  1846.445312      384.051636
...                          ...          ...             ...
3546  POINT (176.04736 62.59637)  1756.709839      379.857666
3547  POINT (173.65729 62.86865)  1774.327759      357.646545
3548  POINT (174.71912 62.44507)  1805.762573      374.800110
3549  POINT (173.18512 62.00031)  1841.642090      350.549011
3550  POINT (171.62189 61.54938)  1752.140625      350.487610
```

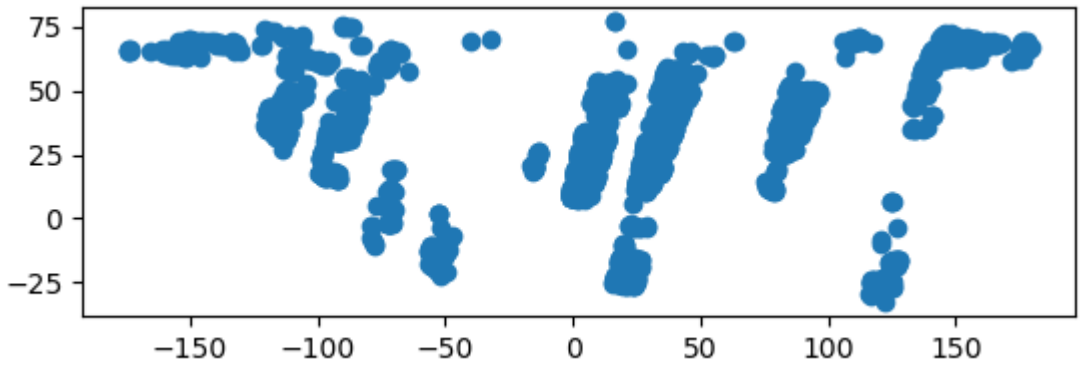
(continues on next page)

(continued from previous page)

```
                              time
0     2010-07-04 09:37:42.694144249
1     2010-07-04 09:41:00.557484627
2     2010-07-04 09:41:04.307856560
3     2010-07-04 09:41:10.557832718
4     2010-07-04 09:41:16.057863235
...                             ...
3546  2010-07-04 23:03:35.278387070
3547  2010-07-04 23:03:35.778565407
3548  2010-07-04 23:03:40.528650284
3549  2010-07-04 23:03:50.778765678
3550  2010-07-04 23:04:01.028881073

[3551 rows x 4 columns]
```

[5]:
```
ghg_df.plot()
```

[5]:
```
<Axes: >
```



Access to Zarr Data

Some datasets from the Open Data Portal have been migrated to the Zarr format. This allows for faster opening and processing, so it makes sense to check whether data is provided in the Zarr format first.

The data is available via store abbreviation ‘cci-zarr-store’.

[1]:
```
from xcube.core.store import new_data_store
```

[2]:
```
zarr_store = new_data_store('esa-cci-zarr')
```

Again, let’s see what data sets are available.

[3]:
```
datasets = zarr_store.list_data_ids()
datasets
```

[3]:
```
['ESACCI-BIOMASS-L4-AGB-MERGED-100m-2010-2018-fv2.0.zarr',
 'ESACCI-BIOMASS-L4-AGB-MERGED-100m-2010-2020-fv4.0.zarr',
 'ESACCI-GHG-L2-CH4-SCIAMACHY-WFMD-2002-2011-fv1.zarr',
 'ESACCI-GHG-L2-CO2-OCO-2-FOCAL-2014-2021-v10.zarr',
```

(continues on next page)

(continued from previous page)

```
 'ESACCI-GHG-L2-CO2-SCIAMACHY-WFMD-2002-2012-fv1.zarr',
 'ESACCI-ICESHEETS_Antarctica_GMB-2002-2016-v1.1.zarr',
 'ESACCI-ICESHEETS_Greenland_GMB-2003-2016-v1.1.zarr',
 'ESACCI-L3C_CLOUD-CLD_PRODUCTS-AVHRR_NOAA-1982-2016-fv3.0.zarr',
 'ESACCI-L3C_SNOW-SWE-1979-2018-fv1.0.zarr',
 'ESACCI-L3C_SNOW-SWE-1979-2020-fv2.0.zarr',
 'ESACCI-L4_GHRSST-SST-GMPE-GLOB_CDR2.0-1981-2016-v02.0-fv01.0.zarr',
 'ESACCI-LAKES-L3S-LK_PRODUCTS-MERGED-1992-09-fv2.0.1.zarr',
 'ESACCI-LC-L4-LCCS-Map-300m-P1Y-1992-2015-v2.0.7b.zarr',
 'ESACCI-LST-L3C-LST-MODISA-0.01deg_1DAILY_DAY-2002-2018-fv3.00.zarr',
 'ESACCI-LST-L3C-LST-MODISA-0.01deg_1DAILY_NIGHT-2002-2018-fv3.00.zarr',
 'ESACCI-LST-L3S-LST-IRCDR_-0.01deg_1DAILY_DAY-1995-2020-fv3.00.zarr',
 'ESACCI-LST-L3S-LST-IRCDR_-0.01deg_1DAILY_NIGHT-1995-2020-fv3.00.zarr',
 'ESACCI-LST-L3S-LST-IRCDR_-0.01deg_1MONTHLY_DAY-1995-2020-fv3.00.zarr',
 'ESACCI-LST-L3S-LST-IRCDR_-0.01deg_1MONTHLY_NIGHT-1995-2020-fv3.00.zarr',
 'ESACCI-OC-L3S-IOP-MERGED-1M_MONTHLY_4km_GEO_PML_OCx-QAA-1997-2022-fv6.0.zarr',
 'ESACCI-OC-L3S-IOP-MERGED-1M_MONTHLY_4km_GEO_PML_OCx_QAA-1997-2020-fv5.0.zarr',
 'ESACCI-OC-L3S-IOP-MERGED-1Y_YEARLY_4km_GEO_PML_OCx_QAA-1997-2022-fv6.0.zarr',
 'ESACCI-OC-L3S-IOP-MERGED-1Y_YEARLY_4km_GEO_PML_RRS-1997-2022-fv6.0.zarr',
 'ESACCI-OC-L3S-IOP-MERGED-8D_DAILY_4km_GEO_PML_OCx_QAA-1997-2022-fv6.0.zarr',
 'ESACCI-OC-L3S-OC_PRODUCTS-MERGED-1Y_YEARLY_4km_GEO_PML_OCx_QAA-1997-2022-fv6.0.zarr',
 'ESACCI-OC-L3S-RRS-MERGED-1M_MONTHLY_4km_GEO_PML_RRS-1997-2022-fv6.0.zarr',
 'ESACCI-PERMAFROST-L4-ALT-MODISLST-AREA4_PP-1997-2018-fv02.0.zarr',
 'ESACCI-SEAICE-L3C-SITHICK-RA2_ENVISAT-NH25KMEASE2-2002-2012-fv2.0.zarr',
 'ESACCI-SEAICE-L3C-SITHICK-SIRAL_CRYOSAT2-NH25KMEASE2-2010-2017-fv2.0.zarr',
 'ESACCI-SEAICE-L4-SICONC-AMSR_50.0kmEASE2-NH-2002-2017-fv2.1.zarr',
 'ESACCI-SEALEVEL-IND-MSLTR-MERGED-1993-2016-fv02.zarr',
 'ESACCI-SEALEVEL-L4-MSLA-MERGED-1993-2015-fv02.zarr',
 'ESACCI-SOILMOISTURE-L3S-SSMV-COMBINED-1978-2020-fv05.3.zarr',
 'ESACCI-SOILMOISTURE-L3S-SSMV-COMBINED-1978-2021-fv07.1.zarr']
```

The names are similar but different to the ones from the Climate Toolbox store. We can have a look at a dataset's metadata to find out more about it.

[4]:
```
zarr_store.describe_data('ESACCI-SOILMOISTURE-L3S-SSMV-COMBINED-1978-2021-fv07.1.zarr')
```

[4]:
```
<xcube.core.store.descriptor.DatasetDescriptor at 0x7fe7536dcad0>
```

Cubes can easily be opened from the store like this:

[5]:
```
cube = zarr_store.open_data('ESACCI-SOILMOISTURE-L3S-SSMV-COMBINED-1978-2021-fv07.1.zarr
↪')
cube
```

[5]:
```
<xarray.Dataset>
Dimensions:         (time: 15767, lat: 720, lon: 1440)
Coordinates:
  * lat             (lat) float64 89.88 89.62 89.38 ... -89.38 -89.62 -89.88
  * lon             (lon) float64 -179.9 -179.6 -179.4 ... 179.4 179.6 179.9
  * time            (time) datetime64[ns] 1978-11-01 1978-11-02 ... 2021-12-31
Data variables:
    dnflag          (time, lat, lon) float32 dask.array<chunksize=(16, 720, 720),␣
↪meta=np.ndarray>
```

(continues on next page)

(continued from previous page)

```
    flag            (time, lat, lon) float32 dask.array<chunksize=(16, 720, 720),␣
↪meta=np.ndarray>
    freqbandID      (time, lat, lon) float32 dask.array<chunksize=(16, 720, 720),␣
↪meta=np.ndarray>
    mode            (time, lat, lon) float32 dask.array<chunksize=(16, 720, 720),␣
↪meta=np.ndarray>
    sensor          (time, lat, lon) float64 dask.array<chunksize=(16, 720, 720),␣
↪meta=np.ndarray>
    sm              (time, lat, lon) float32 dask.array<chunksize=(16, 720, 720),␣
↪meta=np.ndarray>
    sm_uncertainty  (time, lat, lon) float32 dask.array<chunksize=(16, 720, 720),␣
↪meta=np.ndarray>
    t0              (time, lat, lon) float64 dask.array<chunksize=(16, 720, 720),␣
↪meta=np.ndarray>
Attributes: (12/44)
    Conventions:                  CF-1.9
    cdm_data_type:                Grid
    comment:                      This dataset was produced with funding of t...
    contact:                      cci_sm_contact@eodc.eu
    creator_email:                cci_sm_developer@eodc.eu
    creator_name:                 Department of Geodesy and Geoinformation, V...
    ...                           ...
    time_coverage_end_product:    20211231T235959Z
    time_coverage_resolution:     P1D
    time_coverage_start:          1978-11-01 00:00:00
    time_coverage_start_product:  19781101T000000Z
    title:                        ESA CCI Surface Soil Moisture COMBINED acti...
    tracking_id:                  ad35798e-58e0-488f-b5b9-593874a47700
```

Subsets of the data may easily be created like this:

[6]:
```
sub_cube = cube.sel({
    'lat': slice(40.40, -40.40),
    'lon': slice(-23.40, 57.40),
    'time': slice('2000-01-01', '2000-12-31')
    }
)
sub_cube
```

[6]:
```
<xarray.Dataset>
Dimensions:         (time: 366, lat: 324, lon: 324)
Coordinates:
  * lat             (lat) float64 40.38 40.12 39.88 ... -39.88 -40.12 -40.38
  * lon             (lon) float64 -23.38 -23.12 -22.88 ... 56.88 57.12 57.38
  * time            (time) datetime64[ns] 2000-01-01 2000-01-02 ... 2000-12-31
Data variables:
    dnflag          (time, lat, lon) float32 dask.array<chunksize=(13, 324, 94), meta=np.
↪ndarray>
    flag            (time, lat, lon) float32 dask.array<chunksize=(13, 324, 94), meta=np.
↪ndarray>
    freqbandID      (time, lat, lon) float32 dask.array<chunksize=(13, 324, 94), meta=np.
↪ndarray>
    mode            (time, lat, lon) float32 dask.array<chunksize=(13, 324, 94), meta=np.
```

(continues on next page)

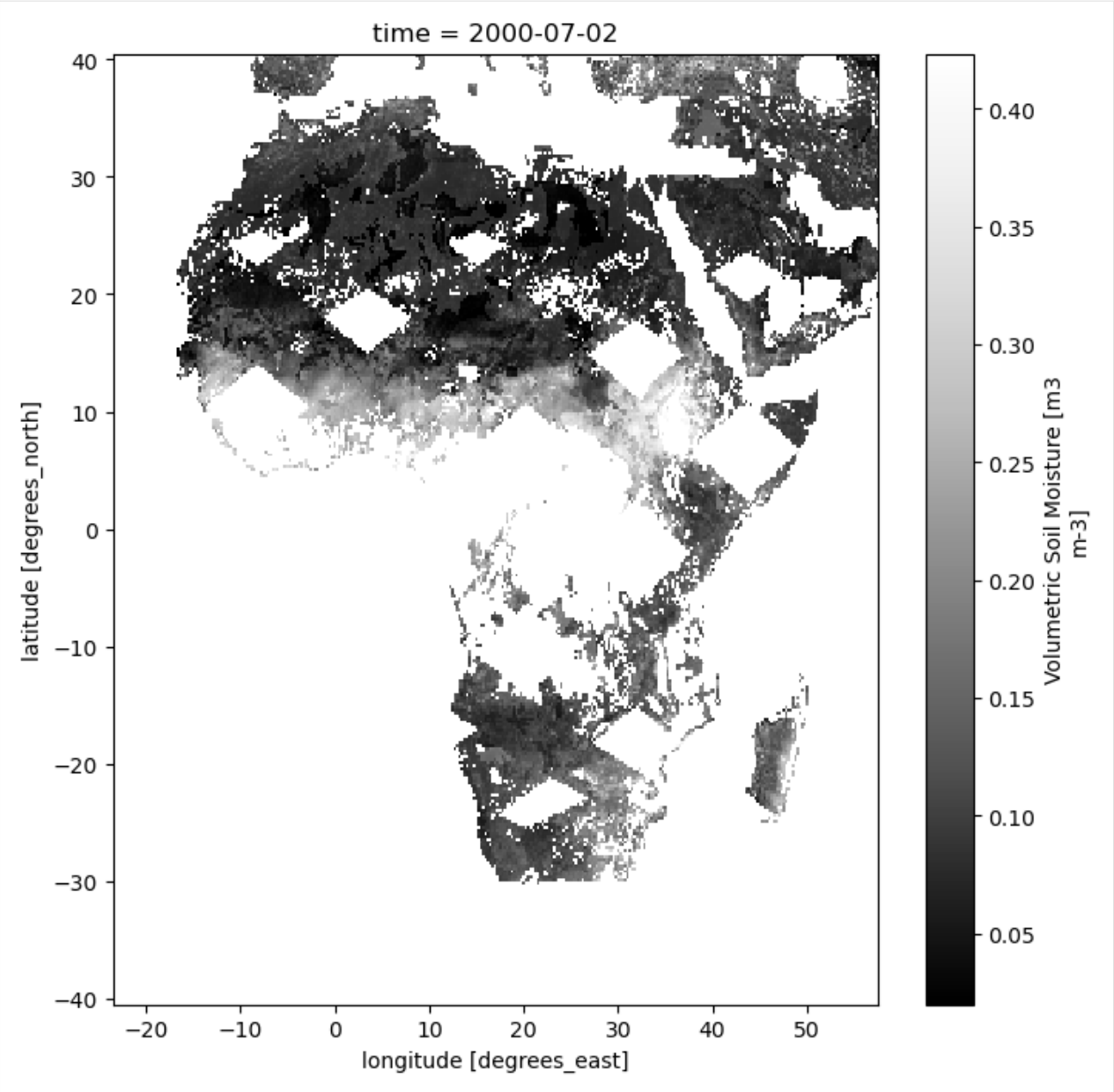
(continued from previous page)

```
↪ndarray>
    sensor          (time, lat, lon) float64 dask.array<chunksize=(13, 324, 94), meta=np.
↪ndarray>
    sm              (time, lat, lon) float32 dask.array<chunksize=(13, 324, 94), meta=np.
↪ndarray>
    sm_uncertainty  (time, lat, lon) float32 dask.array<chunksize=(13, 324, 94), meta=np.
↪ndarray>
    t0              (time, lat, lon) float64 dask.array<chunksize=(13, 324, 94), meta=np.
↪ndarray>
Attributes: (12/44)
    Conventions:                  CF-1.9
    cdm_data_type:                Grid
    comment:                      This dataset was produced with funding of t...
    contact:                      cci_sm_contact@eodc.eu
    creator_email:                cci_sm_developer@eodc.eu
    creator_name:                 Department of Geodesy and Geoinformation, V...
    ...                           ...
    time_coverage_end_product:    20211231T235959Z
    time_coverage_resolution:     P1D
    time_coverage_start:          1978-11-01 00:00:00
    time_coverage_start_product:  19781101T000000Z
    title:                        ESA CCI Surface Soil Moisture COMBINED acti...
    tracking_id:                  ad35798e-58e0-488f-b5b9-593874a47700
```

... and we can plot the data.

[7]:
```
sub_cube.sm.sel(time='2000-07-01 12:00:00', method='nearest').plot.imshow(cmap='Greys_r',
↪ figsize=(8, 8))
```

[7]:
```
<matplotlib.image.AxesImage at 0x7fe752e0f350>
```



Access to Data in NetCDF format viewed as Zarr

Some datasets offered by the Open Data Portal are provided via references format as specified by the kerchunk package. This allows accessing to the original NetCDF data files with an I/O performance comparable to Zarr.

The data is available via store identifier 'cci-kerchunk-store'.

[1]:
```
from xcube.core.store import new_data_store
```

[2]:
```
kerchunk_store = new_data_store('esa-cci-kc')
```

Again, let's see what data sets are available.

[3]:
```
datasets = kerchunk_store.list_data_ids()
datasets
```

[3]:
```
['ESACCI-BIOMASS-L4-AGB-CHANGE-100m-2018-2017-fv4.0-kr1.1',
 'ESACCI-BIOMASS-L4-AGB-CHANGE-100m-2019-2018-fv4.0-kr1.1',
 'ESACCI-BIOMASS-L4-AGB-CHANGE-100m-2020-2010-fv4.0-kr1.1',
 'ESACCI-BIOMASS-L4-AGB-CHANGE-100m-2020-2019-fv4.0-kr1.1',
 'ESACCI-BIOMASS-L4-AGB-MERGED-100m-2010-2020-fv4.0-kr1.1',
 'ESACCI-L3C_CLOUD-CLD_PRODUCTS-ATSR2_AATSR-199506-201204-fv3.0-kr1.1',
 'ESACCI-L3C_CLOUD-CLD_PRODUCTS-AVHRR_AM-199109-201612-fv3.0-kr1.1',
 'ESACCI-L3C_CLOUD-CLD_PRODUCTS-AVHRR_PM-198201-201612-fv3.0-kr1.1',
 'ESACCI-L3C_SNOW-SWE-MERGED-19790102-20200524-fv2.0-kr1.1',
 'ESACCI-L4_FIRE-BA-MODIS-20010101-20200120-fv5.1-kr1.2',
 'ESACCI-LC-L4-LCCS-Map-300m-P1Y-1992-2015-v2.0.7b-kr1.1',
 'ESACCI-LC-L4-PFT-Map-300m-P1Y-1992-2020-v2.0.8-kr1.1',
 'ESACCI-LST-L3C-LST-MODISA-0.01deg_1MONTHLY_DAY-200207-201812-fv3.00-kr1.1',
 'ESACCI-LST-L3C-LST-MODISA-0.01deg_1MONTHLY_NIGHT-200207-201812-fv3.00-kr1.1',
 'ESACCI-LST-L3C-LST-MODIST-0.01deg_1MONTHLY_DAY-200003-201812-fv3.00-kr1.1',
 'ESACCI-LST-L3C-LST-MODIST-0.01deg_1MONTHLY_NIGHT-200003-201812-fv3.00-kr1.1',
 'ESACCI-LST-L3S-LST-IRCDR_-0.01deg_1MONTHLY_DAY-199508-202012-fv2.00-kr1.1',
 'ESACCI-LST-L3S-LST-IRCDR_-0.01deg_1MONTHLY_NIGHT-199508-202012-fv2.00-kr1.1',
 'ESACCI-PERMAFROST-L4-ALT-ERA5_MODISLST_BIASCORRECTED-AREA4_PP-1997-2002-fv03.0-kr1.1',
 'ESACCI-PERMAFROST-L4-ALT-MODISLST_CRYOGRID-AREA4_PP-2003-2019-fv03.0-kr1.1',
 'ESACCI-SOILMOISTURE-L3S-SSMS-ACTIVE-19910805-20211231-fv07.1-kr1.1',
 'ESACCI-SOILMOISTURE-L3S-SSMV-COMBINED-19781101-20211231-fv07.1-kr1.1',
 'ESACCI-SOILMOISTURE-L3S-SSMV-PASSIVE-19781101-20211231-fv07.1-kr1.1',
 'ESACCI-SOILMOISTURE-L3S-SSMV-COMBINED_ADJUSTED-19781101-20211231-fv07.1-kr1.1']
```

The names are similar but different to the ones from the Climate Toolbox store. We can have a look at a dataset's metadata to find out more about it.

[4]:
```
kerchunk_store.describe_data('ESACCI-SOILMOISTURE-L3S-SSMV-COMBINED-19781101-20211231-
↪fv07.1-kr1.1')
```

[4]:
```
<xcube.core.store.descriptor.DatasetDescriptor at 0x7f18e8e72810>
```

Cubes can easily be opened from the store like this:

[5]:
```
cube = kerchunk_store.open_data('ESACCI-SOILMOISTURE-L3S-SSMV-COMBINED-19781101-20211231-
↪fv07.1-kr1.1')
cube
```

[5]:
```
<xarray.Dataset>
Dimensions:         (time: 15767, lat: 720, lon: 1440)
Coordinates:
  * lat             (lat) float64 89.88 89.62 89.38 ... -89.38 -89.62 -89.88
  * lon             (lon) float64 -179.9 -179.6 -179.4 ... 179.4 179.6 179.9
  * time            (time) datetime64[ns] 1978-11-01 1978-11-02 ... 2021-12-31
Data variables:
    dnflag          (time, lat, lon) float32 dask.array<chunksize=(1, 720, 1440),␣
↪meta=np.ndarray>
    flag            (time, lat, lon) float32 dask.array<chunksize=(1, 720, 1440),␣
↪meta=np.ndarray>
    freqbandID      (time, lat, lon) float32 dask.array<chunksize=(1, 720, 1440),␣
↪meta=np.ndarray>
```

(continues on next page)

(continued from previous page)

```
    mode            (time, lat, lon) float32 dask.array<chunksize=(1, 720, 1440),␣
↪meta=np.ndarray>
    sensor          (time, lat, lon) float64 dask.array<chunksize=(1, 720, 1440),␣
↪meta=np.ndarray>
    sm              (time, lat, lon) float32 dask.array<chunksize=(1, 720, 1440),␣
↪meta=np.ndarray>
    sm_uncertainty  (time, lat, lon) float32 dask.array<chunksize=(1, 720, 1440),␣
↪meta=np.ndarray>
    t0              (time, lat, lon) datetime64[ns] dask.array<chunksize=(1, 720, 1440),␣
↪meta=np.ndarray>
Attributes: (12/46)
    Conventions:                  CF-1.9
    cdm_data_type:                Grid
    comment:                      This dataset was produced with funding of t...
    contact:                      cci_sm_contact@eodc.eu
    creator_email:                cci_sm_developer@eodc.eu
    creator_name:                 Department of Geodesy and Geoinformation, V...
    ...                           ...
    time_coverage_start:          19781101T000000Z
    time_coverage_start_product:  19781101T000000Z
    title:                        ESA CCI Surface Soil Moisture COMBINED acti...
    tracking_id:                  dc6ade2c-e51b-4a94-81fa-751df95a85a6
    kerchunk_revision:            kr1.1
    kerchunk_creation_date:       031023T093359
```

Subsets of the data may easily be created like this:

[6]:
```
sub_cube = cube.sel({
    'lat': slice(40.40, -40.40),
    'lon': slice(-23.40, 57.40),
    'time': slice('2000-01-01', '2000-12-31')
    }
)
sub_cube
```

[6]:
```
<xarray.Dataset>
Dimensions:         (time: 366, lat: 324, lon: 324)
Coordinates:
  * lat             (lat) float64 40.38 40.12 39.88 ... -39.88 -40.12 -40.38
  * lon             (lon) float64 -23.38 -23.12 -22.88 ... 56.88 57.12 57.38
  * time            (time) datetime64[ns] 2000-01-01 2000-01-02 ... 2000-12-31
Data variables:
    dnflag          (time, lat, lon) float32 dask.array<chunksize=(1, 324, 324), meta=np.
↪ndarray>
    flag            (time, lat, lon) float32 dask.array<chunksize=(1, 324, 324), meta=np.
↪ndarray>
    freqbandID      (time, lat, lon) float32 dask.array<chunksize=(1, 324, 324), meta=np.
↪ndarray>
    mode            (time, lat, lon) float32 dask.array<chunksize=(1, 324, 324), meta=np.
↪ndarray>
    sensor          (time, lat, lon) float64 dask.array<chunksize=(1, 324, 324), meta=np.
↪ndarray>
    sm              (time, lat, lon) float32 dask.array<chunksize=(1, 324, 324), meta=np.
```

(continues on next page)

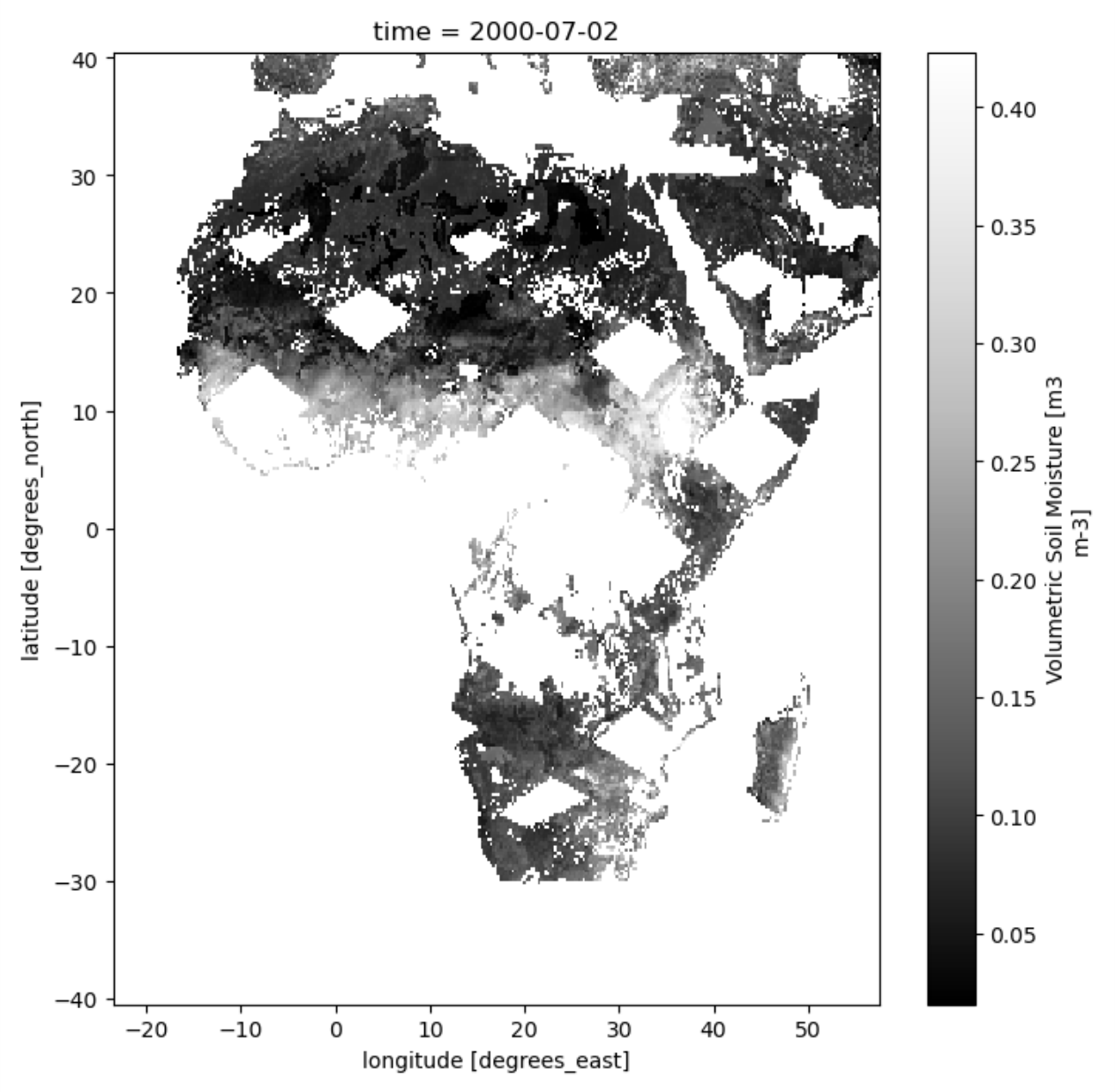
(continued from previous page)

```
↪ndarray>
    sm_uncertainty  (time, lat, lon) float32 dask.array<chunksize=(1, 324, 324), meta=np.
↪ndarray>
    t0              (time, lat, lon) datetime64[ns] dask.array<chunksize=(1, 324, 324),␣
↪meta=np.ndarray>
Attributes: (12/46)
    Conventions:                  CF-1.9
    cdm_data_type:                Grid
    comment:                      This dataset was produced with funding of t...
    contact:                      cci_sm_contact@eodc.eu
    creator_email:                cci_sm_developer@eodc.eu
    creator_name:                 Department of Geodesy and Geoinformation, V...
    ...                           ...
    time_coverage_start:          19781101T000000Z
    time_coverage_start_product:  19781101T000000Z
    title:                        ESA CCI Surface Soil Moisture COMBINED acti...
    tracking_id:                  dc6ade2c-e51b-4a94-81fa-751df95a85a6
    kerchunk_revision:            kr1.1
    kerchunk_creation_date:       031023T093359
```

... and we can plot the data.

[7]:
```
sub_cube.sm.sel(time='2000-07-01 12:00:00', method='nearest').plot.imshow(cmap='Greys_r',
↪ figsize=(8, 8))
```

[7]:
```
<matplotlib.image.AxesImage at 0x7f18c98810d0>
```



[ ]:

Finding Operations

This notebook serves to show how operations can be detected and how information about these operations can be shown.

[1]:
```
from IPython.display import JSON

from esa_climate_toolbox.core import get_op_meta_info
from esa_climate_toolbox.core import list_operations
```

Operations of the ESA Climate Toolbox can be listed by calling the `list_operations` command.

[2]:
```
list_operations()
```

[2]:
```
['adjust_spatial_attrs',
 'adjust_temporal_attrs',
 'aggregate_statistics',
 'anomaly_external',
 'anomaly_internal',
 'arithmetics',
 'climatology',
 'coregister',
 'data_frame_max',
 'data_frame_min',
 'data_frame_subset',
 'detect_outliers',
 'diff',
 'find_closest',
 'merge',
 'normalize',
 'query',
 'reduce',
 'resample',
 'select_features',
 'select_var',
 'subset_spatial',
 'subset_temporal',
 'subset_temporal_index',
 'temporal_aggregation',
 'tseries_mean',
 'tseries_point']
```

To get information about the required input parameters, you can ask for an operation's meta info and see its inputs. Under outputs, you will find what you will get from applying the operation.

[3]:
```
JSON(get_op_meta_info('coregister'))
```

[3]:
```
<IPython.core.display.JSON object>
```

[ ]:
```

```

Usage of Operations

This notebook serves to show how operations provided by the toolbox can be applied. For this purpose, the following aspects are covered:

- Access to ESA CCI Ozone and Cloud data (Atmosphere Mole Content of Ozone and Cloud Cover)
- Geometric adjustments (coregistration)
- Spatial (point, polygon) and temporal subsetting
- Visualisation of time series

Ingest data and create datasets

[1]:
```
from IPython.display import JSON

from xcube.core.store import new_data_store

import esa_climate_toolbox.ops as ops

from esa_climate_toolbox.core import get_op
from esa_climate_toolbox.core import get_op_meta_info
from esa_climate_toolbox.util.monitor import ConsoleMonitor
```

First, read in the ESA Climate Data Centre Data Store.

[2]:
```
data_store = new_data_store('esa-climate-data-centre')
```

Open an ozone dataset (see Notebook 1-ECT_General_Data_Access to find which options you have).

[3]:
```
ozone_ds = data_store.open_data('esacci.OZONE.mon.L3.NP.multi-sensor.multi-platform.
↪MERGED.fv0002.r1',
                                time_range=['2007-01-01', '2007-06-30'])
ozone_ds
```

[3]:
```
<xarray.Dataset>
Dimensions:           (time: 6, layers: 16, lat: 180, lon: 360,
                       air_pressure: 17, bnds: 2)
Coordinates:
  * air_pressure      (air_pressure) float32 1.013e+03 446.0 196.4 ... 0.05 0.01
  * lat               (lat) float32 -89.5 -88.5 -87.5 -86.5 ... 87.5 88.5 89.5
  * layers            (layers) int32 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
  * lon               (lon) float32 -179.5 -178.5 -177.5 ... 177.5 178.5 179.5
  * time              (time) datetime64[ns] 2007-01-16T12:00:00 ... 2007-06-16
    time_bnds         (time, bnds) datetime64[ns] dask.array<chunksize=(6, 2), meta=np.
↪ndarray>
Dimensions without coordinates: bnds
Data variables:
    O3_du             (time, layers, lat, lon) float32 dask.array<chunksize=(1, 16, 180,␣
↪360), meta=np.ndarray>
    O3_du_tot         (time, lat, lon) float32 dask.array<chunksize=(1, 180, 360),␣
↪meta=np.ndarray>
    O3_ndens          (time, air_pressure, lat, lon) float32 dask.array<chunksize=(1, 17,
↪ 180, 360), meta=np.ndarray>
    O3_vmr            (time, air_pressure, lat, lon) float32 dask.array<chunksize=(1, 17,
↪ 180, 360), meta=np.ndarray>
    O3e_du            (time, layers, lat, lon) float32 dask.array<chunksize=(1, 16, 180,␣
↪360), meta=np.ndarray>
    O3e_du_tot        (time, lat, lon) float32 dask.array<chunksize=(1, 180, 360),␣
↪meta=np.ndarray>
    O3e_ndens         (time, air_pressure, lat, lon) float32 dask.array<chunksize=(1, 17,
↪ 180, 360), meta=np.ndarray>
    O3e_vmr           (time, air_pressure, lat, lon) float32 dask.array<chunksize=(1, 17,
↪ 180, 360), meta=np.ndarray>
    surface_pressure  (time, lat, lon) float32 dask.array<chunksize=(1, 180, 360),␣
↪meta=np.ndarray>
```

(continues on next page)

(continued from previous page)

```
Attributes:
    Conventions:              CF-1.7
    title:                    esacci.OZONE.mon.L3.NP.multi-sensor.multi-platfo...
    date_created:             2024-02-29T21:03:01.536182
    processing_level:         L3
    time_coverage_start:      2007-01-01T00:00:00
    time_coverage_end:        2007-07-01T00:00:00
    time_coverage_duration:   P181DT0H0M0S
    history:                  [{'program': 'esa_climate_toolbox.ds.chunkstore...
```

Open a cloud dataset next.

[4]:
```
cloud_ds = data_store.open_data('esacci.CLOUD.mon.L3C.CLD_PRODUCTS.multi-sensor.multi-
↪platform.ATSR2-AATSR.3-0.r1',
                                time_range=['2007-01-01', '2007-06-30'])
cloud_ds
```

[4]:
```
<xarray.Dataset>
Dimensions:                          (time: 6, lat: 360, lon: 720, hist_phase: 2,
                                      hist1d_cer_bin_centre: 11,
                                      hist1d_cer_bin_border: 12,
                                      hist1d_cla_vis006_bin_centre: 13,
                                      hist1d_cla_vis006_bin_border: 14,
                                      hist1d_cla_vis008_bin_centre: 13,
                                      ...
                                      hist1d_cwp_bin_centre: 14,
                                      hist1d_cwp_bin_border: 15,
                                      hist2d_cot_bin_border: 14,
                                      hist2d_cot_bin_centre: 13,
                                      hist2d_ctp_bin_centre: 15,
                                      hist2d_ctp_bin_border: 16, bnds: 2)
Coordinates: (12/23)
  * hist1d_cer_bin_border            (hist1d_cer_bin_border) float32 0.0 ... 80.0
  * hist1d_cer_bin_centre            (hist1d_cer_bin_centre) float32 1.5 ... 70.0
  * hist1d_cla_vis006_bin_border     (hist1d_cla_vis006_bin_border) float32 0.0 ...
  * hist1d_cla_vis006_bin_centre     (hist1d_cla_vis006_bin_centre) float32 0.05...
  * hist1d_cla_vis008_bin_border     (hist1d_cla_vis008_bin_border) float32 0.0 ...
  * hist1d_cla_vis008_bin_centre     (hist1d_cla_vis008_bin_centre) float32 0.05...
    ...                               ...
  * hist2d_ctp_bin_centre            (hist2d_ctp_bin_centre) float32 45.5 ... 1...
  * hist_phase                       (hist_phase) int8 0 1
  * lat                              (lat) float32 -89.75 -89.25 ... 89.25 89.75
  * lon                              (lon) float32 -179.8 -179.2 ... 179.2 179.8
  * time                             (time) datetime64[ns] 2007-01-16T12:00:00 ...
    time_bnds                        (time, bnds) datetime64[ns] dask.array<chunksize=(6,␣
↪2), meta=np.ndarray>
Dimensions without coordinates: bnds
Data variables: (12/182)
    boa_lwdn                         (time, lat, lon) float32 dask.array<chunksize=(1, 360,␣
↪720), meta=np.ndarray>
    boa_lwdn_clr                     (time, lat, lon) float32 dask.array<chunksize=(1, 360,␣
↪720), meta=np.ndarray>
    boa_lwup                         (time, lat, lon) float32 dask.array<chunksize=(1, 360,␣
```

(continues on next page)

(continued from previous page)

```
↪720), meta=np.ndarray>
    boa_lwup_clr                (time, lat, lon) float32 dask.array<chunksize=(1, 360,␣
↪720), meta=np.ndarray>
    boa_swdn                    (time, lat, lon) float32 dask.array<chunksize=(1, 360,␣
↪720), meta=np.ndarray>
    boa_swdn_clr                (time, lat, lon) float32 dask.array<chunksize=(1, 360,␣
↪720), meta=np.ndarray>
    ...                          ...
    toa_swdn                    (time, lat, lon) float32 dask.array<chunksize=(1, 360,␣
↪720), meta=np.ndarray>
    toa_swup                    (time, lat, lon) float32 dask.array<chunksize=(1, 360,␣
↪720), meta=np.ndarray>
    toa_swup_clr                (time, lat, lon) float32 dask.array<chunksize=(1, 360,␣
↪720), meta=np.ndarray>
    toa_swup_hig                (time, lat, lon) float32 dask.array<chunksize=(1, 360,␣
↪720), meta=np.ndarray>
    toa_swup_low                (time, lat, lon) float32 dask.array<chunksize=(1, 360,␣
↪720), meta=np.ndarray>
    toa_swup_mid                (time, lat, lon) float32 dask.array<chunksize=(1, 360,␣
↪720), meta=np.ndarray>
Attributes:
    Conventions:            CF-1.7
    title:                  esacci.CLOUD.mon.L3C.CLD_PRODUCTS.multi-sensor.m...
    date_created:           2024-02-29T21:03:34.085371
    processing_level:       L3C
    time_coverage_start:    2007-01-01T00:00:00
    time_coverage_end:      2007-07-01T00:00:00
    time_coverage_duration: P181DT0H0M0S
    history:                [{'program': 'esa_climate_toolbox.ds.chunkstore...
```

Filter datasets to select the desired variables

The datasets are rather large, so we opt to make datasets that have only the variable we are interested in. The `select_var` op may be used for that.

[5]:
```
JSON(get_op_meta_info('select_var'))
```

[5]:
```
<IPython.core.display.JSON object>
```

[6]:
```
svar_op = get_op('select_var')
```

We choose the variable `cfc` (cloud area fraction) from the cloud dataset …

[7]:
```
cfc_ds = svar_op(ds=cloud_ds, var='cfc')
cfc_ds
```

[7]:
```
<xarray.Dataset>
Dimensions:                       (time: 6, lat: 360, lon: 720,
                                   hist1d_cer_bin_border: 12,
                                   hist1d_cer_bin_centre: 11,
                                   hist1d_cla_vis006_bin_border: 14,
                                   hist1d_cla_vis006_bin_centre: 13,
```

(continues on next page)

(continued from previous page)

```
                                hist1d_cla_vis008_bin_border: 14,
                                ...
                                hist1d_cwp_bin_centre: 14,
                                hist2d_cot_bin_border: 14,
                                hist2d_cot_bin_centre: 13,
                                hist2d_ctp_bin_border: 16,
                                hist2d_ctp_bin_centre: 15, hist_phase: 2,
                                bnds: 2)
Coordinates: (12/23)
  * hist1d_cer_bin_border         (hist1d_cer_bin_border) float32 0.0 ... 80.0
  * hist1d_cer_bin_centre         (hist1d_cer_bin_centre) float32 1.5 ... 70.0
  * hist1d_cla_vis006_bin_border  (hist1d_cla_vis006_bin_border) float32 0.0 ...
  * hist1d_cla_vis006_bin_centre  (hist1d_cla_vis006_bin_centre) float32 0.05...
  * hist1d_cla_vis008_bin_border  (hist1d_cla_vis008_bin_border) float32 0.0 ...
  * hist1d_cla_vis008_bin_centre  (hist1d_cla_vis008_bin_centre) float32 0.05...
    ...                            ...
  * hist2d_ctp_bin_centre         (hist2d_ctp_bin_centre) float32 45.5 ... 1...
  * hist_phase                    (hist_phase) int8 0 1
  * lat                           (lat) float32 -89.75 -89.25 ... 89.25 89.75
  * lon                           (lon) float32 -179.8 -179.2 ... 179.2 179.8
  * time                          (time) datetime64[ns] 2007-01-16T12:00:00 ...
    time_bnds                     (time, bnds) datetime64[ns] dask.array<chunksize=(6,␣
↪2), meta=np.ndarray>
Dimensions without coordinates: bnds
Data variables:
    cfc                           (time, lat, lon) float32 dask.array<chunksize=(1, 360,␣
↪720), meta=np.ndarray>
Attributes:
    Conventions:              CF-1.7
    title:                    esacci.CLOUD.mon.L3C.CLD_PRODUCTS.multi-sensor.m...
    date_created:             2024-02-29T21:03:34.085371
    processing_level:         L3C
    time_coverage_start:      2007-01-01T00:00:00
    time_coverage_end:        2007-07-01T00:00:00
    time_coverage_duration:   P181DT0H0M0S
    history:                  [{'program': 'esa_climate_toolbox.ds.chunkstore...
```

... and `O3_du_tot` (the atmosphere mole content of ozone) from the ozone dataset.

[8]:
```
ozone_tot_ds = svar_op(ds=ozone_ds, var='O3_du_tot')
ozone_tot_ds
```

[8]:
```
<xarray.Dataset>
Dimensions:       (time: 6, lat: 180, lon: 360, air_pressure: 17, layers: 16,
                   bnds: 2)
Coordinates:
  * air_pressure  (air_pressure) float32 1.013e+03 446.0 196.4 ... 0.05 0.01
  * lat           (lat) float32 -89.5 -88.5 -87.5 -86.5 ... 86.5 87.5 88.5 89.5
  * layers        (layers) int32 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
  * lon           (lon) float32 -179.5 -178.5 -177.5 ... 177.5 178.5 179.5
  * time          (time) datetime64[ns] 2007-01-16T12:00:00 ... 2007-06-16
    time_bnds     (time, bnds) datetime64[ns] dask.array<chunksize=(6, 2), meta=np.
↪ndarray>
```

(continues on next page)

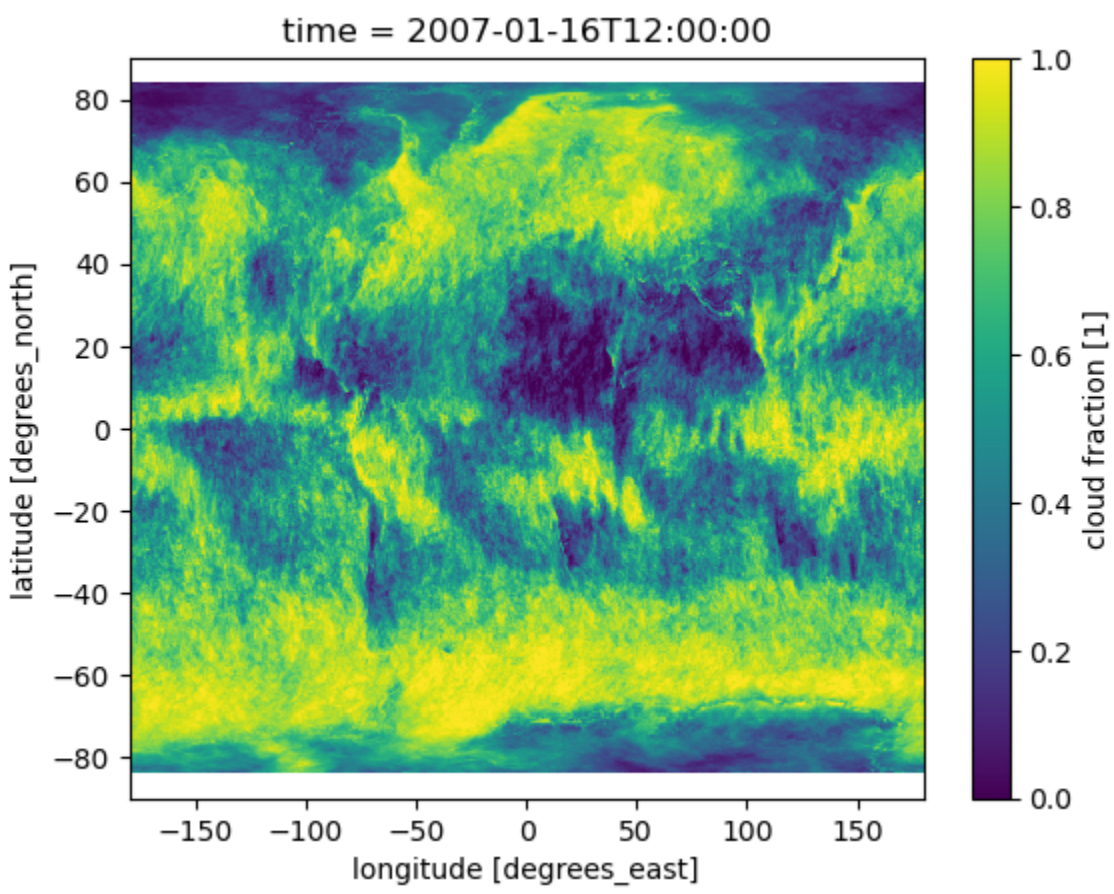
(continued from previous page)

```
Dimensions without coordinates: bnds
Data variables:
    O3_du_tot     (time, lat, lon) float32 dask.array<chunksize=(1, 180, 360), meta=np.
↪ndarray>
Attributes:
    Conventions:              CF-1.7
    title:                    esacci.OZONE.mon.L3.NP.multi-sensor.multi-platfo...
    date_created:             2024-02-29T21:03:01.536182
    processing_level:         L3
    time_coverage_start:      2007-01-01T00:00:00
    time_coverage_end:        2007-07-01T00:00:00
    time_coverage_duration:   P181DT0H0M0S
    history:                  [{'program': 'esa_climate_toolbox.ds.chunkstore...
```

Plot the first time slices of the datasets

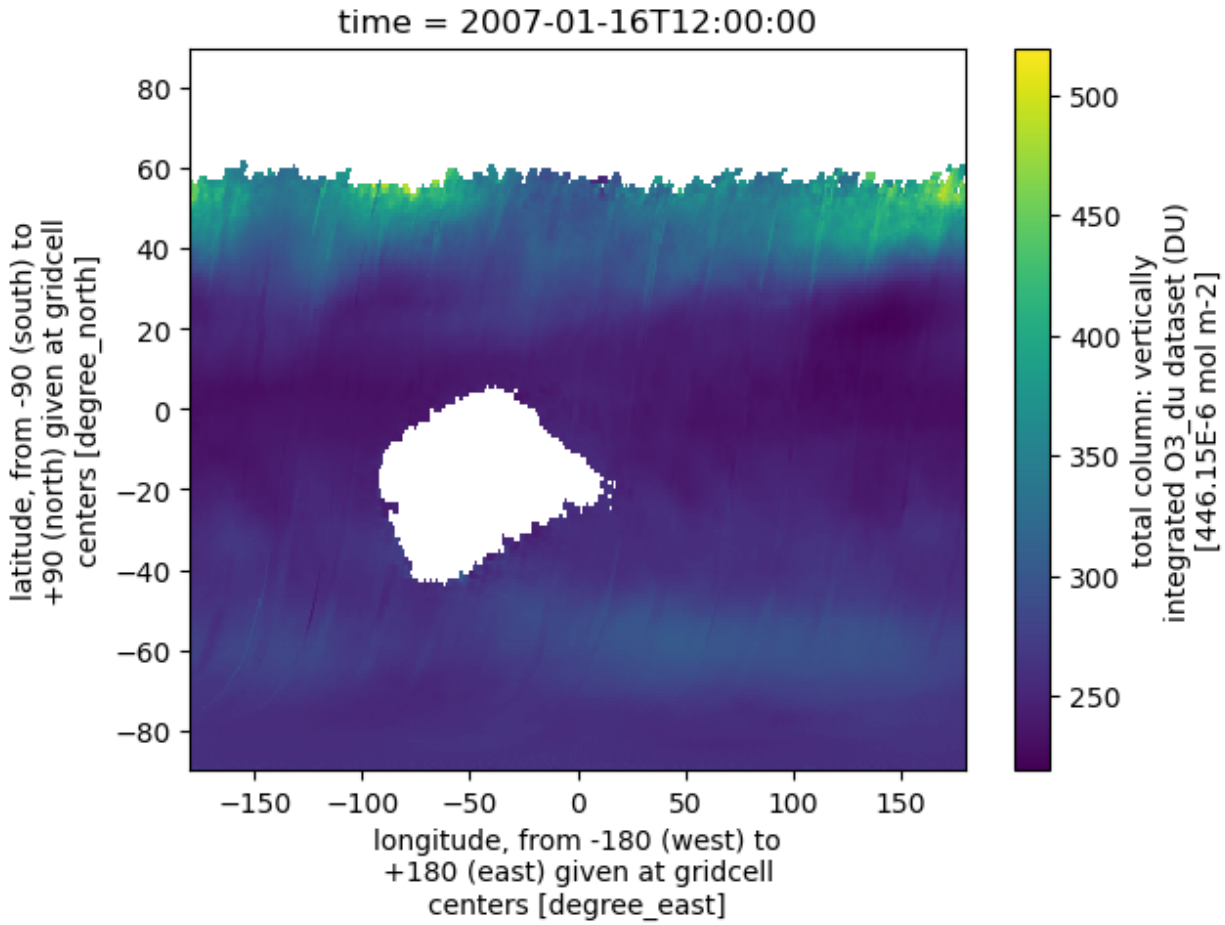
[9]:
```
%matplotlib inline

cfc_ds.cfc.isel(time=0).plot()
```

[9]:
```
<matplotlib.collections.QuadMesh at 0x7f423a14cf90>
```



[10]:
```
ozone_tot_ds.O3_du_tot.isel(time=0).plot()
```

[10]:
```
<matplotlib.collections.QuadMesh at 0x7f423a298f10>
```



Co-register datasets by resampling

We see that though both datasets are global, they are in different spatial resolutions:

[11]:
```
print(cfc_ds['cfc'].shape)
print(ozone_tot_ds['O3_du_tot'].shape)
```

```
(6, 360, 720)
(6, 180, 360)
```

To bring them to the same resolution, we use the `coregister` function.

[12]:
```
JSON(get_op_meta_info('coregister'))
```

[12]:
```
<IPython.core.display.JSON object>
```

[13]:
```
coregister_op = get_op('coregister')
```

We get the cloud dataset to the resolution of the ozone dataset.

[14]:
```
cfc_res_ds = coregister_op(ozone_tot_ds, cfc_ds)
```

Check the variable shapes again to see we were successful:

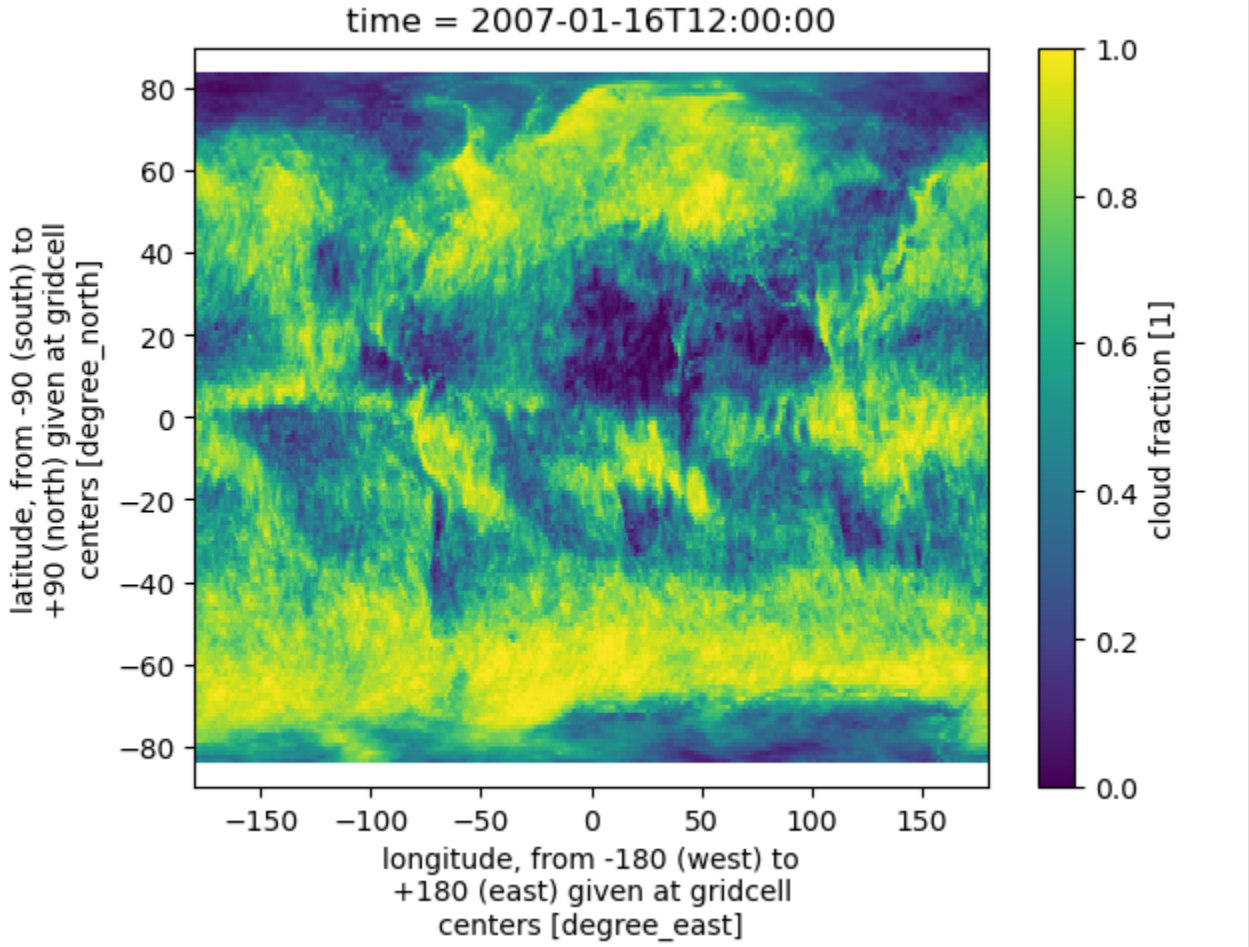
[15]:
```
print(cfc_res_ds['cfc'].shape)
print(ozone_tot_ds['03_du_tot'].shape)
```

```
(6, 180, 360)
(6, 180, 360)
```

And plot again:

[16]:
```
cfc_res_ds.cfc.isel(time=0).plot()
```

[16]:
```
<matplotlib.collections.QuadMesh at 0x7f4240fa0fd0>
```



Select the desired spatial region

We want to continue our analysis with a subset, say, Africa. For this we use `subset_spatial`.

[17]:
```
JSON(get_op_meta_info('subset_spatial'))
```

[17]:
```
<IPython.core.display.JSON object>
```

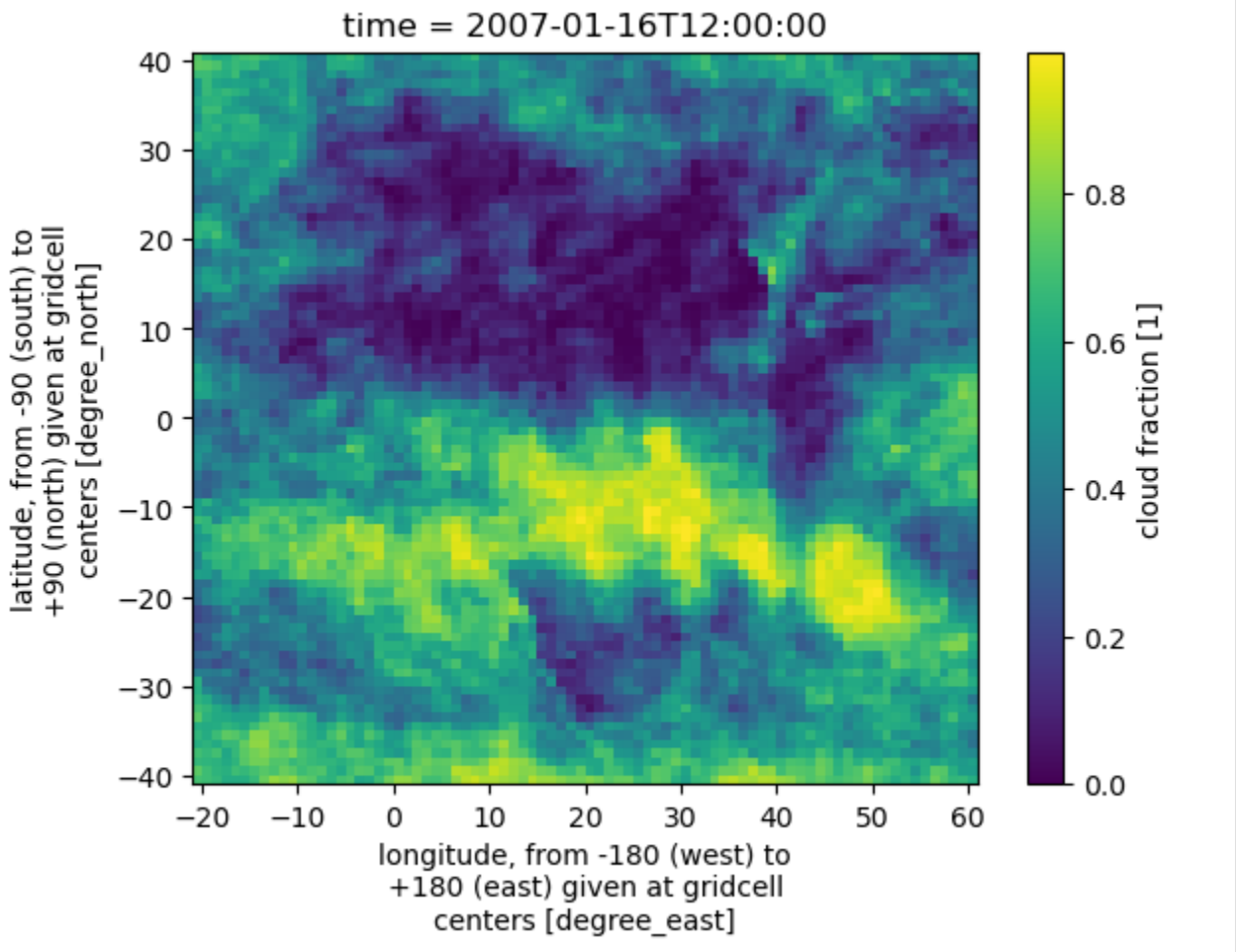
[18]:
```
subset_spatial_op = get_op('subset_spatial')
```

[19]:
```
africa = '-20.0, -40.0, 60.0, 40.0'
# 'lon_min, lat_min, lon_max, lat_max'
cfc_africa_res_ds = subset_spatial_op(cfc_res_ds, africa)
ozone_tot_africa_ds = subset_spatial_op(ozone_tot_ds, africa)
```

This is a good time to compare the plots of the resampled subset and the original plot to see the effects of our work:

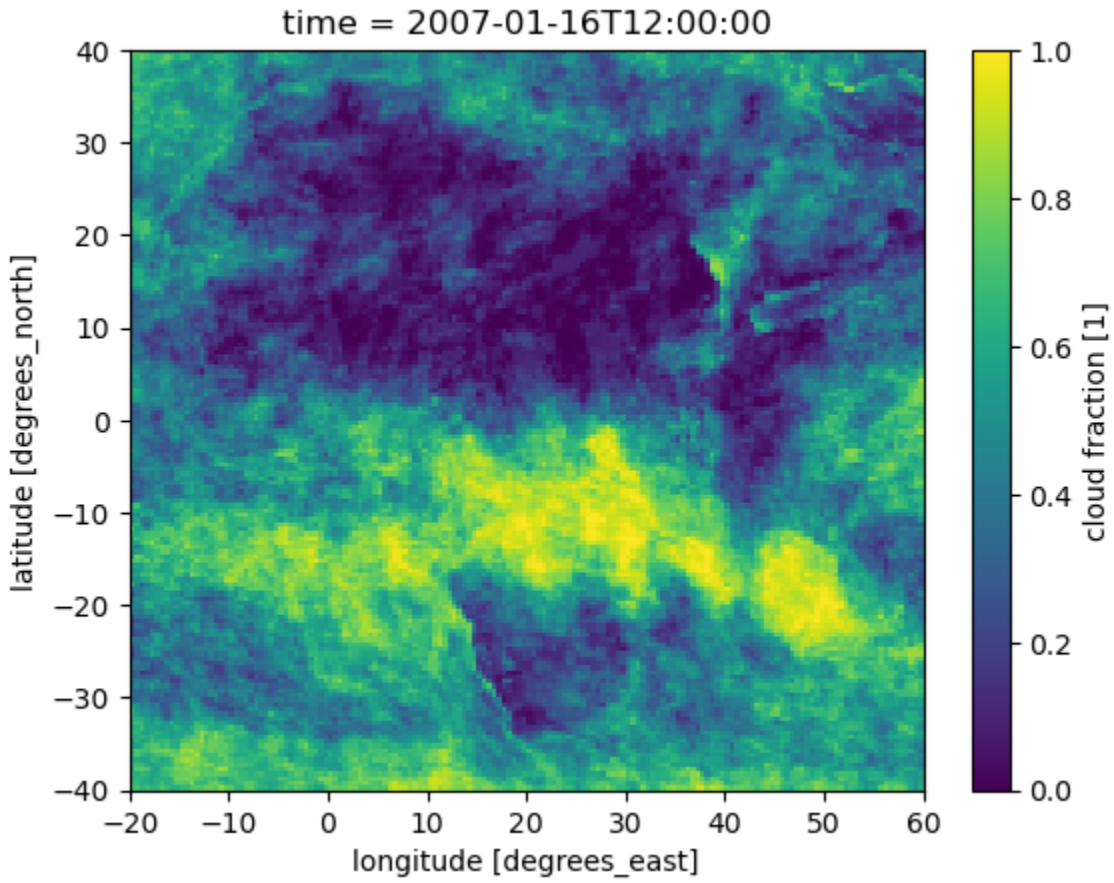
[20]:
```
cfc_africa_res_ds.cfc.isel(time=0).plot()
```

[20]:
```
<matplotlib.collections.QuadMesh at 0x7f42398b8f10>
```



[21]:
```
cfc_ds.cfc.isel(time=0).sel({'lat': slice(-40, 40), 'lon': slice(-20, 60)}).plot()
```

[21]:
```
<matplotlib.collections.QuadMesh at 0x7f42396cce50>
```



Select the desired temporal region

We may also create a temporal subset:

[22]:
```
JSON(get_op_meta_info('subset_temporal'))
```

[22]:
```
<IPython.core.display.JSON object>
```

[23]:
```
subset_temporal_op = get_op('subset_temporal')
```

[24]:
```
cfc_africa_res_janmar_ds = subset_temporal_op(cfc_africa_res_ds, '2007-01-01, 2007-03-31
↪')
ozone_tot_africa_janmar_ds = subset_temporal_op(ozone_tot_africa_ds, '2007-01-01, 2007-
↪03-31')
```

[25]:
```
print(cfc_africa_res_janmar_ds.time)
```

```
<xarray.DataArray 'time' (time: 3)>
array(['2007-01-16T12:00:00.000000000', '2007-02-15T00:00:00.000000000',
       '2007-03-16T12:00:00.000000000'], dtype='datetime64[ns]')
Coordinates:
```

(continues on next page)

(continued from previous page)

```
  * time     (time) datetime64[ns] 2007-01-16T12:00:00 ... 2007-03-16T12:00:00
Attributes:
    standard_name:  time
    bounds:         time_bnds
```

[26]:
```
print(ozone_tot_africa_janmar_ds.time)
```

```
<xarray.DataArray 'time' (time: 3)>
array(['2007-01-16T12:00:00.000000000', '2007-02-15T00:00:00.000000000',
       '2007-03-16T12:00:00.000000000'], dtype='datetime64[ns]')
Coordinates:
  * time     (time) datetime64[ns] 2007-01-16T12:00:00 ... 2007-03-16T12:00:00
Attributes:
    standard_name:  time
    bounds:         time_bnds
```

Retrieve and plot timeseries

Finally, we can create time series from these datasets. Operation `tseries_point` can be used to get a time series for a given point, `tseries_mean` will get the mean and standard deviation across the dataset.

[27]:
```
JSON(get_op_meta_info('tseries_point'))
```

[27]:
```
<IPython.core.display.JSON object>
```

[28]:
```
JSON(get_op_meta_info('tseries_mean'))
```

[28]:
```
<IPython.core.display.JSON object>
```

[29]:
```
tseries_point_op = get_op('tseries_point')
tseries_mean_op = get_op('tseries_mean')
```

[30]:
```
ozone_ts_point = tseries_point_op(ozone_tot_africa_janmar_ds, point='50, 50')
cfc_ts_point = tseries_point_op(cfc_africa_res_janmar_ds, point='50, 50')
ozone_ts_mean = tseries_mean_op(ozone_tot_africa_janmar_ds, var='O3_du_tot')
cfc_ts_mean = tseries_mean_op(cfc_africa_res_janmar_ds, var='cfc')
```

[31]:
```
print(ozone_ts_mean)
```

```
<xarray.Dataset>
Dimensions:         (air_pressure: 17, lat: 82, layers: 16, lon: 82, time: 3,
                     bnds: 2)
Coordinates:
  * air_pressure    (air_pressure) float32 1.013e+03 446.0 196.4 ... 0.05 0.01
  * lat             (lat) float32 -40.5 -39.5 -38.5 -37.5 ... 38.5 39.5 40.5
  * layers          (layers) int32 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
  * lon             (lon) float32 -20.5 -19.5 -18.5 -17.5 ... 58.5 59.5 60.5
  * time            (time) datetime64[ns] 2007-01-16T12:00:00 ... 2007-03-16T...
    time_bnds       (time, bnds) datetime64[ns] dask.array<chunksize=(3, 2), meta=np.
↪ndarray>
Dimensions without coordinates: bnds
```

(continues on next page)

(continued from previous page)

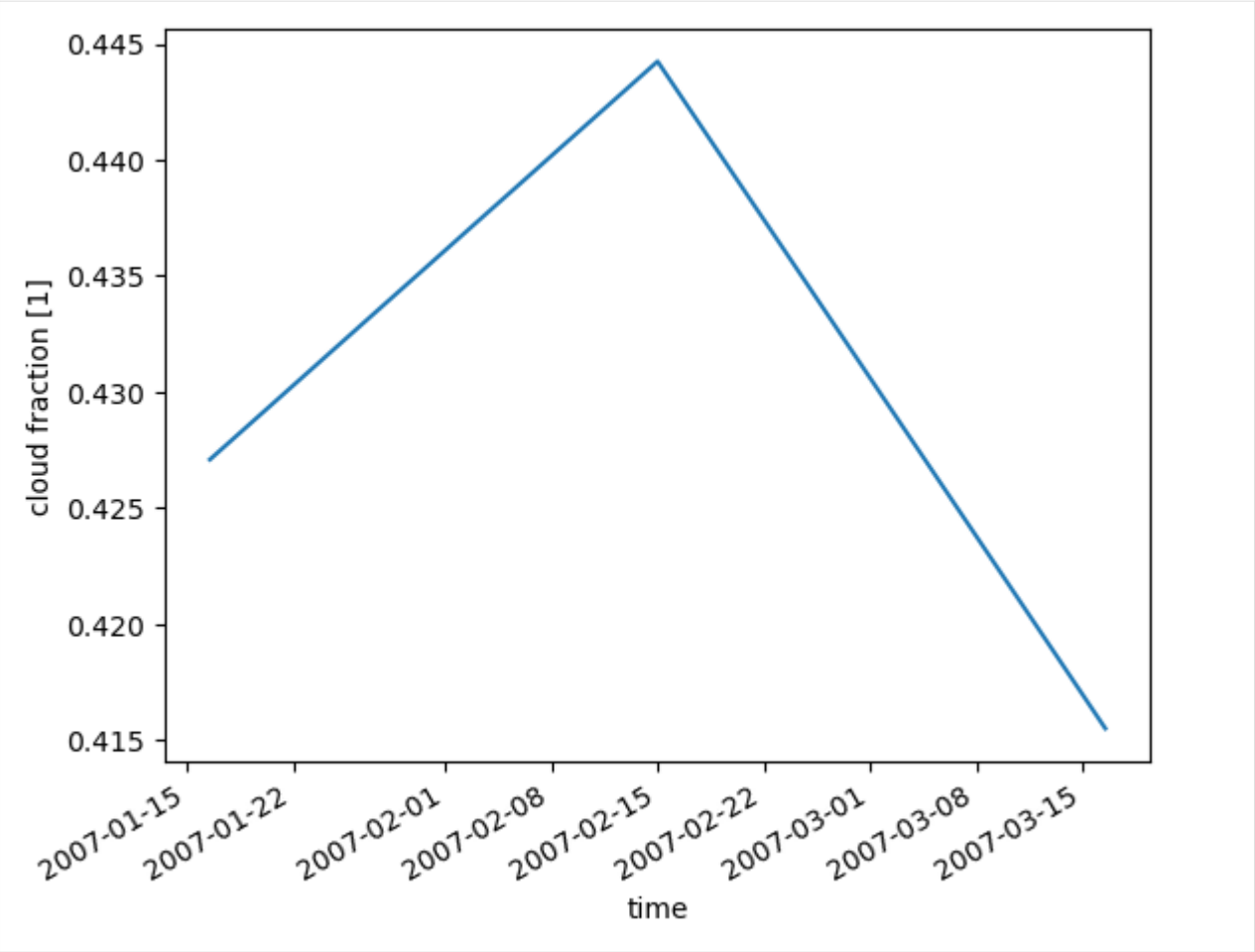
```
Data variables:
    O3_du_tot_mean  (time) float32 dask.array<chunksize=(1,), meta=np.ndarray>
    O3_du_tot_std   (time) float32 dask.array<chunksize=(1,), meta=np.ndarray>
Attributes: (12/19)
    Conventions:                CF-1.7
    title:                      esacci.OZONE.mon.L3.NP.multi-sensor.multi-pla...
    date_created:               2024-02-29T21:03:01.536182
    processing_level:           L3
    time_coverage_start:        2007-01-01T00:00:00
    time_coverage_end:          2007-04-01T00:00:00
    ...                         ...
    geospatial_lat_min:         -41.0
    geospatial_lat_max:         41.0
    geospatial_lat_units:       degree_north
    geospatial_bounds_crs:      CRS84
    geospatial_bounds:          POLYGON((-21.0 -41.0, -21.0 41.0, 61.0 41.0, ...
    time_coverage_resolution:   P1M
```

[32]:
```
print(cfc_ts_mean)
```

```
<xarray.Dataset>
Dimensions:   (lat: 82, lon: 82, time: 3)
Coordinates:
  * lat       (lat) float32 -40.5 -39.5 -38.5 -37.5 ... 37.5 38.5 39.5 40.5
  * lon       (lon) float32 -20.5 -19.5 -18.5 -17.5 ... 57.5 58.5 59.5 60.5
  * time      (time) datetime64[ns] 2007-01-16T12:00:00 ... 2007-03-16T12:00:00
Data variables:
    cfc_mean  (time) float32 dask.array<chunksize=(1,), meta=np.ndarray>
    cfc_std   (time) float32 dask.array<chunksize=(1,), meta=np.ndarray>
Attributes: (12/19)
    Conventions:                CF-1.7
    title:                      esacci.CLOUD.mon.L3C.CLD_PRODUCTS.multi-senso...
    date_created:               2024-02-29T21:03:34.085371
    processing_level:           L3C
    time_coverage_start:        2007-01-16T12:00:00
    time_coverage_end:          2007-03-16T12:00:00
    ...                         ...
    geospatial_lat_min:         -41.0
    geospatial_lat_max:         41.0
    geospatial_lat_units:       degree_north
    geospatial_bounds_crs:      CRS84
    geospatial_bounds:          POLYGON((-21.0 -41.0, -21.0 41.0, 61.0 41.0, ...
    time_coverage_resolution:   P1M
```

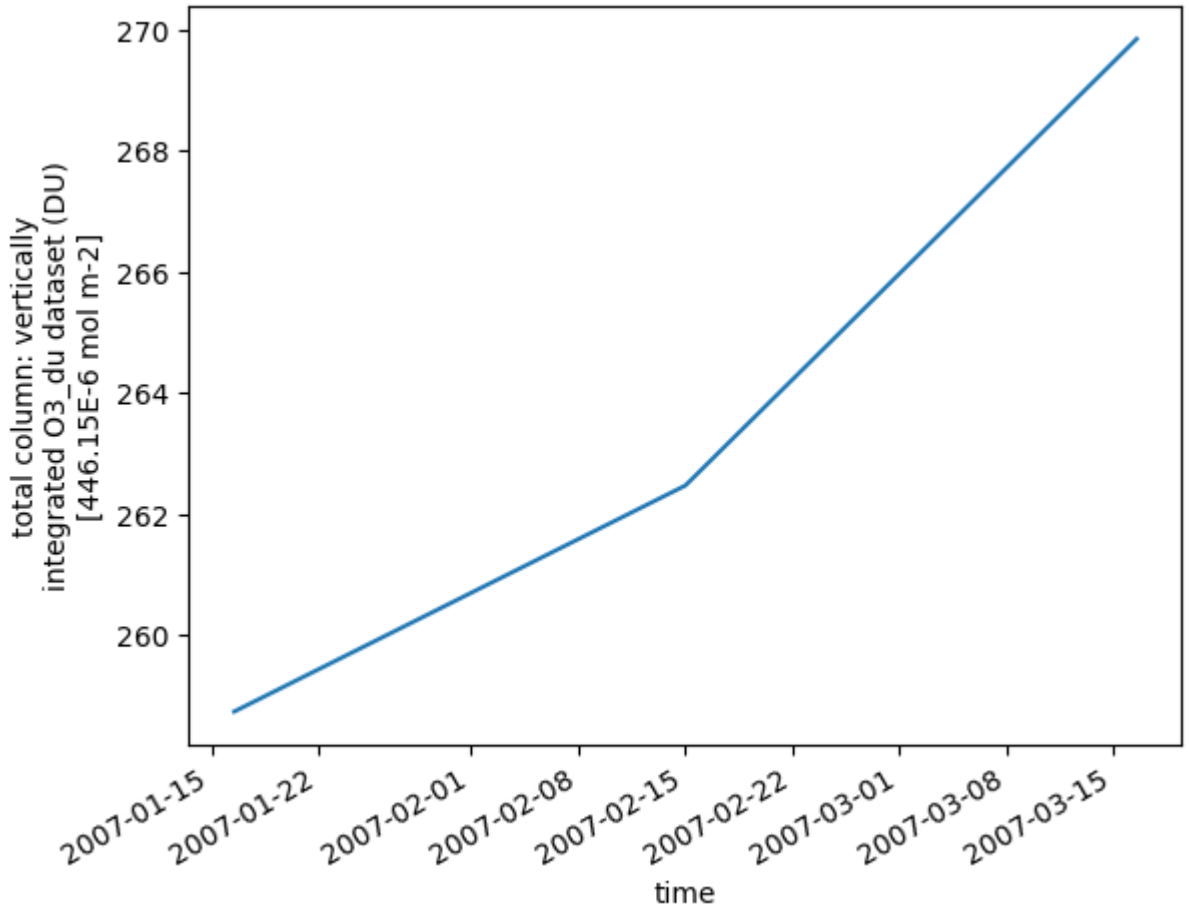
[33]:
```
cfc_ts_mean.cfc_mean.plot()
```

[33]:
```
[<matplotlib.lines.Line2D at 0x7f4239627190>]
```



[34]:
```
ozone_ts_mean.O3_du_tot_mean.plot()
```

[34]:
```
[<matplotlib.lines.Line2D at 0x7f4239843e90>]
```



[ ]:

6.5 API Reference

6.5.1 Datasets and Datastores

Exploring Data

esa_climate_toolbox.core.**find_data_store**(*ds_id: str*) → Tuple[str | None, DataStore | None]

Find the data store that includes the given *ds_id*. This will raise an exception if the *ds_id* is given in more than one data store.

Parameters
ds_id – A data source identifier.

Returns
All data sources matching the given constrains.

esa_climate_toolbox.core.**get_store**(*store_id: str*)

Returns the data store of the given name. :param store_id: The name of the store should have.

Returns
A data store

esa_climate_toolbox.core.**get_search_params**(*store_id: str*, *data_type: str*) → Dict

Returns potential search parameters that can be used to search for datasets in a data store.

Parameters

- **store_id** – The id of the store which shall be searched for data
- **data_type** – An optional data type to specify the type of data to be searched

Returns
A dictionary containing search parameters

esa_climate_toolbox.core.**list_datasets**(*store_id: str = None*, *data_type: str | None | type | DataType = None*, *include_attrs: Container[str] = None*) → List[str] | List[Tuple[str, Dict[str, Any]]]

Returns the names of datasets of a given store.

Parameters

- **store_id** – The name of the data store
- **data_type** – A datatype that may be provided to restrict the search, e.g., 'dataset' or 'geodataframe'
- **include_attrs** – An optional list to retrieve additional meta information if required.

Returns
Either a list of the dataset names within the store, or a list of tuples, each consisting of a name and a dictionary with additional information.

esa_climate_toolbox.core.**list_ecvs**() → List[str]

Returns a list of names of essential climate variables served by the ESA Climate Toolbox.

Returns
A list of names of essential climate variables.

esa_climate_toolbox.core.**list_ecv_datasets**(*ecv: str*, *data_type: str | None | type | DataType = None*, *include_attrs: Container[str] = None*) → List[str] | List[Tuple[str, Dict[str, Any]]]

Returns the names of datasets for a given essential climate variable.

Parameters

- **ecv** – The name of the essential climate variable
- **data_type** – A datatype that may be provided to restrict the search, e.g., 'dataset' or 'geodataframe'
- **include_attrs** – An optional list to retrieve additional meta information if required.

Returns
Either a list of dataset names for the given ecv, or a list of tuples, each consisting of a name and a dictionary with additional information.

esa_climate_toolbox.core.**list_stores**() → List[str]

Lists the names of the data stores which are provided by Returns meta information about an operation.

Parameters

- **op_name** – The name of the operation for which meta information shall be provided.

- **op_registry** – An optional OpRegistry, in case the default one should not be used.

Returns
A dictionary representation of an operator's meta info, providing information about input parameters and the expected output.

esa_climate_toolbox.core.**search**(*store_id: str*, *data_type: str = None*, ***search_params*) → List[Dict]

Searches in a data store for data that meet the given search criteria.

Parameters

- **store_id** – The id of the store which shall be searched for data
- **data_type** – An optional data type to specify the type of data to be searched
- **search_params** – Store-specific additional search parameters

Returns
A list of dictionaries providing detailed information about the data that meet the specified criteria.

Managing Data

esa_climate_toolbox.core.**add_local_store**(*root: str*, *store_id: str = None*, *max_depth: int = 1*, *read_only: bool = False*, *includes: str = None*, *excludes: str = None*, *title: str = None*, *description: str = None*, *persist: bool = True*) → str

Registers a new data store in the ESA Climate Toolbox to access locally stored data.

Parameters

- **root** – The path to the data.
- **store_id** – The name the store should have. There must not already be a store of the same name.
- **max_depth** – The maximum level of sub-directories that will be browsed for data. Default is 1, i.e., only the data located in the root path will be considered.
- **read_only** – Whether the store is read-only. Default is false.
- **includes** – Allows to specify a pattern about which data shall be served by the store (e.g., aerosol*.nc)
- **excludes** – Allows to specify a pattern about which data shall not be served by the store (e.g., aerosol*.zarr)
- **title** – An optional title for the data store
- **description** – An optional description of the data store
- **persist** – Whether the data store shall be registered permanently, otherwise it will only be for this session. Default is True.

Returns
The id of the newly created store.

esa_climate_toolbox.core.**add_store**(*store_type: str*, *store_params: Mapping[str, Any] = None*, *store_id: str = None*, *title: str = None*, *description: str = None*, *user_data: Any = None*, *persist: bool = True*) → str

Registers a new data store in the ESA Climate Toolbox. This function allows to also specify non-local data stores.

Parameters

- **store_type** – The type of data store to create, e.g., 's3'.
- **store_params** – A mapping containing store-specific parameters which are required to initiate the store.
- **store_id** – The name the store should have. There must not already be a store of the same name.
- **title** – An optional title for the data store
- **description** – An optional description of the data store
- **user_data** – Any additional user data
- **persist** – Whether the data store shall be registered permanently, otherwise it will only be for this session. Default is True.

Returns
The id of the newly created store.

esa_climate_toolbox.core.**remove_store**(*store_id: str*, *persist: bool = True*)

Removes a store from the internal store registry. No actual data will be deleted.

Parameters

- **store_id** – The name of the store to be removed
- **persist** – Whether the data store shall be unregistered permanently, otherwise it will only be for this session. Default is True.

Returns
Either a list of dataset names for the given ecv, or a list of tuples, each consisting of a name and a dictionary with additional information.

Reading and Writing Data

esa_climate_toolbox.core.**get_output_store_id**() → str | None

Returns the name of the store that by default will be used for writing.

Returns
The id of the default output store.

esa_climate_toolbox.core.**get_supported_formats**(*data: Any*, *store_id: str*) → List[str]

Returns the list of formats to which the store at the given store_id may write the given data.

Parameters

- **data** – The data which shall be written
- **store_id** – The id of the store to which the data shall be written

Returns
A list of supported output formats

esa_climate_toolbox.core.**open_data**(*dataset_id: str*, *time_range: Tuple[str, str] | Tuple[datetime, datetime] | Tuple[date, date] | str = None*, *region: Polygon | List[Tuple[float, float]] | str | Tuple[float, float, float, float] = None*, *var_names: List[str] | str = None*, *data_store_id: str = None*, *monitor: Monitor = Monitor.NONE*) → Tuple[Any, str]

Open a dataset from a data store.

Parameters

- **dataset_id** – The identifier of the dataset. Must not be empty.
- **time_range** – An optional time constraint comprising start and end date. If given, it must be a `TimeRangeLike`.
- **region** – An optional region constraint. If given, it must be a `PolygonLike`.
- **var_names** – Optional names of variables to be included. If given, it must be a `VarNamesLike`.
- **data_store_id** – Optional data store identifier. If given, *ds_id* will only be looked up from the specified data store.
- **monitor** – A progress monitor

Returns
A tuple consisting of a new dataset instance and its id

esa_climate_toolbox.core.**set_output_store**(*store_id: str*)

Specifies which store shall be the standard output store. This value is not persisted and must be set every session.

Parameters
store_id – The name of the store that shall be the output store.

esa_climate_toolbox.core.**write_data**(*data: Any*, *data_id: str = None*, *store_id: str = None*, *format_id: str = None*, *replace: bool = False*, *monitor: Monitor = Monitor.NONE*) → str

Writes data

Parameters

- **data** – The data which shall be written
- **data_id** – A data id under which the data shall be written to the store. If not given, a data id will be created.
- **store_id** – The id of the store to which the data shall be written. If none is given, the data is written to the standard output store.
- **format_id** – A format that shall be used to write the data. If none is given, the data will be written in the default format for the data type, e.g., 'zarr' for datasets.
- **replace** – Whether a dataset with the same id in the store shall be replaced. If False, an exception will be raised. Default is False.
- **monitor** – A monitor to measure the writing process

Returns
The data id under which the data can be accessed from the store.

6.5.2 Operations

Aggregation

esa_climate_toolbox.ops.**climatology**(**args*, *monitor: Monitor = Monitor.NONE*, ***kwargs*)

Create a 'mean over years' dataset by averaging the values of the given input dataset over all years. The output is a climatological dataset with the same resolution as the input dataset. E.g. a daily input dataset will create a daily climatology consisting of 365 days, a monthly input dataset will create a monthly climatology, etc.

Seasonal input datasets must have matching seasons over all years denoted by the same date each year. E.g., first date of each quarter. The output dataset will then be a seasonal climatology where each season is denoted with the same date as in the input dataset.

For further information on climatological datasets, see http://cfconventions.org/cf-conventions/v1.6.0/cf-conventions.html#climatological-statistics

Parameters

- **ds** – A dataset to average
- **var** – If given, only these variables will be preserved in the resulting dataset
- **monitor** – A progress monitor

Returns

A climatological long term average dataset

esa_climate_toolbox.ops.**reduce**(**args*, *monitor:* Monitor = *Monitor.NONE*, ***kwargs*)

Reduce the given variables of the given dataset along the given dimensions. If no variables are given, all variables of the dataset will be reduced. If no dimensions are given, all dimensions will be reduced. If no variables have been given explicitly, it can be set that only variables featuring numeric values should be reduced.

Parameters

- **ds** – Dataset to reduce
- **var** – Variables in the dataset to reduce
- **dim** – Dataset dimensions along which to reduce
- **method** – reduction method
- **monitor** – A progress monitor

esa_climate_toolbox.ops.**temporal_aggregation**(**args*, *monitor:* Monitor = *Monitor.NONE*, ***kwargs*)

Perform aggregation of dataset according to the given aggregation *method* and time period *period*.

Note that the operation does not perform weighting. Depending on the combination of input and output resolutions, as well as aggregation method, the resulting dataset might yield unexpected results.

The possible values if *period* are the offset-aliases supported by the Pandas package: http://pandas.pydata.org/pandas-docs/stable/timeseries.html#offset-aliases

Some examples for *period* values:

- 'QS-DEC' will result in a dataset aggregated to DJF, MAM, JJA, SON seasons, each denoted by the first date of the season.
- 'QS-JUN' produces an output dataset on a quarterly resolution where the year ends in 1st of June and each quarter is denoted by its first date.
- '8MS' produces an output dataset on an eight-month resolution where each period is denoted by the first date. Note that such periods will not be consistent over years.
- '8D' produces a dataset on an eight day resolution.

Parameters

- **ds** – Dataset to aggregate
- **method** – Aggregation method
- **period** – Aggregation time period

Returns
Aggregated dataset

Anomalies

esa_climate_toolbox.ops.**anomaly_external**(**args*, *monitor:* Monitor = *Monitor.NONE*, ***kwargs*)

Calculate anomaly with external reference data, for example, a climatology. The given reference dataset is expected to consist of 12 time slices, one for each month.

The returned dataset will contain the variable names found in both - the reference and the given dataset. Names found in the given dataset, but not in the reference, will be dropped from the resulting dataset. The calculated anomaly will be against the corresponding month of the reference data. E.g. January against January, etc.

In case spatial extents differ between the reference and the given dataset, the anomaly will be calculated on the intersection.

Parameters

- **ds** – The dataset to calculate anomalies from
- **file** – Path to reference data file
- **transform** – Apply the given transformation before calculating the anomaly. For supported operations see help on 'ds_arithmetics' operation.
- **monitor** – a progress monitor.

Returns
The anomaly dataset

esa_climate_toolbox.ops.**anomaly_internal**(**args*, *monitor:* Monitor = *Monitor.NONE*, ***kwargs*)

Calculate anomaly using as reference data the mean of an optional region and time slice from the given dataset. If no time slice/spatial region is given, the operation will calculate anomaly using the mean of the whole dataset as the reference.

This is done for each data array in the dataset. :param ds: The dataset to calculate anomalies from :param time_range: Time range to use for reference data :param region: Spatial region to use for reference data :param monitor: a progress monitor. :return: The anomaly dataset

Arithmetics

esa_climate_toolbox.ops.**arithmetics**(**args*, *monitor:* Monitor = *Monitor.NONE*, ***kwargs*)

Do arithmetic operations on the given dataset by providing a list of arithmetic operations and the corresponding constant. The operations will be applied to the dataset in the order in which they appear in the list. For example: 'log,+5,-2,/3,*2'

Currently supported arithmetic operations: log,log10,log2,log1p,exp,+,-,/,*

where:
log - natural logarithm log10 - base 10 logarithm log2 - base 2 logarithm log1p - log(1+x) exp - the exponential

The operations will be applied element-wise to all arrays of the dataset.

Parameters

- **ds** – The dataset to which to apply arithmetic operations
- **ops** – A comma separated list of arithmetic operations to apply

- **monitor** – a progress monitor.

Returns
The dataset with given arithmetic operations applied

esa_climate_toolbox.ops.**diff**(**args*, *monitor:* Monitor = *Monitor.NONE*, ***kwargs*)

Calculate the difference of two datasets (ds - ds2). This is done by matching variable names in the two datasets against each other and taking the difference of matching variables.

If lat/lon/time extents differ between the datasets, the default behavior is to take the intersection of the datasets and run subtraction on that. However, broadcasting is possible. E.g. ds(lat/lon/time) - ds(lat/lon) is valid. In this case the subtrahend will be stretched to the size of ds(lat/lon/time) so that it can be subtracted. This also works if the subtrahend is a single time slice of arbitrary temporal position. In this case, the time dimension will be squeezed out leaving a lat/lon dataset.

Parameters

- **ds** – The minuend dataset
- **ds2** – The subtrahend dataset
- **monitor** – a progress monitor.

Returns
The difference dataset

Coregistration

esa_climate_toolbox.ops.**coregister**(**args*, *monitor:* Monitor = *Monitor.NONE*, ***kwargs*)

Perform coregistration of two datasets by resampling the replica dataset onto the grid of the primary. If upsampling has to be performed, this is achieved using interpolation, if downsampling has to be performed, the pixels of the replica dataset are aggregated to form a coarser grid.

The returned dataset will contain the lat/lon intersection of provided primary and replica datasets, resampled unto the primary grid frequency.

This operation works on datasets whose spatial dimensions are defined on pixel-registered grids that are equidistant in lat/lon coordinates, i.e., data points define the middle of a pixel and pixels have the same size across the dataset.

This operation will resample all variables in a dataset, as the lat/lon grid is defined per dataset. It works only if all variables in the dataset have lat and lon as dimensions.

For an overview of downsampling/upsampling methods used in this operation, please see https://github.com/CAB-LAB/gridtools

Whether upsampling or downsampling has to be performed is determined automatically based on the relationship of the grids of the provided datasets.

Parameters

- **ds_primary** – The dataset whose grid is used for resampling
- **ds_replica** – The dataset that will be resampled
- **method_us** – Interpolation method to use for upsampling.
- **method_ds** – Interpolation method to use for downsampling.
- **monitor** – a progress monitor.

Returns
The replica dataset resampled on the grid of the primary

Data Frame Operations

esa_climate_toolbox.ops.**aggregate_statistics**(**args*, *monitor:* Monitor = *Monitor.NONE*, ***kwargs*)

Aggregate columns into count, mean, median, sum, std, min, and max. Return a new (Geo)DataFrame with a single row containing all aggregated values. Specify whether the geometries of the GeoDataFrame are to be aggregated. All geometries are merged union-like.

The return data type will always be the same as the input data type.

Parameters

- **df** – The (Geo)DataFrame to be analysed
- **var_names** – Variables to be aggregated ('None' uses all aggregatable columns)
- **aggregate_geometry** – Aggregate (union like) the geometry and add it to the resulting GeoDataFrame
- **monitor** – Monitor for progress bar

Returns

returns either DataFrame or GeoDataFrame. Keeps input data type

esa_climate_toolbox.ops.**data_frame_max**(**args*, *monitor:* Monitor = *Monitor.NONE*, ***kwargs*)

Select the first record of a data frame for which the given variable value is maximal.

Parameters

- **df** – The data frame or dataset.
- **var** – The variable.

Returns

A new, one-record data frame.

esa_climate_toolbox.ops.**data_frame_min**(**args*, *monitor:* Monitor = *Monitor.NONE*, ***kwargs*)

Select the first record of a data frame for which the given variable value is minimal.

Parameters

- **df** – The data frame or dataset.
- **var** – The variable.

Returns

A new, one-record data frame.

esa_climate_toolbox.ops.**data_frame_subset**(**args*, *monitor:* Monitor = *Monitor.NONE*, ***kwargs*)

Create a GeoDataFrame subset from given variables (data frame columns) and/or region.

Parameters

- **gdf** – A GeoDataFrame.
- **region_op** – The geometric operation to be performed if *region* is given.
- **region** – A region polygon used to filter rows.
- **var_names** – The variables (columns) to select.

Returns

A GeoDataFrame subset.

esa_climate_toolbox.ops.**find_closest**(**args*, *monitor:* Monitor = *Monitor.NONE*, ***kwargs*)

Find the *max_results* records closest to given *location* in the given GeoDataFrame *gdf*. Return a new GeoDataFrame containing the closest records.

If *dist_col_name* is given, store the actual distances in this column.

Distances are great-circle distances measured in degrees from a representative center of the given *location* geometry to the representative centres of each geometry in the *gdf*.

Parameters

- **gdf** – The GeoDataFrame.
- **location** – A location given as arbitrary geometry.
- **max_results** – Maximum number of results.
- **max_dist** – Ignore records whose distance is greater than this value in degrees.
- **dist_col_name** – Optional name of a new column that will store the actual distances.
- **monitor** – A progress monitor.

Returns

A new GeoDataFrame containing the closest records.

esa_climate_toolbox.ops.**query**(**args*, *monitor:* Monitor = *Monitor.NONE*, ***kwargs*)

Select records from the given data frame where the given conditional query expression evaluates to "True".

If the data frame *df* contains a geometry column (a `GeoDataFrame` object), then the query expression *query_expr* can also contain geometric relationship tests, for example the expression `"population > 100000 and @within('-10, 34, 20, 60')"` could be used on a data frame with the *population* and a *geometry* column to query for larger cities in West-Europe.

The geometric relationship tests are * `@almost_equals(geom)` - does a feature's geometry almost equal the given `geom`; * `@contains(geom)` - does a feature's geometry contain the given `geom`; * `@crosses(geom)` - does a feature's geometry cross the given `geom`; * `@disjoint(geom)` - does a feature's geometry not at all intersect the given `geom`; * `@intersects(geom)` - does a feature's geometry intersect with given `geom`; * `@touches(geom)` - does a feature's geometry have a point in common with given `geom` but does not intersect it; * `@within(geom)` - is a feature's geometry contained within given `geom`.

The *geom* argument may be a point `"<lon>, <lat>"` text string, a bounding box `"<lon1>, <lat1>, <lon2>, <lat2>"` text, or any valid geometry WKT.

Parameters

- **df** – The data frame or dataset.
- **query_expr** – The conditional query expression.

Returns

A new data frame.

Resampling

esa_climate_toolbox.ops.**resample**(**args*, *monitor:* Monitor = *Monitor.NONE*, ***kwargs*)

Resample a dataset to the provided x- and y-resolution. The resolution must be given in the units of the CRS. It can be set which method to use to upsample integer or float variables (in case the new resolution is finer than the old one) or to downsample them (in case the new resolution is coarser).

Parameters

- **ds** – The input dataset.
- **x_res** – The resolution in x-direction.
- **y_res** – The resolution in y-direction.
- **upsampling_float** – The upsampling method to be used for float values. This value is only used when the new resolution is finer than the previous one. Allowed values are 'nearest_neighbor', 'bilinear', '2nd-order spline', 'cubic', '4th-order spline', and '5th-order spline'. The default is 'bilinear'.
- **upsampling_int** – The upsampling method to be used for integer and boolean values. This value is only used when the new resolution is finer than the previous one. Allowed values are 'nearest_neighbor', 'bilinear', '2nd-order spline', 'cubic', '4th-order spline', and '5th-order spline'. The default is 'nearest_neighbor'.
- **downsampling_float** – The downsampling method to be used for float values. This value is only used when the new resolution is coarser than the previous one. Allowed values are 'nearest_neighbor', 'mean', 'min', and 'max'. The default is 'mean'.
- **downsampling_int** – The downsampling method to be used for integer and boolean values. This value is only used when the new resolution is coarser than the previous one. Allowed values are 'nearest_neighbor', 'mean', 'min', and 'max'. The default is 'nearest_neighbor'.

Returns

A new dataset resampled to the new resolutions.

esa_climate_toolbox.ops.**resample_2d**(*src*, *w*, *h*, *ds_method=54*, *us_method=11*, *fill_value=None*, *mode_rank=1*, *out=None*)

Resample a 2-D grid to a new resolution.

Parameters

- **src** – 2-D *ndarray*
- **w** – *int* New grid width
- **h** – *int* New grid height
- **ds_method** – one of the *DS_* constants, optional Grid cell aggregation method for a possible downsampling
- **us_method** – one of the *US_* constants, optional Grid cell interpolation method for a possible upsampling
- **fill_value** – *scalar*, optional If `None`, it is taken from **src** if it is a masked array, otherwise from *out* if it is a masked array, otherwise numpy's default value is used.
- **mode_rank** – *scalar*, optional The rank of the frequency determined by the *ds_method* `DS_MODE`. One (the default) means most frequent value, two means second most frequent value, and so forth.
- **out** – 2-D *ndarray*, optional Alternate output array in which to place the result. The default is *None*; if provided, it must have the same shape as the expected output.

Returns
An resampled version of the *src* array.

esa_climate_toolbox.ops.**downsample_2d**(*src*, *w*, *h*, *method=54*, *fill_value=None*, *mode_rank=1*, *out=None*)

Downsample a 2-D grid to a lower resolution by aggregating original grid cells.

Parameters

- **src** – 2-D *ndarray*
- **w** – *int* Grid width, which must be less than or equal to *src.shape[-1]*
- **h** – *int* Grid height, which must be less than or equal to *src.shape[-2]*
- **method** – one of the *DS_* constants, optional Grid cell aggregation method
- **fill_value** – *scalar*, optional If `None`, it is taken from **src** if it is a masked array, otherwise from *out* if it is a masked array, otherwise numpy's default value is used.
- **mode_rank** – *scalar*, optional The rank of the frequency determined by the *method* `DS_MODE`. One (the default) means most frequent value, two means second most frequent value, and so forth.
- **out** – 2-D *ndarray*, optional Alternate output array in which to place the result. The default is *None*; if provided, it must have the same shape as the expected output.

Returns
A downsampled version of the *src* array.

esa_climate_toolbox.ops.**upsample_2d**(*src*, *w*, *h*, *method=11*, *fill_value=None*, *out=None*)

Upsample a 2-D grid to a higher resolution by interpolating original grid cells.

Parameters

- **src** – 2-D *ndarray*
- **w** – *int* Grid width, which must be greater than or equal to *src.shape[-1]*
- **h** – *int* Grid height, which must be greater than or equal to *src.shape[-2]*
- **method** – one of the *US_* constants, optional Grid cell interpolation method
- **fill_value** – *scalar*, optional If `None`, it is taken from **src** if it is a masked array, otherwise from *out* if it is a masked array, otherwise numpy's default value is used.
- **out** – 2-D *ndarray*, optional Alternate output array in which to place the result. The default is *None*; if provided, it must have the same shape as the expected output.

Returns
An upsampled version of the *src* array.

Subsetting

esa_climate_toolbox.ops.**subset_spatial**(**args*, *monitor:* Monitor = *Monitor.NONE*, ***kwargs*)

Do a spatial subset of the dataset

Parameters

- **ds** – Dataset to subset
- **region** – Spatial region to subset
- **mask** – Should values falling in the bounding box of the polygon but not the polygon itself be masked with NaN.

- **monitor** – A monitor to report the progress of the process

Returns
Subset dataset

esa_climate_toolbox.ops.**subset_temporal**(**args*, *monitor:* Monitor = *Monitor.NONE*, ***kwargs*)

Do a temporal subset of the dataset.

Parameters

- **ds** – Dataset or dataframe to subset
- **time_range** – Time range to select

Returns
Subset dataset

esa_climate_toolbox.ops.**subset_temporal_index**(**args*, *monitor:* Monitor = *Monitor.NONE*, ***kwargs*)

Do a temporal indices based subset

Parameters

- **ds** – Dataset or dataframe to subset
- **time_ind_min** – Minimum time index to select
- **time_ind_max** – Maximum time index to select

Returns
Subset dataset

Timeseries

esa_climate_toolbox.ops.**tseries_point**(**args*, *monitor:* Monitor = *Monitor.NONE*, ***kwargs*)

Extract time-series from *ds* at given *lon*, *lat* position using interpolation *method* for each *var* given in a comma separated list of variables.

The operation returns a new timeseries dataset, that contains the point timeseries for all required variables with original variable meta-information preserved.

If a variable has more than three dimensions, the resulting timeseries variable will preserve all other dimensions except for lon/lat.

Parameters

- **ds** – The dataset from which to perform timeseries extraction.
- **point** – Point to extract, e.g. (lon,lat)
- **var** – Variable(s) for which to perform the timeseries selection if none is given, all variables in the dataset will be used.
- **method** – Interpolation method to use.

Returns
A timeseries dataset

esa_climate_toolbox.ops.**tseries_mean**(**args*, *monitor:* Monitor = *Monitor.NONE*, ***kwargs*)

Extract spatial mean timeseries of the provided variables, return the dataset that in addition to all the information in the given dataset contains also timeseries data for the provided variables, following naming convention 'var_name1_ts_mean'. In addition, the standard deviation is computed.

If a data variable with more dimensions than time/lat/lon is provided, the data will be reduced by taking the mean of all data values at a single time position resulting in one dimensional timeseries data variable.

Parameters

- **ds** – The dataset from which to perform timeseries extraction.
- **var** – Variables for which to perform timeseries extraction
- **mean_suffix** – Mean suffix to use for resulting datasets
- **std_suffix** – Std suffix to use for resulting datasets
- **monitor** – a progress monitor.

Returns

Dataset with timeseries variables

Misc

esa_climate_toolbox.ops.**detect_outliers**(**args*, *monitor:* Monitor = *Monitor.NONE*, ***kwargs*)

Detect outliers in the given Dataset.

When mask=True the input dataset should not contain nan values, otherwise all existing nan values will be marked as 'outliers' in the mask data array added to the output dataset.

Parameters

- **ds** – The dataset or dataframe for which to do outlier detection
- **var** – Variable or variables in the dataset to which to do outlier detection. Note that when multiple variables are selected, absolute threshold values might not make much sense. Wild cards can be used to select multiple variables matching a pattern.
- **threshold_low** – Values less or equal to this will be removed/masked
- **threshold_high** – Values greater or equal to this will be removed/masked
- **quantiles** – If True, threshold values are treated as quantiles, otherwise as absolute values.
- **mask** – If True, an ancillary variable containing flag values for outliers will be added to the dataset. Otherwise, outliers will be replaced with nan directly in the data variables.
- **monitor** – A progress monitor.

Returns

The dataset with outliers masked or replaced with nan

esa_climate_toolbox.ops.**merge**(**args*, *monitor:* Monitor = *Monitor.NONE*, ***kwargs*)

Merge up to four datasets to produce a new dataset with combined variables from each input dataset.

This is a wrapper for the `xarray.merge()` function.

For documentation refer to xarray documentation at http://xarray.pydata.org/en/stable/generated/xarray.Dataset.merge.html#xarray.Dataset.merge

The *compat* argument indicates how to compare variables of the same name for potential conflicts:

- "broadcast_equals": all values must be equal when variables are broadcast against each other to ensure common dimensions.
- "equals": all values and dimensions must be the same.
- "identical": all values, dimensions and attributes must be the same.
- **"no_conflicts": only values which are not null in both datasets must be** equal. The returned dataset then contains the combination of all non-null values.

Parameters

- **ds_1** – The first input dataset.
- **ds_2** – The second input dataset.
- **ds_3** – An optional 3rd input dataset.
- **ds_4** – An optional 4th input dataset.
- **join** – How to combine objects with different indexes.
- **compat** – How to compare variables of the same name for potential conflicts.

Returns
A new dataset with combined variables from each input dataset.

esa_climate_toolbox.ops.**normalize**(**args*, *monitor:* Monitor = *Monitor.NONE*, ***kwargs*)

Normalize the geo- and time-coding upon opening the given dataset w.r.t. to a common (CF-compatible) convention used within the ESA Climate Toolbox. This will maximize the compatibility of a dataset for usage with operations.

That is, * variables named "latitude" will be renamed to "lat"; * variables named "longitude" or "long" will be renamed to "lon";

Then, for equi-rectangular grids, * Remove 2D "lat" and "lon" variables; * Two new 1D coordinate variables "lat" and "lon" will be generated from original 2D forms.

Finally, it will be ensured that a "time" coordinate variable will be of type *datetime*.

Parameters
ds – The dataset to normalize.

Returns
The normalized dataset, or the original dataset, if it is already "normal".

esa_climate_toolbox.ops.**adjust_spatial_attrs**(**args*, *monitor:* Monitor = *Monitor.NONE*, ***kwargs*)

Adjust the global spatial attributes of the dataset by doing some introspection of the dataset and adjusting the appropriate attributes accordingly.

In case the determined attributes do not exist in the dataset, these will be added.

For more information on suggested global attributes see Attribute Convention for Data Discovery

Parameters

- **ds** – Dataset to adjust
- **allow_point** – Whether a dataset containing a single point is allowed

Returns
Adjusted dataset

esa_climate_toolbox.ops.**adjust_temporal_attrs**(**args*, *monitor:* Monitor = *Monitor.NONE*, ***kwargs*)

Adjust the global temporal attributes of the dataset by doing some introspection of the dataset and adjusting the appropriate attributes accordingly.

In case the determined attributes do not exist in the dataset, these will be added.

If the temporal attributes exist, but the dataset lacks a variable 'time', a new dimension 'time' of size one will be added and related coordinate variables 'time' and 'time_bnds' are added to the dataset. The dimension of all non-coordinate variables will be expanded by the new time dimension.

For more information on suggested global attributes see Attribute Convention for Data Discovery

Parameters
ds – Dataset to adjust

Returns
Adjusted dataset

6.5.3 Operation Registration API

class esa_climate_toolbox.core.**Operation**(*wrapped_op: Callable*, *op_meta_info=None*)

An Operation comprises a wrapped callable (e.g. function, constructor, lambda form) and additional meta-information about the wrapped operation itself and its inputs and outputs.

Parameters

- **wrapped_op** – some callable object that will be wrapped.
- **op_meta_info** – operation meta information.

property op_meta_info: *OpMetaInfo*

Returns
Meta-information about the operation, see `esa_climate_toolbox.core.op.OpMetaInfo`.

property wrapped_op: Callable

Returns
The actual operation object which may be any callable.

class esa_climate_toolbox.core.**OpMetaInfo**(*qualified_name: str*, *has_monitor: bool = False*, *header: dict = None*, *input_names: List[str] = None*, *inputs: Dict[str, Dict[str, Any]] = None*, *outputs: Dict[str, Dict[str, Any]] = None*)

Represents meta-information about an operation:

- *qualified_name*: a an ideally unique, qualified operation name
- *header*: dictionary of arbitrary operation attributes
- `input`: ordered dictionary of named inputs, each mapping to a dictionary of arbitrary input attributes
- `output`: ordered dictionary of named outputs, each mapping to a dictionary of arbitrary output attributes

Warning: *OpMetaInfo* ` objects should be considered immutable. However, the dictionaries mentioned above are returned "as-is", mostly for performance reasons. Changing entries in these dictionaries directly may cause unwanted side-effects.

Parameters

- **qualified_name** – The operation's qualified name.
- **has_monitor** – Whether the operation supports a *Monitor* keyword argument named `monitor`.
- **header** – Header information dictionary.
- **input_names** – Input information dictionary.
- **inputs** – Input information dictionary.
- **outputs** – Output information dictionary.

MONITOR_INPUT_NAME = 'monitor'

The constant `'monitor'`, which is the name of an operation input that will receive a *Monitor* object as value.

RETURN_OUTPUT_NAME = 'return'

The constant `'return'`, which is the name of a single, unnamed operation output.

property has_monitor: bool

Returns
`True` if the operation supports a *Monitor* value as additional keyword argument named `monitor`.

property has_named_outputs: bool

Returns
`True` if the output value of the operation is expected be a dictionary-like mapping of output names to output values.

property header: Dict[str, Any]

Returns
Operation header attributes.

property input_names: List[str]

The input names in the order they have been declared.

Returns
List of input names.

property inputs: Dict[str, Dict[str, Any]]

Mapping from an input name to a dictionary of properties describing the input.

Returns
Named inputs.

property outputs: Dict[str, Dict[str, Any]]

Mapping from an output name to a dictionary of properties describing the output.

Returns
Named outputs.

property qualified_name: str

Returns
Fully qualified name of the actual operation.

set_default_input_values(*input_values: Dict*)

If any missing input value in *input_values*, set value of "default_value" property, if it exists.

Parameters
input_values – The dictionary of input values that will be modified.

to_json_dict(*data_type_to_json=None*) → Dict[str, Any]

Return a JSON-serializable dictionary representation of this object. E.g. values of the *data_type* ` property are converted from Python types to their string representation.

Returns
A JSON-serializable dictionary

validate_input_values(*input_values: ~typing.Dict*, *except_types=None*, *validation_exception_class=<class 'ValueError'>*)

Validate given *input_values* against the operation's input properties.

Parameters

- **input_values** – The dictionary of input values.
- **except_types** – A set of types or `None`. If an input value's type is in this set, it will not be validated against the various input properties, such as `data_type`, `nullable`, `value_set`, `value_range`.
- **validation_exception_class** – The exception class to be used to raise exceptions if validation fails. Must derive from `BaseException`. Defaults to `ValueError`.

Raises

validation_error_class – If *input_values* are invalid w.r.t. to the operation's input properties.

validate_output_values(*output_values: ~typing.Dict*, *validation_exception_class: type = <class 'ValueError'>*)

Validate given *output_values* against the operation's output properties.

Parameters

- **output_values** – The dictionary of output values.
- **validation_exception_class** – The exception class to be used to raise exceptions if validation fails. Must derive from `BaseException`. Defaults to `ValueError`.

Raises

validation_error_class – If *output_values* are invalid w.r.t. to the operation's output properties.

esa_climate_toolbox.core.**op**(*tags=UNDEFINED*, *version=UNDEFINED*, *res_pattern=UNDEFINED*, *deprecated=UNDEFINED*, *registry=OP_REGISTRY*, ***properties*)

`op` is a decorator function that registers a Python function or class in the default operation registry or the one given by *registry*, if any. Any other keywords arguments in *header* are added to the operation's meta-information header. Classes annotated by this decorator must have callable instances.

When a function is registered, an introspection is performed. During this process, initial operation the meta-information header property *description* is derived from the function's docstring.

If any output of this operation will have its history information automatically updated, there should be version information found in the operation header. Thus it's always a good idea to add it to all operations:

```
@op(version='X.x')
```

Parameters

- **tags** – An optional list of string tags.
- **version** – An optional version string.
- **res_pattern** – An optional pattern that will be used to generate the names for data resources that are used to hold a reference to the objects returned by the operation. Currently, the only pattern variable that is supported and that must be present is `{index}` which will be replaced by an integer number that is guaranteed to produce a unique resource name.

- **deprecated** – An optional boolean or a string. If a string is used, it should explain why the operation has been deprecated and which new operation to use instead. If set to `True`, the operation's doc-string should explain the deprecation.
- **registry** – The operation registry.
- **properties** – Other properties (keyword arguments) that will be added to the meta-information of operation.

esa_climate_toolbox.core.**op_input**(*input_name: str*, *default_value=UNDEFINED*, *units=UNDEFINED*, *data_type=UNDEFINED*, *nullable=UNDEFINED*, *value_set_source=UNDEFINED*, *value_set=UNDEFINED*, *value_range=UNDEFINED*, *script_lang=UNDEFINED*, *deprecated=UNDEFINED*, *position=UNDEFINED*, *context=UNDEFINED*, *registry=OP_REGISTRY*, ***properties*)

`op_input` is a decorator function that provides meta-information for an operation input identified by *input_name*. If the decorated function or class is not registered as an operation yet, it is added to the default operation registry or the one given by *registry*, if any.

When a function is registered, an introspection is performed. During this process, initial operation meta-information input properties are derived for each positional and keyword argument named *input_name*:

Derived property	Source
position	The position of a positional argument, e.g. `2` for input `z` in `def f(x, y, z, c=2)`.
default_value	The value of a keyword argument, e.g. `52.3` for input `latitude` from argument definition `latitude:float=52.3`
data_type	The type annotation type, e.g. `float` for input `latitude` from argument definition `latitude:float`

The derived properties listed above plus any of *value_set*, *value_range*, and any key-value pairs in *properties* are added to the input's meta-information. A key-value pair in *properties* will always overwrite the derived properties listed above.

Parameters

- **input_name** – The name of an input.
- **default_value** – A default value.
- **units** – The geo-physical units of the input value.
- **data_type** – The data type of the input values. If not given, the type of any given, non-None *default_value* is used.
- **nullable** – If `True`, the value of the input may be `None`. If not given, it will be set to `True` if the *default_value* is `None`.
- **value_set_source** – The name of an input, which can be used to generate a dynamic value set.
- **value_set** – A sequence of the valid values. Note that all values in this sequence must be compatible with *data_type*.
- **value_range** – A sequence specifying the possible range of valid values.
- **script_lang** – The programming language for a parameter of data_type "str" that provides source code of a script, e.g. "python".

- **deprecated** – An optional boolean or a string. If a string is used, it should explain why the input has been deprecated and which new input to use instead. If set to `True`, the input's doc-string should explain the deprecation.
- **position** – The zero-based position of an input.
- **context** – If `True`, the value of the operation input will be a dictionary representing the current execution context. If *context* is a string, the value of the operation input will be the result of evaluating the string as Python expression with the current execution context as local environment. This means, *context* may be an expression such as 'value_cache', 'workspace.base_dir', 'step', 'step.id'.
- **properties** – Other properties (keyword arguments) that will be added to the meta-information of the named output.
- **registry** – Optional operation registry.

esa_climate_toolbox.core.**op_output**(*output_name: str*, *data_type=UNDEFINED*, *deprecated=UNDEFINED*, *registry=OP_REGISTRY*, ***properties*)

`op_output` is a decorator function that provides meta-information for an operation output identified by *output_name*. If the decorated function or class is not registered as an operation yet, it is added to the default operation registry or the one given by *registry*, if any.

If your function does not return multiple named outputs, use the *op_return()* decorator function. Note that:

```
@op_return(...)
def my_func(...):
    ...
```

if equivalent to:

```
@op_output('return', ...)
def my_func(...):
    ...
```

To automatically add information about the ESA Climate Toolbox, its version, this operation and its inputs, to this output, set 'add_history' to True:

```
@op_output('name', add_history=True)
```

Note that the operation should have version information added to it when add_history is True:

```
@op(version='X.x')
```

Parameters

- **output_name** – The name of the output.
- **data_type** – The data type of the output value.
- **deprecated** – An optional boolean or a string. If a string is used, it should explain why the output has been deprecated and which new output to use instead. If set to `True`, the output's doc-string should explain the deprecation.
- **properties** – Other properties (keyword arguments) that will be added to the meta-information of the named output.
- **registry** – Optional operation registry.

esa_climate_toolbox.core.**op_return**(*data_type=UNDEFINED*, *registry=OP_REGISTRY*, ***properties*)

op_return is a decorator function that provides meta-information for a single, anonymous operation return value (whose output name is "return"). If the decorated function or class is not registered as an operation yet, it is added to the default operation registry or the one given by *registry*, if any. Any other keywords arguments in *properties* are added to the output's meta-information.

When a function is registered, an introspection is performed. During this process, initial operation meta-information output properties are derived from the function's return type annotation, that is *data_type* will be, e.g., `float` if a function is annotated as `def f(x, y) -> float: ...`.

The derived *data_type* property and any key-value pairs in *properties* are added to the output's meta-information. A key-value pair in *properties* will always overwrite a derived *data_type*.

If your function returns multiple named outputs, use the *op_output()* decorator function. Note that:

```
@op_return(...)
def my_func(...):
    ...
```

if equivalent to:

```
@op_output('return', ...)
def my_func(...):
    ...
```

To automatically add information about the ESA Climate Toolbox, its version, this operation and its inputs, to this output, set 'add_history' to True:

```
@op_return(add_history=True)
```

Note that the operation should have version information added to it when add_history is True:

```
@op(version='X.x')
```

Parameters

- **data_type** – The data type of the return value.
- **properties** – Other properties (keyword arguments) that will be added to the meta-information of the return value.
- **registry** – The operation registry.

esa_climate_toolbox.core.**new_expression_op**(*op_meta_info:* OpMetaInfo, *expression: str*) → *Operation*

Create an operation that wraps a Python expression.

Parameters

- **op_meta_info** – Meta-information about the resulting operation and the operation's inputs and outputs.
- **expression** – The Python expression. May refer to any name given in *op_meta_info.input*.

Returns

The Python expression wrapped into an operation.

esa_climate_toolbox.core.**new_subprocess_op**(*op_meta_info: OpMetaInfo*, *command_pattern: str*, *run_python: bool = False*, *cwd: str | None = None*, *env: Dict[str, str] = None*, *shell: bool = False*, *started: str | Callable = None*, *progress: str | Callable = None*, *done: str | Callable = None*) → *Operation*

Create an operation for a child program run in a new process.

Parameters

- **op_meta_info** – Meta-information about the resulting operation and the operation's inputs and outputs.
- **command_pattern** – A pattern that will be interpolated to obtain the actual command to be executed. May contain "{input_name}" fields which will be replaced by the actual input value converted to text. *input_name* must refer to a valid operation input name in *op_meta_info.input* or it must be the value of either the "write_to" or "read_from" property of another input's property map.
- **run_python** – If True, *command_pattern* refers to a Python script which will be executed with the Python interpreter that the Climate Toolbox uses.
- **cwd** – Current working directory to run the command line in.
- **env** – Environment variables passed to the shell that executes the command line.
- **shell** – Whether to use the shell as the program to execute.
- **started** – Either a callable that receives a text line from the executable's stdout and returns a tuple (label, total_work) or a regex that must match in order to signal the start of progress monitoring. The regex must provide the group names "label" or "total_work" or both, e.g. "(?P<label>w+)" or "(?P<total_work>d+)"
- **progress** – Either a callable that receives a text line from the executable's stdout and returns a tuple (work, msg) or a regex that must match in order to signal process. The regex must provide group names "work" or "msg" or both, e.g., "(?P<msg>w+)" or "(?P<work>d+)"
- **done** – Either a callable that receives a text line a text line from the executable's stdout and returns True or False or a regex that must match in order to signal the end of progress monitoring.

Returns

The executable wrapped into an operation.

Managing Operations

esa_climate_toolbox.core.**get_op**(*op_name: str*, *op_registry: OpRegistry = OP_REGISTRY*) → *Operation*

Returns an operation.

Parameters

- **op_name** – The name of the operation.
- **op_registry** – An optional OpRegistry, in case the default one should not be used.

Returns

An operation which may directly be called

esa_climate_toolbox.core.**get_op_meta_info**(*op_name: str*, *op_registry: OpRegistry = OP_REGISTRY*) → Dict

Returns meta information about an operation.

Parameters

- **op_name** – The name of the operation for which meta information shall be provided.
- **op_registry** – An optional OpRegistry, in case the default one should not be used.

Returns

A dictionary representation of an operator's meta info, providing information about input parameters and the expected output.

esa_climate_toolbox.core.**list_operations**(*op_registry: OpRegistry = OP_REGISTRY*, *include_qualified_name: bool = False*)

Lists the operations that are provided by the ESA Climate Toolbox.

Parameters

- **op_registry** – An optional OpRegistry, in case the default one should not be used.
- **include_qualified_name** – If true, a more expressive qualified name will be returned along with the method name. Default is false.

Returns

Either a list of the names of operations, or a list of tuples, each consisting of the operation name and a qualified name

6.5.4 Task Monitoring API

class esa_climate_toolbox.core.**Monitor**

A monitor is used to both observe and control a running task.

The `Monitor` class is an abstract base class for concrete monitors. Derived classes must implement the following three abstract methods: *`start()`*, *`progress()`*, and *`done()`*. Derived classes must implement also the following two abstract methods, if they want cancellation support: *`cancel()`* and *`is_cancelled()`*.

Pass `Monitor.NONE` to functions that expect a monitor instead of passing `None`.

Given here is an example of how progress monitors should be used by functions::

```
def long_running_task(a, b, c, monitor):
    with monitor.starting('doing a long running task', total_work=100)
        # do 30% of the work here
        monitor.progress(work=30)
        # do 70% of the work here
        monitor.progress(work=70)
```

If a function makes calls to other functions that also support a monitor, a *child-monitor* is used::

```
def long_running_task(a, b, c, monitor):
    with monitor.starting('doing a long running task', total_work=100)
        # let other_task do 30% of the work
        other_task(a, b, c, monitor=monitor.child(work=30))
        # let other_task do 70% of the work
        other_task(a, b, c, monitor=monitor.child(work=70))
```

cancel()

Request the task to be cancelled. This method will be usually called from the code that created the monitor, not by users of the monitor. For example, a GUI could create the monitor due to an invocation of a

long-running task, and then the user wishes to cancel that task. The default implementation does nothing. Override to implement something useful.

check_for_cancellation()

Checks if the monitor has been cancelled and raises a `Cancellation` in that case.

child(*work: float = 1*) → *Monitor*

Return a child monitor for the given partial amount of *work*.

Parameters
work – The partial amount of work.

Returns
a sub-monitor

abstract done()

Call to signal that a task has been done.

is_cancelled() → bool

Check if there is an external request to cancel the current task observed by this monitor.

Users of a monitor shall frequently call this method and check its return value. If cancellation is requested, they should politely exit the current processing in a proper way, e.g., by cleaning up allocated resources. The default implementation returns `False`. Subclasses shall override this method to return `True` if a task cancellation request was detected.

Returns
`True` if task cancellation was requested externally. The default implementation returns `False`.

observing(*label: str*)

A context manager for easier use of progress monitors. Observes a `dask` task and reports back to the monitor.

Parameters
label – Passed to the monitor's `start` method

Returns

abstract progress(*work: float = None*, *msg: str = None*)

Call to signal that a task has mode some progress.

Parameters

- **work** – The incremental amount of work.
- **msg** – A detail message.

abstract start(*label: str*, *total_work: float = None*)

Call to signal that a task has started.

Note that *label* and *total_work* are not passed to `__init__`, because they are usually not known at constructions time. It is the responsibility of the task to derive the appropriate values for these.

Parameters

- **label** – A task label
- **total_work** – The total amount of work

starting(*label: str*, *total_work: float = None*)

A context manager for easier use of progress monitors. Calls the monitor's `start` method with *label* and *total_work*. Will then take care of calling `Monitor.done()`.

Parameters

- **label** – Passed to the monitor's `start` method
- **total_work** – Passed to the monitor's `start` method

Returns

class esa_climate_toolbox.core.**ChildMonitor**(*parent_monitor:* Monitor, *partial_work: float*)

A child monitor is responsible for a partial amount of work of a *parent_monitor*.

Parameters

- **parent_monitor** – the parent monitor
- **partial_work** – the partial amount of work of *parent_monitor*.

cancel()

Request the task to be cancelled. This method will be usually called from the code that created the monitor, not by users of the monitor. For example, a GUI could create the monitor due to an invocation of a long-running task, and then the user wishes to cancel that task. The default implementation does nothing. Override to implement something useful.

done()

Call to signal that a task has been done.

is_cancelled() → bool

Check if there is an external request to cancel the current task observed by this monitor.

Users of a monitor shall frequently call this method and check its return value. If cancellation is requested, they should politely exit the current processing in a proper way, e.g., by cleaning up allocated resources. The default implementation returns `False`. Subclasses shall override this method to return `True` if a task cancellation request was detected.

Returns

`True` if task cancellation was requested externally. The default implementation returns `False`.

progress(*work: float = None*, *msg: str = None*)

Call to signal that a task has mode some progress.

Parameters

- **work** – The incremental amount of work.
- **msg** – A detail message.

start(*label: str*, *total_work: float = None*)

Call to signal that a task has started.

Note that *label* and *total_work* are not passed to `__init__`, because they are usually not known at constructions time. It is the responsibility of the task to derive the appropriate values for these.

Parameters

- **label** – A task label
- **total_work** – The total amount of work

INDEX