xcube

Release 1.3.0

Brockmann Consult GmbH

Oct 25, 2023

GETTING STARTED

1	Overview	3
2	Examples	7
3	Installation	15
4	CLI	19
5	Python API	51
6	Web API and Server	101
7	Viewer App	103
8	Data Access	113
9	The xcube generator	123
10	xcube Dataset Convention	129
11	Common data store conventions	135
12	xcube Developer Guide	141
13	Plugins	151
14	Indices and tables	155
Inc	lex	157

xcube has been developed to generate, manipulate, analyse, and publish data cubes from EO data.

CHAPTER

OVERVIEW

xcube is an open-source Python package and toolkit that has been developed to provide Earth observation (EO) data in an analysis-ready form to users. xcube achieves this by carefully converting EO data sources into self-contained *data cubes* that can be published in the cloud. The Python package is expanded and maintained by [Brockmann Consult GmbH](https://www.brockmann-consult.de)

1.1 Data Cube

The interpretation of the term *data cube* in the EO domain usually depends on the current context. It may refer to a data service such as Sentinel Hub, to some abstract API, or to a concrete set of spatial images that form a time-series.

This section briefly explains the specific concept of a data cube used in the xcube project - the xcube dataset.

1.2 xcube Dataset

1.2.1 Data Model

An xcube dataset contains one or more (geo-physical) data variables whose values are stored in cells of a common multidimensional, spatio-temporal grid. The dimensions are usually time, latitude, and longitude, however other dimensions may be present.

All xcube datasets are structured in the same way following a common data model. They are also self-describing by providing metadata for the cube and all cube's variables following the CF conventions. For details regarding the common data model, please refer to the *xcube Dataset Convention*.

A xcube dataset's in-memory representation in Python programs is an xarray.Dataset instance. Each dataset variable is represented by multi-dimensional xarray.DataArray that is arranged in non-overlapping, contiguous sub-regions called *data chunks*.

1.2.2 Data Chunks

Chunked variables allow for out-of-core computations of xcube dataset that don't fit in a single computer's RAM as data chunks can be processed independently from each other.

The way how dataset variables are sub-divided into smaller chunks - their *chunking* - has a substantial impact on processing performance and there is no single ideal chunking for all use cases. For time series analyses it is preferable to have chunks with a smaller spatial dimensions and larger time dimension, for spatial analyses and visualisation on using a map, the opposite is the case.

xcube provide tools for re-chunking of xcube datasets (*xcube chunk*, *xcube level*) and the xcube server (*xcube serve*) allows serving the same data cubes using different chunkings. For further reading have a look into the Chunking and Performance section of the xarray documentation.

1.2.3 Processing Model

When xcube datasets are opened, only the cube's structure and its metadata are loaded into memory. The actual data arrays of variables are loaded on-demand only, and only for chunks intersecting the desired sub-region.

Operations that generate new data variables from existing ones will be chunked in the same way. Therefore, such operation chains generate a processing graph providing a deferred, concurrent execution model.

1.2.4 Data Format

For the external, physical representation of xcube datasets we usually use the Zarr format. Zarr takes full advantage of data chunks and supports parallel processing of chunks that may originate from the local file system or from remote cloud storage such as S3 and GCS.

1.2.5 Python Packages

The xcube package builds heavily on Python's big data ecosystem for handling huge N-dimensional data arrays and exploiting cloud-based storage and processing resources. In particular, xcube's in-memory data model is provided by xarray, the memory management and processing model is provided through dask, and the external format is provided by zarr. xarray, dask, and zarr have increased their popularity for big data solutions over the last couple of years, for creating scalable and efficient EO data solutions.

1.3 Toolkit

On top of xarray, dask, zarr, and other popular Python data science packages, xcube provides various higher-level tools to generate, manipulate, and publish xcube datasets:

- CLI access, generate, modify, and analyse xcube datasets using the xcube tool;
- Python API access, generate, modify, and analyse xcube datasets via Python programs and notebooks;
- Web API and Server access, analyse, visualize xcube datasets via an xcube server;
- Viewer App publish and visualise xcube datasets using maps and time-series charts.

1.4 Workflows

The basic use case is to generate an xcube dataset and deploy it so that your users can access it:

- 1. generate an xcube dataset from some EO data sources using the xcube gen tool with a specific input processor.
- 2. optimize the generated xcube dataset with respect to specific use cases using the xcube chunk tool.
- 3. optimize the generated xcube dataset by consolidating metadata and elimination of empty chunks using *xcube optimize* and *xcube prune* tools.
- 4. deploy the optimized xcube dataset(s) to some location (e.g. on AWS S3) where users can access them.

Then you can:

- 5. access, analyse, modify, transform, visualise the data using the *Python API* and xarray API through Python programs or JupyterLab, or
- 6. extract data points by coordinates from a cube using the *xcube extract* tool, or
- 7. resample the cube in time to generate temporal aggregations using the *xcube resample* tool.

Another way to provide the data to users is via the *xcube server*, that provides a RESTful API and a WMTS. The latter is used to visualise spatial subsets of xcube datasets efficiently at any zoom level. To provide optimal visualisation and data extraction performance through the xcube server, xcube datasets may be prepared beforehand. Steps 8 to 10 are optional.

- 8. verify a dataset to be published conforms with the *xcube Dataset Convention* using the *xcube verify* tool.
- 9. adjust your dataset chunking to be optimal for generating spatial image tiles and generate a multi-resolution image pyramid using the *xcube chunk* and *xcube level* tools.
- 10. create a dataset variant optimal for time series-extraction again using the *xcube chunk* tool.
- 11. configure xcube datasets and publish them through the xcube server using the *xcube serve* tool.

You may then use a WMTS-compatible client to visualise the datasets or develop your own xcube server client that will make use of the xcube's REST API.

The easiest way to visualise your data is using the xcube *Viewer App*, a single-page web application that can be configured to work with xcube server URLs.

CHAPTER

TWO

EXAMPLES

When you follow the examples section you can build your first tiny xcube dataset and view it in the xcube-viewer by using the xcube server. The examples section is still growing and improving :)

Have fun exploring xcube!

Warning: This chapter is a work in progress and currently less than a draft.

2.1 Generating an xcube dataset

In the following example a tiny demo xcube dataset is generated.

2.1.1 Analysed Sea Surface Temperature over the Global Ocean

Input data for this example is located in the xcube repository. The input files contain analysed sea surface temperature and sea surface temperature anomaly over the global ocean and are provided by Copernicus Marine Environment Monitoring Service. The data is described in a dedicated Product User Manual.

Before starting the example, you need to activate the xcube environment:

```
$ conda activate xcube
```

If you want to take a look at the input data you can use cli/xcube dump to print out the metadata of a selected input file:

```
$ xcube dump examples/gen/data/20170605120000-UKMO-L4_GHRSST-SSTfnd-OSTIAanom-GLOB-v02.0-

$ fv02.0.nc
```

```
<xarray.Dataset>
                  (lat: 720, lon: 1440, time: 1)
Dimensions:
Coordinates:
                  (lat) float32 -89.875 -89.625 -89.375 ... 89.375 89.625 89.875
  * lat
  * lon
                  (lon) float32 0.125 0.375 0.625 ... 359.375 359.625 359.875
  * time
                  (time) object 2017-06-05 12:00:00
Data variables:
    sst_anomaly
                  (time, lat, lon) float32 ...
   analysed_sst (time, lat, lon) float32 ...
Attributes:
    Conventions:
                                CF-1.4
```

(continued from previous page)

	(continued from previous pr
title:	Global SST & Sea Ice Anomaly, L4 OSTIA, 0.25
summary:	A merged, multi-sensor L4 Foundation SST anom
references:	Donlon, C.J., Martin, M., Stark, J.D., Robert
institution:	UKMO
history:	Created from sst :temperature regridded with a
comment:	WARNING Some applications are unable to prope
license:	These data are available free of charge under
id:	UKMO-L4LRfnd_GLOB-OSTIAanom
naming_authority:	org.ghrsst
<pre>product_version:</pre>	2.0
uuid:	5c1665b7-06e8-499d-a281-857dcbfd07e2
gds_version_id:	2.0
<pre>netcdf_version_id:</pre>	3.6
date_created:	20170606T061737Z
<pre>start_time:</pre>	20170605 T000000Z
<pre>time_coverage_start:</pre>	20170605 T000000Z
stop_time:	20170606 T000000Z
<pre>time_coverage_end:</pre>	20170606T000000Z
file_quality_level:	3
source:	UKMO-L4HRfnd-GLOB-OSTIA
platform:	Aqua, Envisat, NOAA-18, NOAA-19, MetOpA, MSG1
sensor:	AATSR, AMSR, AVHRR, AVHRR_GAC, SEVIRI, TMI
metadata_conventions:	Unidata Observation Dataset v1.0
metadata_link:	http://data.nodc.noaa.gov/NESDIS_DataCenters/
keywords:	Oceans > Ocean Temperature > Sea Surface Temp
keywords_vocabulary:	NASA Global Change Master Directory (GCMD) Sc
<pre>standard_name_vocabulary:</pre>	NetCDF Climate and Forecast (CF) Metadata Con
westernmost_longitude:	0.0
easternmost_longitude:	360.0
southernmost_latitude:	-90.0
northernmost_latitude:	90.0 9.35 dogmoo
<pre>spatial_resolution:</pre>	<pre>0.25 degree degrees_north</pre>
<pre>geospatial_lat_units: geospatial_lat_resolution:</pre>	0.25 degree
<pre>geospatial_lat_resolution. geospatial_lon_units:</pre>	degrees_east
<pre>geospatial_lon_resolution:</pre>	0.25 degree
acknowledgment:	Please acknowledge the use of these data with
creator_name:	Met Office as part of CMEMS
creator_email:	servicedesk.cmems@mercator-ocean.eu
creator_url:	http://marine.copernicus.eu/
project:	Group for High Resolution Sea Surface Tempera
publisher_name:	GHRSST Project Office
publisher_url:	http://www.ghrsst.org
publisher_email:	ghrsst-po@nceo.ac.uk
processing_level:	L4
cdm_data_type:	grid
· _ / ·	

Below an example xcube dataset will be created, which will contain the variable analysed_sst. The metadata for a specific variable can be viewed by:

\$ xcube dump examples/gen/data/20170605120000-UKMO-L4_GHRSST-SSTfnd-OSTIAanom-GLOB-v02.0-\$ fv02.0.nc --var analysed_sst

```
<xarray.DataArray 'analysed_sst' (time: 1, lat: 720, lon: 1440)>
[1036800 values with dtype=float32]
Coordinates:
  * lat
             (lat) float32 -89.875 -89.625 -89.375 ... 89.375 89.625 89.875
  * lon
             (lon) float32 0.125 0.375 0.625 0.875 ... 359.375 359.625 359.875
  * time
             (time) object 2017-06-05 12:00:00
Attributes:
                    analysed sea surface temperature
    long_name:
    standard_name:
                    sea_surface_foundation_temperature
    type:
                    foundation
   units:
                    kelvin
   valid_min:
                    -300
   valid_max:
                    4500
    source:
                    UKMO-L4HRfnd-GLOB-OSTIA
    comment:
```

For creating a toy xcube dataset you can execute the command-line below. Please adjust the paths to your needs:

The configuration file specifies the input processor, which in this case is the default one. The output size is 10240, 5632. The bounding box of the data cube is given by output_region in the configuration file. The output format (output_writer_name) is defined as well. The chunking of the dimensions can be set by the chunksizes attribute of the output_writer_params parameter, and in the example configuration file the chunking is set for latitude and longitude. If the chunking is not set, a automatic chunking is applied. The spatial resampling method (output_resampling) is set to 'nearest' and the configuration file contains only one variable which will be included into the xcube dataset - 'analysed-sst'.

The Analysed Sea Surface Temperature data set contains the variable already as needed. This means no pixel masking needs to be applied. However, this might differ depending on the input data. You can take a look at a configuration file which takes Sentinel-3 Ocean and Land Colour Instrument (OLCI) as input files, which is a bit more complex. The advantage of using pixel expressions is, that the generated cube contains only valid pixels and the user of the data cube does not have to worry about something like land-masking or invalid values. Furthermore, the generated data cube is spatially regular. This means the data are aligned on a common spatial grid and cover the same region. The time stamps are kept from the input data set.

Caution: If you have input data that has file names not only varying with the time stamp but with e.g. A and B as well, you need to pass the input files in the desired order via a text file. Each line of the text file should contain the path to one input file. If you pass the input files in a desired order, then do not use the parameter **--sort** within the commandline interface.

2.1.2 Optimizing and pruning a xcube dataset

If you want to optimize your generated xcube dataset e.g. for publishing it in a xcube viewer via xcube serve you can use cli/xcube optimize:

\$ xcube optimize demo_SST_xcube.zarr -C

By executing the command above, an optimized xcube dataset called demo_SST_xcube-optimized.zarr will be created. You can take a look into the directory of the original xcube dataset and the optimized one, and you will notice that a file called .zmetadata. .zmetadata contains the information stored in .zattrs and .zarray of each variable of the xcube dataset and makes requests of metadata faster. The option -C optimizes coordinate variables by converting any chunked arrays into single, non-chunked, contiguous arrays.

For deleting empty chunks cli/xcube prune can be used. It deletes all data files associated with empty (NaN-only) chunks of an xcube dataset, and is restricted to the ZARR format.

\$ xcube prune demo_SST_xcube-optimized.zarr

The pruned xcube dataset is saved in place and does not need an output path. The size of the xcube dataset was 6,8 MB before pruning it and 6,5 MB afterwards. According to the output printed to the terminal, 30 block files were deleted.

The metadata of the xcube dataset can be viewed with cli/xcube dump as well:

\$ xcube dump demo_SST_xcube-optimized.zarr

```
<xarray.Dataset>
Dimensions:
                  (bnds: 2, lat: 5632, lon: 10240, time: 3)
Coordinates:
  * lat
                  (lat) float64 62.67 62.66 62.66 62.66 ... 48.01 48.0 48.0
                  (lat, bnds) float64 dask.array<shape=(5632, 2), chunksize=(5632, 2)>
    lat bnds
  * lon
                  (lon) float64 -16.0 -16.0 -15.99 -15.99 ... 10.66 10.66 10.67
    lon_bnds
                  (lon, bnds) float64 dask.array<shape=(10240, 2), chunksize=(10240, 2)>
  * time
                  (time) datetime64[ns] 2017-06-05T12:00:00 ... 2017-06-07T12:00:00
    time_bnds
                  (time, bnds) datetime64[ns] dask.array<shape=(3, 2), chunksize=(3, 2)>
Dimensions without coordinates: bnds
Data variables:
    analysed_sst (time, lat, lon) float64 dask.array<shape=(3, 5632, 10240),
\rightarrow chunksize=(1, 704, 640)>
Attributes:
    acknowledgment:
                                Data Cube produced based on data provided by ...
    comment:
   contributor name:
    contributor_role:
                                info@brockmann-consult.de
   creator email:
                                Brockmann Consult GmbH
   creator_name:
   creator_url:
                                https://www.brockmann-consult.de
   date_modified:
                                2019-09-25T08:50:32.169031
                                62.666666666666666
   geospatial_lat_max:
   geospatial_lat_min:
                                48.0
                                0.002604166666666666
    geospatial_lat_resolution:
    geospatial_lat_units:
                                degrees_north
    geospatial_lon_max:
                                10.66666666666666
                                -16.0
   geospatial_lon_min:
                                0.002604166666666665
   geospatial_lon_resolution:
   geospatial_lon_units:
                                degrees_east
   history:
                                xcube/reproj-snap-nc
                                demo-bc-sst-sns-l2c-v1
   id:
    institution:
                                Brockmann Consult GmbH
   kevwords:
                                terms and conditions of the DCS4COP data dist...
   license:
   naming_authority:
                                bc
                                L2C
   processing_level:
   project:
                                xcube
                                info@brockmann-consult.de
   publisher_email:
                                Brockmann Consult GmbH
   publisher_name:
   publisher_url:
                                https://www.brockmann-consult.de
   references:
                                https://dcs4cop.eu/
```

(continued from previous page)

```
source:CMEMS Global SST & Sea Ice Anomaly Data Cubestandard_name_vocabulary:summary:2017-06-08T00:00:00.000000000time_coverage_end:2017-06-05T00:00:00.000000000time_coverage_start:2017-06-05T00:00:00.000000000title:CMEMS Global SST Anomaly Data Cube
```

The metadata for the variable analysed_sst can be viewed:

\$ xcube dump demo_SST_xcube-optimized.zarr --var analysed_sst

```
<xarray.DataArray 'analysed_sst' (time: 3, lat: 5632, lon: 10240)>
dask.array<shape=(3, 5632, 10240), dtype=float64, chunksize=(1, 704, 640)>
Coordinates:
  * lat
             (lat) float64 62.67 62.66 62.66 62.66 ... 48.01 48.01 48.0 48.0
  * lon
             (lon) float64 -16.0 -16.0 -15.99 -15.99 ... 10.66 10.66 10.66 10.67
             (time) datetime64[ns] 2017-06-05T12:00:00 ... 2017-06-07T12:00:00
  * time
Attributes:
   comment:
    long_name:
                         analysed sea surface temperature
   source:
                         UKMO-L4HRfnd-GLOB-OSTIA
   spatial_resampling: Nearest
   standard_name:
                         sea_surface_foundation_temperature
   type:
                         foundation
   units:
                         kelvin
   valid_max:
                         4500
   valid_min:
                         -300
```

Warning: This chapter is a work in progress and currently less than a draft.

2.2 Publishing xcube datasets

This example demonstrates how to run an xcube server to publish existing xcube datasets.

2.2.1 Running the server

To run the server on default port 8080 using the demo configuration:

\$ xcube serve --verbose -c examples/serve/demo/config.yml

To run the server using a particular xcube dataset path and styling information for a variable:

```
$ xcube serve --styles conc_chl=(0,20,"viridis") examples/serve/demo/cube-1-250-250.zarr
```

2.2.2 Test it

After starting the server, you can check the various functions provided by xcube Web API. To explore the functions, open

base-url>/openapi.html.

2.2.3 xcube Viewer

xcube datasets published through xcube serve can be visualised using the xcube-viewer web application. To do so, run xcube serve with the --open-viewer flag.

In order make this option usable, xcube-viewer must be installed and build:

- 1. Download and install yarn.
- 2. Download and build xcube-viewer:

```
$ git clone https://github.com/dcs4cop/xcube-viewer.git
$ cd xcube-viewer
$ yarn install
$ yarn build
```

3. Configure xcube serve so it finds the xcube-viewer On Linux (please adjust path):

```
$ export XCUBE_VIEWER_PATH=/abs/path/to/xcube-viewer/build
```

```
On Windows (please adjust path):
```

> SET XCUBE_VIEWER_PATH=/abs/path/to/xcube-viewer/build

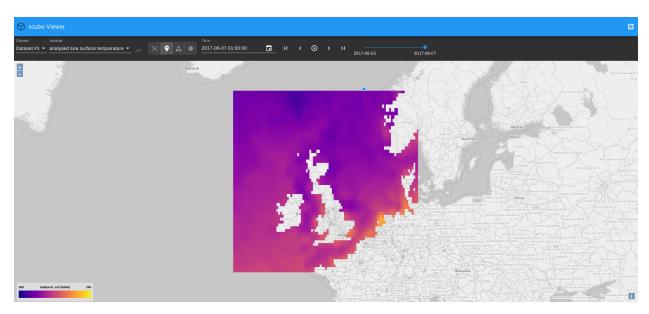
4. Then run xcube serve --open-viewer:

```
$ xcube serve --open-viewer --styles conc_chl=(0,20,"viridis") examples/serve/demo/cube-
→1-250-250.zarr
```

Viewing the generated xcube dataset described in the example Generating an xcube dataset:

```
$ xcube serve --open-viewer --styles "analysed_sst=(280,290,'plasma')" demo_SST_xcube-

optimized.zarr
```



In case you get an error message "cannot reach server" on the very bottom of the web app's main window, refresh the page.

You can play around with the value range displayed in the viewer, here it is set to min=280K and max=290K. The colormap used for mapping can be modified as well and the colormaps provided by matplotlib can be used.

2.2.4 Other clients

There are example HTML pages for some tile server clients. They need to be run in a web server. If you don't have one, you can use Node's httpserver:

```
$ npm install -g httpserver
```

After starting both the xcube server and web server, e.g. on port 9090:

\$ httpserver -d -p 9090

you can run the client demos by following their links given below.

OpenLayers

- OpenLayers 4 Demo
- OpenLayers 4 Demo with WMTS

Cesium

To run the Cesium Demo first download Cesium and unpack the zip into the xcube serve source directory so that there exists an ./Cesium-x.y.z sub-directory. You may have to adapt the Cesium version number in the demo's HTML file.

CHAPTER

THREE

INSTALLATION

3.1 Prerequisites

xcube releases are packaged as conda packages in the conda-forge channel. It is recommended to install xcube into a conda environment using the mamba package manager, which will also automatically install and manage xcube's dependencies. You can find installation instructions for mamba itself here, if you don't already have it installed.

In addition to mamba, there are alternative package managers available for conda environments:

- 1. The original conda tool. When considering this tool, please note that package installation and management with conda may be significantly slower than with mamba.
- 2. The micromamba tool, a minimalistic, self-contained version of mamba.

3.2 Overview of installation methods

There are two main ways to install the xcube package:

- 1. **Install an official release** from a conda-forge package, using the mamba package manager. This method is recommended for most users.
- 2. Use mamba to install only xcube's dependencies, but not xcube itself. Then **clone the xcube git repository** and install directly from your local repository. You should use this method if you intend to participate in the development of xcube, or if you need to use features that are so new that they are not yet available in an officially release conda-forge package.

These methods are described in more detail in the following sections.

3.3 Installation from the conda-forge package

To install the latest release of xcube into a new conda environment called xcube, run the following command.

mamba create --name xcube --channel conda-forge xcube

You can give the environment a different name by providing a different argument to the --name option.

To install xcube into an existing, currently activated conda environment, use the following command.

mamba install --channel conda-forge xcube

3.4 Installation from the source code repository

First, clone the repository and create a conda environment from it:

```
git clone https://github.com/dcs4cop/xcube.git
cd xcube
mamba env create
```

From this point on, all instructions assume that your current directory is the root of the xcube repository.

The mamba env create command above creates an environment according to the specifications in the environment. yml file in the repository, which by default takes the name xcube. Then, to activate the environment and install xcube from the repository:

```
mamba activate xcube
pip install --no-deps --editable .
```

The second command installs xcube in 'editable mode', meaning that it will be run directly from the repository, and changes to the code in the repository will take immediate effect without reinstallation. (As an alternative to pip, the command python setup.py develop can be used, but this is no longer recommended. Among other things, pip has the advantage of allowing easy deinstallation of installed packages.)

To update the install to the latest repository version and update the environment to reflect to any changes in environment.yml:

```
mamba activate xcube
git pull --force
mamba env update -n xcube --file environment.yml --prune
```

To install pytest and run the unit test suite:

```
mamba install pytest pytest
```

To analyse test coverage (after installing pytest as above):

pytest --cov=xcube

To produce an HTML coverage report:

pytest --cov-report html --cov=xcube

3.5 Docker

To start a demo using docker use the following commands

```
docker build -t [your name] .
docker run [your name]
docker run -d -p [host port]:8080 [your name]
```

Example 1:

```
docker build -t xcube:0.10.0 .
docker run xcube:0.10.0
```

This will create the docker container and list the functionality of the xcube cli.

Example 2:

```
docker build -t xcube:0.10.0 .
docker run -d -p 8001:8080 xcube:0.10.0 "xcube serve -v --address 0.0.0.0 --port 8080 -c_ 
→/home/xcube/examples/serve/demo/config.yml"
docker ps
```

This will start a service in the background which can be accessed through port 8001, as the startup of a service is configured as default behaviour.

CLI

The xcube command-line interface (CLI) is a single executable cli/xcube with several sub-commands comprising functions ranging from xcube dataset generation, over analysis and manipulation, to dataset publication.

4.1 Common Arguments and Options

Most of the commands operate on inputs that are xcube datasets. Such inputs are consistently named CUBE and provided as one or more command arguments. CUBE inputs may be a path into the local file system or a path into some object storage bucket, e.g. in AWS S3. Command inputs of other types are consistently called INPUT.

Many commands also output something, i.e. are writing files. The paths or names of such outputs are consistently provided by the -o OUTPUT or --output OUTPUT option. As the output is an option, there is usually a default value for it. If multiply file formats are supported, commands usually provide a -f FORMAT or --format FORMAT option. If omitted, the format may be guessed from the output's name.

4.2 Cube generation

4.2.1 xcube gen

Synopsis

Generate xcube dataset.

```
$ xcube gen --help
```

```
Usage: xcube gen [OPTIONS] [INPUT]...

Generate xcube dataset. Data cubes may be created in one go or successively

for all given inputs. Each input is expected to provide a single time slice

which may be appended, inserted or which may replace an existing time slice

in the output dataset. The input paths may be one or more input files or a

pattern that may contain wildcards '?', '*', and '**'. The input paths can

also be passed as lines of a text file. To do so, provide exactly one input

file with ".txt" extension which contains the actual input paths to be used.

Options:

-P, --proc INPUT-PROCESSOR Input processor name. The available input

processor names and additional information
```

(continued from previous page)

	about input processors can be accessed by
	calling xcube geninfo . Defaults to
	"default", an input processor that can deal
	with simple datasets whose variables have
	dimensions ("lat", "lon") and conform with
	the CF conventions.
<pre>-c,config CONFIG</pre>	xcube dataset configuration file in YAML
	format. More than one config input file is
	allowed When passing several config files,
	they are merged considering the order passed
	via command line.
-o,output OUTPUT	Output path. Defaults to 'out.zarr'
-f,format FORMAT	Output format. Information about output
,	formats can be accessed by calling xcube gen
	info. If omitted, the format will be
	guessed from the given output path.
-S,size SIZE	Output size in pixels using format
5, 5120 5121	" <width>,<height>".</height></width>
-R,region REGION	Output region using format " <lon-min>,<lat-< td=""></lat-<></lon-min>
R, Tegron Aldron	min>, <lon-max>,<lat-max>"</lat-max></lon-max>
variables,vars VARIABLES	Variables to be included in output. Comma-
Valiables, Vals VARIABLES	separated list of names which may contain
	wildcard characters "*" and "?".
nocompling	
resampling	line lieuene e Merel Mediere MirelMedel Merere et 1011027
\rightarrow [Average Bilinear Cubic CubicSp	line Lanczos Max Median Min Mode Nearest Q1 Q3]
	Fallback spatial resampling algorithm to be
	used for all variables. Defaults to
	'Nearest'. The choices for the resampling
	algorithm are: ['Average', 'Bilinear',
	'Cubic', 'CubicSpline', 'Lanczos', 'Max',
	'Median', 'Min', 'Mode', 'Nearest', 'Q1',
	'Q3']
-a,append	Deprecated. The command will now always
	create, insert, replace, or append input
	slices.
prof	Collect profiling information and dump
	results after processing.
no_sort	The input file list will not be sorted
	before creating the xcube dataset. If
	no_sort parameter is passed, the order of
	the input list will be kept. This parameter
	should be used for better performance,
	provided that the input file list is in
	correct order (continuous time).
-I,info	Displays additional information about format
	options or about input processors.
dry_run	Just read and process inputs, but don't
	produce any outputs.
help	Show this message and exit.

Below is the ouput of a xcube gen --info call showing five input processors installed via plugins.

\$	xcube	gen	info
----	-------	-----	------

· · · · · · · · · · · · · · · · · · ·					
input processors to be used with optionproc:					
default	Single-scene NetCDF/CF inputs in xcube standard				
⇔format					
rbins-seviri-highroc-scene-12	RBINS SEVIRI HIGHROC single-scene Level-2 NetCDF				
→inputs					
rbins-seviri-highroc-daily-12	RBINS SEVIRI HIGHROC daily Level-2 NetCDF inputs				
<pre>snap-olci-highroc-l2</pre>	SNAP Sentinel-3 OLCI HIGHROC Level-2 NetCDF inputs				
<pre>snap-olci-cyanoalert-l2</pre>	SNAP Sentinel-3 OLCI CyanoAlert Level-2 NetCDF inputs				
vito-s2plus-12	VITO Sentinel-2 Plus Level 2 NetCDF inputs				
For more input processors use existing "xcube-gen" plugins from the github. →organisation DCS4COP or write own plugin.					
Output formats to be used with optionformat:					
zarr (*.zarr)	Zarr file format (http://zarr.readthedocs.io)				
netcdf4 (*.nc)	NetCDF-4 file format				
CSV (*.CSV)	CSV file format				
mem (*.mem)	In-memory dataset I/O				

Configuration File

Configuration files passed to xcube gen via the -c, --config option use YAML format. Multiple configuration files may be given. In this case all configurations are merged into a single one. Parameter values will be overwritten by subsequent configurations if they are scalars. If they are objects / mappings, their values will be deeply merged.

The following parameters can be used in the configuration files:

input_processor

[str] The name of an *input processor*. See -P, --proc option above.

Default

The default value is 'default', xcube's default input processor. It can ingest and process inputs that

- use an EPSG: 4326 (or compatible) grid;
- have 1-D lon and lat coordinate variables using WGS84 coordinates and decimal degrees;
- have a decodable 1-D time coordinate or define the one of the following global attribute pairs time_coverage_start and time_coverage_end, time_start and time_end or time_stop;
- provide data variables with the dimensions time, lat, lon, in this order.
- conform to the **`CF Conventions`_**.

output_size

[[int, int]] The spatial dimension sizes of the output dataset given as number of grid cells in longitude and latitude direction (width and height).

output_region

[[float, float, float, float]] The spatial extent of output datasets given as a bounding box [lat-min, lat-min, lon-max, lat-max] using decimal degrees.

output_variables

[[variable-definitions]] The definition of variables that will be included in the output dataset. Each variable definition may be just a name or a mapping from a name to variable attributes. If it is just a name it must be the name of an existing variable either in the INPUT or in processed_variables. If the variable definition is a mapping, some of the attributes affect the way how variables are processed. All but the name attributes become variable metadata in the output.

name

[str] The new name of the variable in the output.

valid_pixel_expression

[str] An expression used to mask this variable, see *Expressions*. The expression identifies all valid pixels in each INPUT.

resampling

[str] The resampling method used. See --resampling option above.

Default

By default, all variables in INPUT will occur in output.

processed_variables

[[variable-definitions]] The definition of variables that will be produced or processed after reading each IN-PUT. The main purpose is to generate intermediate variables that can be referred to in the expression in other variable definitions in processed_variables and valid_pixel_expression in variable definitions in output_variables. The following attributes are recognised:

expression

[str] An expression used to produce this variable, see *Expressions*.

output_writer_name

[str] The name of a supported output format. May be one of 'zarr', 'netcdf4', 'mem'.

Default

'zarr

output_writer_params

[str] A mapping that defines parameters that are passed to output writer denoted by output_writer_name. Through the output_writer_params a packing of the variables may be defined. If not specified the default does not apply any packing which results in:

```
_FillValue: nan
dtype: dtype('float32')
```

and for coordinate variables

dtype: dtype('int64')

The user may specify a different packing variables, which might be useful for reducing the storage size of the datacubes. Currently it is only implemented for zarr format. This may be done by passing the parameters for packing as the following:

```
output_writer_params:
```

```
packing:
  analysed_sst:
    scale_factor: 0.07324442274239326
    add_offset: -300.0
    dtype: 'uint16'
    _FillValue: 0.65535
```

Furthermore the compressor may be defined as well by, if not specified the default compressor (cname='lz4', clevel=5, shuffle=SHUFFLE, blocksize=0) is used.

```
output_writer_params:
    compressor:
        cname: 'zstd'
        clevel: 1
        shuffle: 2
```

output_metadata

[[*attribute-definitions*]] General metadata that will be present in the output dataset as global attributes. You can put any common CF attributes here.

Any attributes that are mappings will be "flattened" by concatenating the attribute names using the underscrore character. For example,:

```
publisher:
  name: "Brockmann Consult GmbH"
  url: "https://www.brockmann-consult.de"
```

will create the two entries:

```
publisher_name: "Brockmann Consult GmbH"
publisher_url: "https://www.brockmann-consult.de"
```

Expressions

Expressions are plain text values of the expression and valid_pixel_expression attributes of the variable definitions in the processed_variables and output_variables parameters. The expression syntax is that of standard Python. xcube gen uses expressions to produce new variables listed in processed_variables and to mask variables by the valid_pixel_expression.

An expression may refer any variables in the INPUT datasets and any variables defined by the processed_variables parameter. Expressions may make use of most of the standard Python operators and may apply all numpy ufuncs to referred variables. Also most of the xarray.DataArray API may be used on variables within an expression.

In order to utilise flagged variables, the syntax variable_name.flag_name can be used in expressions. According to the CF Conventions, flagged variables are variables whose metadata include the attributes flag_meanings and flag_values and/or flag_masks. The flag_meanings attribute enumerates the allowed values for flag_name. The flag attributes must be present in the variables of each INPUT.

Example

An example that uses a configuration file only:

\$ xcube gen --config ./config.yml /data/eo-data/SST/2018/**/*.nc

An example that uses the default input processor and passes all other configuration via command-line options:

Some input processors have been developed for specific EO data sources used within the DCS4COP project. They may serve as examples how to develop input processor plug-ins:

- xcube-gen-rbins
- xcube-gen-bc
- xcube-gen-vito

Python API

The related Python API function is xcube.core.gen.gen.gen_cube().

4.2.2 xcube grid

Attention: This tool will likely change in the near future.

Synopsis

Find spatial xcube dataset resolutions and adjust bounding boxes.

```
$ xcube grid --help
```

```
Usage: xcube grid [OPTIONS] COMMAND [ARGS]...
  Find spatial xcube dataset resolutions and adjust bounding boxes.
  We find suitable resolutions with respect to a possibly regional fixed
  Earth grid and adjust regional spatial bounding boxes to that grid. We
  also try to select the resolutions such that they are taken from a certain
  level of a multi-resolution pyramid whose level resolutions increase by a
  factor of two.
  The graticule at a given resolution level L within the grid is given by
      RES(L) = COVERAGE * HEIGHT(L)
      \text{HEIGHT}(L) = \text{HEIGHT}_0 * 2 \wedge L
      LON(L, I) = LON_MIN + I * HEIGHT_0 * RES(L)
      LAT(L, J) = LAT_MIN + J * HEIGHT_0 * RES(L)
  With
      RES:
                Grid resolution in degrees.
      HEIGHT:
                Number of vertical grid cells for given level
      HEIGHT_0: Number of vertical grid cells at lowest resolution level.
  Let WIDTH and HEIGHT be the number of horizontal and vertical grid cells
  of a global grid at a certain LEVEL with WIDTH * RES = 360 and HEIGHT *
  RES = 180, then we also force HEIGHT = TILE * 2 ^ LEVEL.
Options:
  --help Show this message and exit.
Commands:
```

(continued from previous page)

```
aboxAdjust a bounding box to a fixed Earth grid.levelsList levels for a resolution or a tile size.resList resolutions close to a target resolution.
```

Example: Find suitable target resolution for a \sim 300m (Sentinel 3 OLCI FR resolution) fixed Earth grid within a deviation of 5%.

```
$ xcube grid res 300m -D 5%
```

TILE	LEVEL	HEIGHT	INV_RES	RES (deg)	RES (m)	, DELTA_1	RES (%)
540	7	69120	384	0.0026041666666	666665	289.9	-3.4
4140	4	66240	368	0.0027173913043	47826	302.5	0.8
8100	3	64800	360	0.0027777777777	77778	309.2	3.1

289.9m is close enough and provides 7 resolution levels, which is good. Its inverse resolution is 384, which is the fixed Earth grid identifier.

We want to see if the resolution pyramid also supports a resolution close to 10m (Sentinel 2 MSI resolution).

\$ xcube grid levels 384 -m 6

LEVEL	HEIGHT	INV_RES	RES (deg)	RES	(m)	
0	540	3	0.33333333333333333	3333		37106.5
1	1080	6	0.166666666666	66666		18553.2
2	2160	12	0.083333333333333	33333		9276.6
11	1105920	6144	0.000162760416	566666	566	18.1
12	2211840	12288	8.138020833333	333e-0	95	9.1
13	4423680	24576	4.069010416666	5664e-	-05	4.5

This indicates we have a resolution of 9.1m at level 12.

Lets assume we have xcube dataset region with longitude from 0 to 5 degrees and latitudes from 50 to 52.5 degrees. What is the adjusted bounding box on a fixed Earth grid with the inverse resolution 384?

\$ xcube grid abox 0,50,5,52.5 384

```
Orig. box coord. = 0.0, 50.0, 5.0, 52.5
Adj. box coord. = 0.0, 49.21875, 5.625, 53.4375
                 = POLYGON ((0.0 50.0, 5.0 50.0, 5.0 52.5, 0.0 52.5, 0.0 50.0))
Orig. box WKT
Adj. box WKT
                 = POLYGON ((0.0 49.21875, 5.625 49.21875, 5.625 53.4375, 0.0 53.4375, 0.
→0 49.21875))
Grid size = 2160 x 1620 cells
with
  TILE
           = 540
  LEVEL
           = 7
  INV_RES = 384
  RES (deg) = 0.0026041666666666665
  RES (m)
           = 289.89450727414993
```

Note, to check bounding box WKTs, you can use the handy Wicket tool.

4.3 Cube computation

4.3.1 xcube compute

Synopsis

Compute a cube variable from other cube variables using a user-provided Python function.

```
$ xcube compute --help
```

```
Usage: xcube compute [OPTIONS] SCRIPT [CUBE]...
  Compute a cube from one or more other cubes.
  The command computes a cube variable from other cube variables in CUBEs
  using a user-provided Python function in SCRIPT.
  The SCRIPT must define a function named "compute":
      def compute(*input_vars: numpy.ndarray,
                  input_params: Mapping[str, Any] = None,
                  dim_coords: Mapping[str, np.ndarray] = None,
                  dim_ranges: Mapping[str, Tuple[int, int]] = None) \
                  -> numpy.ndarray:
          # Compute new numpy array from inputs
          # output_array = ...
          return output_array
  where input_vars are numpy arrays (chunks) in the order given by VARIABLES
  or given by the variable names returned by an optional "initialize" function
  that my be defined in SCRIPT too, see below. input_params is a mapping of
  parameter names to values according to PARAMS or the ones returned by the
  aforesaid "initialize" function. dim_coords is a mapping from dimension name
  to coordinate labels for the current chunk to be computed. dim_ranges is a
  mapping from dimension name to index ranges into coordinate arrays of the
  cube.
  The SCRIPT may define a function named "initialize":
      def initialize(input_cubes: Sequence[xr.Dataset],
                     input_var_names: Sequence[str],
                     input_params: Mapping[str, Any]) \
                     -> Tuple[Sequence[str], Mapping[str, Any]]:
          # Compute new variable names and/or new parameters
          # new_input_var_names = ...
          # new_input_params = ...
          return new_input_var_names, new_input_params
  where input_cubes are the respective CUBEs, input_var_names the respective
  VARIABLES, and input_params are the respective PARAMS. The "initialize"
  function can be used to validate the data cubes, extract the desired
  variables in desired order and to provide some extra processing parameters
```

```
(continued from previous page)
```

```
passed to the "compute" function.
  Note that if no input variable names are specified, no variables are passed
  to the "compute" function.
  The SCRIPT may also define a function named "finalize":
      def finalize(output_cube: xr.Dataset,
                   input_params: Mapping[str, Any]) \
                   -> Optional[xr.Dataset]:
          # Optionally modify output_cube and return it or return None
          return output_cube
  If defined, the "finalize" function will be called before the command writes
  the new cube and then exists. The functions may perform a cleaning up or
  perform side effects such as write the cube to some sink. If the functions
  returns None, the CLI will *not* write any cube data.
Options:
  --variables, --vars VARIABLES Comma-separated list of variable names.
                                 Parameters passed as 'input_params' dict to
  -p, --params PARAMS
                                 compute() and init() functions in SCRIPT.
  -o, --output OUTPUT
                                 Output path. Defaults to 'out.zarr'
  -f, --format FORMAT
                                 Output format.
  -N, --name NAME
                                 Output variable's name.
  -D, --dtype DTYPE
                                 Output variable's data type.
                                 Show this message and exit.
  --help
```

Example

\$ xcube compute s3-olci-cube.zarr ./algoithms/s3-olci-ndvi.py

with ./algoithms/s3-olci-ndvi.py being:

TODO

Python API

The related Python API function is *xcube.core.compute.compute_cube()*.

4.4 Cube inspection

4.4.1 xcube dump

Synopsis

Dump contents of a dataset.

```
$ xcube dump --help
Usage: xcube dump [OPTIONS] INPUT
Dump contents of an input dataset.
Options:
   --variable, --var VARIABLE Name of a variable (multiple allowed).
   -E, --encoding Dump also variable encoding information.
   --help Show this message and exit.
```

Example

```
$ xcube dump xcube_cube.zarr
```

4.4.2 xcube verify

Synopsis

Perform cube verification.

```
$ xcube verify --help
```

```
Usage: xcube verify [OPTIONS] CUBE
Perform cube verification.
The tool verifies that CUBE
* defines the dimensions "time", "lat", "lon";
* has corresponding "time", "lat", "lon" coordinate variables and that they
are valid, e.g. 1-D, non-empty, using correct units;
* has valid bounds variables for "time", "lat", "lon" coordinate
variables, if any;
* has any data variables and that they are valid, e.g. min. 3-D, all have
same dimensions, have at least dimensions "time", "lat", "lon".
* spatial coordinates and their corresponding bounds (if exist) are equidistant
```

(continued from previous page)

```
and monotonically increasing or decreasing.
If INPUT is a valid xcube dataset, the tool returns exit code 0. Otherwise a
violation report is written to stdout and the tool returns exit code 3.
Options:
    --help Show this message and exit.
```

Python API

The related Python API functions are

- xcube.core.verify.verify_cube(), and
- xcube.core.verify.assert_cube().

4.5 Cube data extraction

4.5.1 xcube extract

Synopsis

Extract cube points.

```
$ xcube extract --help
```

```
Usage: xcube extract [OPTIONS] CUBE POINTS
  Extract cube points.
  Extracts data cells from CUBE at coordinates given in each POINTS record and
  writes the resulting values to given output path and format.
  POINTS must be a CSV file that provides at least the columns "lon", "lat",
  and "time". The "lon" and "lat" columns provide a point's location in
  decimal degrees. The "time" column provides a point's date or date-time. Its
  format should preferably be ISO, but other formats may work as well.
Options:
  -o, --output OUTPUT Output path. If omitted, output is written to stdout.
  -f, --format FORMAT Output format. Currently, only 'csv' is supported.
  -C, --coords
                       Include cube cell coordinates in output.
  -B, --bounds
                       Include cube cell coordinate boundaries (if any) in
                       output.
  -I, --indexes
                       Include cube cell indexes in output.
  -R, --refs
                       Include point values as reference in output.
  --help
                       Show this message and exit.
```

Example

\$ xcube extract xcube_cube.zarr -o point_data.csv -Cb --indexes --refs

Python API

Related Python API functions are

- xcube.core.extract.get_cube_values_for_points(),
- xcube.core.extract.get_cube_point_indexes(), and
- xcube.core.extract.get_cube_values_for_indexes().

4.6 Cube manipulation

4.6.1 xcube chunk

Synopsis

(Re-)chunk xcube dataset.

\$ xcube chunk --help

```
Usage: xcube chunk [OPTIONS] CUBE
  (Re-)chunk xcube dataset. Changes the external chunking of all variables of
  CUBE according to CHUNKS and writes the result to OUTPUT.
  Note: There is a possibly more efficient way to (re-)chunk datasets through
  the dedicated tool "rechunker", see https://rechunker.readthedocs.io.
Options:
  -o, --output OUTPUT Output path. Defaults to 'out.zarr'
  -f, --format FORMAT Format of the output. If not given, guessed from
                       OUTPUT.
  -p, --params PARAMS Parameters specific for the output format. Comma-
                       separated list of <key>=<value> pairs.
  -C, --chunks CHUNKS Chunk sizes for each dimension. Comma-separated list of
                       <dim>=<size> pairs, e.g. "time=1,lat=270,lon=270"
  -q, --quiet
                       Disable output of log messages to the console entirely.
                       Note, this will also suppress error and warning
                       messages.
  -v, --verbose
                       Enable output of log messages to the console. Has no
                       effect if --quiet/-q is used. May be given multiple
                       times to control the level of log messages, i.e., -v
                       refers to level INFO, -vv to DETAIL, -vvv to DEBUG,
                       -vvvv to TRACE. If omitted, the log level of the
                       console is WARNING.
  --help
                       Show this message and exit.
```

Example

Python API

The related Python API function is xcube.core.chunk.chunk_dataset().

4.6.2 xcube level

Synopsis

Generate multi-resolution levels.

```
$ xcube level --help
```

```
Usage: xcube level [OPTIONS] INPUT
  Generate multi-resolution levels.
  Transform the given dataset by INPUT into the levels of a multi-level
  pyramid with spatial resolution decreasing by a factor of two in both
  spatial dimensions and write the result to directory OUTPUT.
  INPUT may be an S3 object storage URL of the form "s3://<bucket>/<path>" or
  "https://<endpoint>".
Options:
                                  Output path. If omitted, "INPUT.levels" will
  -o, --output OUTPUT
                                  be used. You can also use S3 object storage
                                  URLs of the form "s3://<bucket>/<path>" or
                                  "https://<endpoint>"
                                  Link the INPUT instead of converting it to a
  -L, --link
                                  level zero dataset. Use with care, as the
                                  INPUT's internal spatial chunk sizes may be
                                  inappropriate for imaging purposes.
  -t, --tile-size TILE_SIZE
                                  Tile size, given as single integer number or
                                  as <tile-width>,<tile-height>. If omitted,
                                  the tile size will be derived from the
                                  INPUT's internal spatial chunk sizes. If the
                                  INPUT is not chunked, tile size will be 512.
  -n, --num-levels-max NUM_LEVELS_MAX
                                  Maximum number of levels to generate. If not
                                  given, the number of levels will be derived
                                  from spatial dimension and tile sizes.
  -A, --agg-methods AGG_METHODS
                                  Aggregation method(s) to be used for data
                                  variables. Either one of "first", "min",
                                  "max", "mean", "median", "auto" or list of
                                  assignments to individual variables using
                                  the notation
```

(continued from previous page)

	<pre>"<var1>=<method1>,<var2>=<method2>," Defaults to "first".</method2></var2></method1></var1></pre>
-r,replace	Whether to replace an existing dataset at
	OUTPUT.
-a,anon	For S3 inputs or outputs, whether the access
	is anonymous. By default, credentials are
	required.
-q,quiet	Disable output of log messages to the
	console entirely. Note, this will also
	suppress error and warning messages.
-v,verbose	Enable output of log messages to the
	console. Has no effect if quiet/-q is
	used. May be given multiple times to control
	the level of log messages, i.e., -v refers
	to level INFO, -vv to DETAIL, -vvv to DEBUG,
	-vvvv to TRACE. If omitted, the log level of
	the console is WARNING.
help	Show this message and exit.

Example

\$ xcube level --link -t 720 data/cubes/test-cube.zarr

Python API

The related Python API functions are

- xcube.core.level.compute_levels(),
- xcube.core.level.read_levels(), and
- xcube.core.level.write_levels().

4.6.3 xcube optimize

Synopsis

Optimize xcube dataset for faster access.

```
$ xcube optimize --help
```

```
Usage: xcube optimize [OPTIONS] CUBE

Optimize xcube dataset for faster access.

Reduces the number of metadata and coordinate data files in xcube dataset

given by CUBE. Consolidated cubes open much faster especially from remote

locations, e.g. in object storage, because obviously much less HTTP requests

are required to fetch initial cube meta information. That is, it merges all

metadata files into a single top-level JSON file ".zmetadata". Optionally,
```

(continued from previous page)

```
it removes any chunking of coordinate variables so they comprise a single
  binary data file instead of one file per data chunk. The primary usage of
  this command is to optimize data cubes for cloud object storage. The command
  currently works only for data cubes using ZARR format.
Options:
  -o, --output OUTPUT Output path. The placeholder "<built-in function
                       input>" will be replaced by the input's filename
                       without extension (such as ".zarr"). Defaults to
                       "{input}-optimized.zarr".
                       Optimize cube in place. Ignores output path.
  -I, --in-place
  -C, --coords
                       Also optimize coordinate variables by converting any
                       chunked arrays into single, non-chunked, contiguous
                       arrays.
  --help
                       Show this message and exit.
```

Examples

Write an cube with consolidated metadata to cube-optimized.zarr:

\$ xcube optimize ./cube.zarr

Write an optimized cube with consolidated metadata and consolidated coordinate variables to optimized/cube.zarr (directory optimized must exist):

\$ xcube optimize -C -o ./optimized/cube.zarr ./cube.zarr

Optimize a cube in-place with consolidated metadata and consolidated coordinate variables:

```
$ xcube optimize -IC ./cube.zarr
```

Python API

The related Python API function is xcube.core.optimize.optimize_dataset().

4.6.4 xcube patch

Synopsis

Patch and consolidate the metadata of a xcube dataset.

\$ xcube patch --help

```
Usage: xcube patch [OPTIONS] DATASET
Patch and consolidate the metadata of a dataset.
DATASET can be either a local filesystem path or a URL. It must point to
either a Zarr dataset (*.zarr) or a xcube multi-level dataset (*.levels).
```

(continues on next page)

(continued from previous page)

```
Additional storage options for a given protocol may be passed by the OPTIONS
  option.
  In METADATA, the special attribute value "__delete__" can be used to remove
  that attribute from dataset or array metadata.
Options:
  --metadata METADATA The metadata to be patched. Must be a JSON or YAML file
                       using Zarr consolidated metadata format.
                       Protocol-specific storage options (see fsspec). Must be
  --options OPTIONS
                       a JSON or YAML file.
  -q, --quiet
                       Disable output of log messages to the console entirely.
                       Note, this will also suppress error and warning
                       messages.
                       Enable output of log messages to the console. Has no
  -v, --verbose
                       effect if --quiet/-q is used. May be given multiple
                       times to control the level of log messages, i.e., -v
                       refers to level INFO, -vv to DETAIL, -vvv to DEBUG,
                       -vvvv to TRACE. If omitted, the log level of the
                       console is WARNING.
                       Do not change any data, just report what would have
  -d, --dry-run
                       been changed.
  --help
                       Show this message and exit.
```

Patch file example

Patch files use the Zarr Consolidated Metadata Format, v1. For example, the following patch file (YAML) will delete the global attribute TileSize and change the value of the attribute long_name of variable conc_chl:

```
zarr_consolidated_format: 1
metadata:
   .zattrs:
    TileSize: __delete__
   conc_chl/.zattrs:
    long_name: Chlorophyll concentration
```

Storage options file example

Here is a storage options file for the "s3" protocol that provides credentials for AWS S3 access:

```
key: AJDKJCLSKKA
secret: kjkl456lkj45632k45j631
```

Usage example

\$ xcube patch s3://my-cubes-bucket/test.zarr --metadata patch.yml -v

4.6.5 xcube prune

Delete empty chunks.

Attention: This tool will likely be integrated into xcube optimize in the near future.

\$ xcube prune --help

```
Usage: xcube prune [OPTIONS] DATASET
  Delete empty chunks. Deletes all data files associated with empty (NaN-only)
  chunks in given DATASET, which must have Zarr format.
Options:
  -q, --quiet
                 Disable output of log messages to the console entirely. Note,
                 this will also suppress error and warning messages.
  -v, --verbose Enable output of log messages to the console. Has no effect
                 if --quiet/-q is used. May be given multiple times to control
                 the level of log messages, i.e., -v refers to level INFO, -vv
                 to DETAIL, -vvv to DEBUG, -vvvv to TRACE. If omitted, the log
                 level of the console is WARNING.
                 Just read and process input, but don't produce any output.
  --dry-run
                 Show this message and exit.
  --help
```

A related Python API function is xcube.core.optimize.get_empty_dataset_chunks().

4.6.6 xcube resample

Synopsis

Resample data along the time dimension.

```
$ xcube resample --help
```

```
Usage: xcube resample [OPTIONS] CUBE

Resample data along the time dimension.

Options:

-c, --config CONFIG xcube dataset configuration file in YAML

format. More than one config input file is

allowed.When passing several config files,

they are merged considering the order passed

via command line.
```

(continues on next page)

(continued from previous pag	e)
------------------------------	----

-o,output OUTPUT	Output path. Defaults to 'out.zarr'.
<pre>-f,format [zarr netcdf4 mem]</pre>	Output format. If omitted, format will be
	guessed from output path.
variables,vars VARIABLES	Comma-separated list of names of variables
	to be included.
-M,method TEXT	Temporal resampling method. Available
	<pre>downsampling methods are 'count', 'first',</pre>
	'last', 'min', 'max', 'sum', 'prod', 'mean',
	<pre>'median', 'std', 'var', the upsampling methods are 'asfreq', 'ffill', 'bfill',</pre>
	'pad', 'nearest', 'interpolate'. If the
	upsampling method is 'interpolate', the
	option 'kind' will be used, if given.
	Other upsampling methods that select
	existing values honour the 'tolerance'
	option. Defaults to 'mean'.
-F,frequency TEXT	Temporal aggregation frequency. Use format
	" <count><offset>" where <offset> is one of</offset></offset></count>
	'H', 'D', 'W', 'M', 'Q', 'Y'. Use 'all' to
	aggregate all time steps included in the
-0,offset TEXT	dataset.Defaults to '1D'. Offset used to adjust the resampled time
-0,OIISEC IEXI	labels. Uses same syntax as frequency. Some
	Pandas date offset strings are supported as
	well.
-B,base INTEGER	For frequencies that evenly subdivide 1 day,
	the origin of the aggregated intervals. For
	example, for '24H' frequency, base could
	range from 0 through 23. Defaults to 0.
-K,kind TEXT	Interpolation kind which will be used if
	upsampling method is 'interpolation'. May be
	<pre>one of 'zero', 'slinear', 'quadratic', 'cubic', 'linear', 'nearest', 'previous',</pre>
	'next' where 'zero', 'slinear', 'quadratic',
	'cubic' refer to a spline interpolation of
	zeroth, first, second or third order;
	'previous' and 'next' simply return the
	previous or next value of the point. For
	more info refer to
	<pre>scipy.interpolate.interp1d(). Defaults to</pre>
	'linear'.
-T,tolerance TEXT	Tolerance for selective upsampling methods.
	Uses same syntax as frequency. If the time delta exceeds the tolerance, fill values
	(NaN) will be used. Defaults to the given
	frequency.
-q,quiet	Disable output of log messages to the
	console entirely. Note, this will also
	suppress error and warning messages.
-v,verbose	Enable output of log messages to the
	console. Has no effect if quiet/-q is

(continues on next page)

(continued from previous page)

```
used. May be given multiple times to control
the level of log messages, i.e., -v refers
to level INFO, -vv to DETAIL, -vvv to DEBUG,
-vvvv to TRACE. If omitted, the log level of
the console is WARNING.
--dry-run Just read and process inputs, but don't
produce any outputs.
--help Show this message and exit.
```

Examples

Upsampling example:

```
$ xcube resample --vars conc_chl,conc_tsm -F 12H -T 6H -M interpolation -K linear_
→examples/serve/demo/cube.nc
```

Downsampling example:

Python API

The related Python API function is xcube.core.resample.resample_in_time().

4.6.7 xcube vars2dim

Synopsis

Convert cube variables into new dimension.

\$ xcube vars2dim --help

```
Usage: xcube vars2dim [OPTIONS] CUBE
  Convert cube variables into new dimension. Moves all variables of CUBE
  into a single new variable <var-name> with a new dimension DIM-NAME and
  writes the results to OUTPUT.
Options:
  --variable, --var VARIABLE Name of the new variable that includes all
                              variables. Defaults to "data".
                              Name of the new dimension into variables.
  -D, --dim_name DIM-NAME
                              Defaults to "var".
  -o, --output OUTPUT
                              Output path. If omitted, 'INPUT-vars2dim.FORMAT'
                              will be used.
  -f, --format FORMAT
                              Format of the output. If not given, guessed from
                              OUTPUT.
  --help
                              Show this message and exit.
```

Python API

The related Python API function is xcube.core.vars2dim.vars_to_dim().

4.7 Cube conversion

4.8 Cube publication

4.8.1 xcube serve

Synopsis

Serve data cubes via web service.

xcube serve starts a light-weight web server that provides various services based on xcube datasets:

- Catalogue services to query for xcube datasets and their variables and dimensions, and feature collections;
- Tile map service, with some OGC WMTS 1.0 compatibility (REST and KVP APIs);
- Dataset services to extract subsets like time-series and profiles for e.g. JavaScript clients.

\$ xcube serve --help

Usage: xcube serve [OPTIONS] [PATHS...] Run the xcube Server for the given configuration and/or the given raster dataset paths given by PATHS. Each of the PATHS arguments can point to a raster dataset such as a Zarr directory (*.zarr), an xcube multi-level Zarr dataset (*.levels), a NetCDF file (*.nc), or a GeoTIFF/COG file (*.tiff). If one of PATHS is a directory that is not a dataset itself, it is scanned **for** readable raster datasets. The --show ASSET option can be used to inspect the current configuration of the server. ASSET is one of: outputs the list of APIs provided by the server apis endpoints outputs the list of all endpoints provided by the server openapi outputs the OpenAPI document representing this server config outputs the effective server configuration outputs the JSON Schema for the server configuration configschema The ASSET may be suffixed by ".yaml" or ".json" forcing the respective output format. The default format is YAML. Note, if --show is provided, the ASSET will be shown and the program will exit immediately.

Options:

(continues on next page)

(continued from previous page)

```
--framework FRAMEWORK
                                  Web server framework. Defaults to "tornado"
                                  Service port number. Defaults to 8080
 -p, --port PORT
 -a, --address ADDRESS
                                  Service address. Defaults to "0.0.0.0".
 -c, --config CONFIG
                                  Configuration YAML or JSON file. If
                                  multiple configuration files are passed, they will be
→merged in
                                  order.
 --base-dir BASE_DIR
                                  Directory used to resolve relative paths in
                                  CONFIG files. Defaults to the parent
                                  directory of (last) CONFIG file.
 --prefix URL_PREFIX
                                  Prefix path to be used for all endpoint
                                  URLs. May include template variables, e.g.,
                                  "api/{version}".
 --revprefix REVERSE_URL_PREFIX
                                 Prefix path to be used for reverse endpoint
                                  URLs that may be reported by server
                                  responses. May include template variables,
                                  e.g., "/proxy/{port}". Defaults to value of
                                  URL_PREFIX.
 --traceperf
                                  Whether to output extra performance logs.
 --update-after TIME
                                  Check for server configuration updates every
                                  TIME seconds.
                                  Unconditionally stop service after TIME
 --stop-after TIME
                                  seconds.
 --show ASSET
                                  Show ASSET and exit. Possible values for
                                  ASSET are 'apis', 'endpoints', 'openapi',
                                  'config', 'configschema' optionally suffixed
                                  by '.yaml' or '.json'.
 --open-viewer
                                  After starting the server, open xcube Viewer
                                  in a browser tab.
                                  Disable output of log messages to the
 -q, --quiet
                                  console entirely. Note, this will also
                                  suppress error and warning messages.
 -v, --verbose
                                  Enable output of log messages to the
                                  console. Has no effect if --quiet/-q is
                                  used. May be given multiple times to control
                                  the level of log messages, i.e., -v refers
                                  to level INFO, -vv to DETAIL, -vvv to DEBUG,
                                  -vvvv to TRACE. If omitted, the log level of
                                  the console is WARNING.
 --help
                                  Show this message and exit.
```

Configuration File

The xcube server is used to configure the xcube datasets to be published.

xcube datasets are any datasets that

- that comply to Unidata's CDM and to the CF Conventions;
- that can be opened with the xarray Python library;
- that have variables that have the dimensions and shape (lat, lon) or (time, lat, lon);
- that have 1D-coordinate variables corresponding to the dimensions;

• that have their spatial grid defined in arbitrary spatial coordinate reference systems.

The xcube server supports xcube datasets stored as local NetCDF files, as well as Zarr directories in the local file system or remote object storage. Remote Zarr datasets must be stored AWS S3 compatible object storage.

As an example, here is the configuration of the demo server. The parts of the demo configuration file are explained in detail further down.

Some hints before, which are not addressed in the server demo configuration file. To increase imaging performance, xcube datasets can be converted to multi-resolution pyramids using the *xcube level* tool. In the configuration, the format must be set to 'levels'. Leveled xcube datasets are configured this way:

```
Datasets:
    - Identifier: my_multi_level_dataset
    Title: My Multi-Level Dataset
    FileSystem: file
    Path: my_multi_level_dataset.levels
```

- ...

To increase time-series extraction performance, xcube datasets may be rechunked with larger chunk size in the time dimension using the *xcube chunk* tool. In the xcube server configuration a hidden dataset is given, and the it is referred to by the non-hidden, actual dataset using the TimeSeriesDataset setting:

```
Datasets:
- Identifier: my_dataset
Title: My Dataset
FileSystem: file
Path: my_dataset.zarr
TimeSeriesDataset: my_dataset_opt_for_ts
- Identifier: my_dataset_opt_for_ts
Title: My Dataset optimized for Time-Series
FileSystem: file
Path: my_ts_opt_dataset.zarr
Hidden: True
```

Server Demo Configuration File

The server configuration file consists of various parts, some of them are necessary others are optional. Here the demo configuration file used in the *example* is explained in detail.

The configuration file consists of five main parts authentication, dataset attribution, datasets, place groups and styles.

Authentication [optional]

In order to display data via xcube-viewer exclusively to registered and authorized users, the data served by xcube serve may be protected by adding Authentication to the server configuration. In order to ensure protection, an *Authority* and an *Audience* needs to be provided. Here authentication by Auth0 is used. Please note the trailing slash in the "Authority" URL.

```
Authentication:
Authority: https://xcube-dev.eu.auth0.com/
Audience: https://xcube-dev/api/
```

Example of OIDC configuration for Keycloak. Please note that there is no trailing slash in the "Authority" URL.

```
Authentication:
Authority: https://kc.brockmann-consult.de/auth/realms/AVL
Audience: avl-xc-api
```

Dataset Attribution [optional]

Dataset Attribution may be added to the server via DatasetAttribution.

```
DatasetAttribution:
    - "© by Brockmann Consult GmbH 2020, contains modified Copernicus Data 2019, processed_

    →by ESA"
    - "© by EU H2020 CyanoAlert project"
```

Base Directory [optional]

A typical xcube server configuration comprises many paths, and relative paths of known configuration parameters are resolved against the base_dir configuration parameter.

```
base_dir: s3://<bucket>/<path-to-your>/<resources>/
```

However, for values of parameters passed to user functions that represent paths in user code, this cannot be done automatically. For such situations, expressions can be used. An expression is any string between "\${"` and `"}" in a configuration value. An expression can contain the variables base_dir (a string), ctx the current server context (type xcube.webapi.datasets.DatasetsContext), as well as the function resolve_config_path(path) that is used to make a path absolut with respect to base_dir and to normalize it. For example

```
Augmentation:
Path: augmentation/metadata.py
Function: metadata:update_metadata
InputParameters:
    bands_config: ${resolve_config_path("../common/bands.yaml")}
```

Viewer Configuration [optional]

The xcube server endpoint /viewer/config/{*path} allows for configuring the viewer accessible via endpoint / viewer. The actual source for the configuration items is configured by xcube server configuration using the new entry Viewer/Configuration/Path, for example:

```
Viewer:
Configuration:
Path: s3://<bucket>/<viewer-config-dir-path>
```

Path [mandatory] must be an absolute filesystem path or a S3 path as in the example above. It points to a directory that is expected to contain the the viewer configuration file *config.json* among other configuration resources, such as custom favicon.ico or logo.png. The file config.json should conform to the [configuration JSON Schema](https://github.com/dcs4cop/xcube-viewer/blob/master/src/resources/config.schema.json). All its values are optional, if not provided, [default values](https://github.com/dcs4cop/xcube-viewer/blob/master/src/resources/config.json) are used instead.

Datasets [mandatory]

In order to publish selected xcube datasets via xcube serve, the datasets need to be described in the server configuration.

Remotely stored xcube Datasets

The following configuration snippet demonstrates how to publish static (persistent) xcube datasets stored in S3-compatible object storage:

```
Datasets:
  - Identifier: remote
    Title: Remote OLCI L2C cube for region SNS
    BoundingBox: [0.0, 50, 5.0, 52.5]
    FileSystem: s3
    Endpoint: "https://s3.eu-central-1.amazonaws.com"
    Path: xcube-examples/OLCI-SNS-RAW-CUBE-2.zarr
    Region: eu-central-1
    Anonymous: true
    Style: default
    ChunkCacheSize: 250M
    PlaceGroups:
      - PlaceGroupRef: inside-cube
      - PlaceGroupRef: outside-cube
    AccessControl:
      RequiredScopes:

    read:datasets
```

The above example of how to specify a xcube dataset to be served above is using a datacube stored in an S3 bucket within the Amazon Cloud. Please have a closer look at the parameter *Anonymous: true*. This means, the datasets permissions are set to public read in your source s3 bucket. If you have a dataset that is not public-read, set *Anonymous: false*. Furthermore, you need to have valid credentials on the machine where the server runs. Credentials may be saved either in a file called .aws/credentials with content like below:

[default]

aws_access_key_id=AKIAIOSFODNN7EXAMPLE
aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

Or they may be exported as environment variables AWS_SECRET_ACCESS_KEY and AWS_ACCESS_KEY_ID.

Further down an example for a *locally stored xcube datasets* will be given, as well as an example of *dynamic xcube datasets*.

Identifier [mandatory] is a unique ID for each xcube dataset, it is ment for machine-to-machine interaction and therefore does not have to be a fancy human-readable name.

Title [optional] should be understandable for humans. This title that will be displayed within the viewer for the dataset selection. If omitted, the key title from the dataset metadata will be used. If that is missing too, the identifier will be used.

BoundingBox [optional] may be set in order to restrict the region which is served from a certain datacube. The notation of the *BoundingBox* is [lon_min,lat_min,lon_max,lat_max].

FileSystem [mandatory] is set to "s3" which lets xcube serve know, that the datacube is located in the cloud.

Endpoint [mandatory] contains information about the cloud provider endpoint, this will differ if you use a different cloud provider.

Path [mandatory] leads to the specific location of the datacube. The particular datacube is stored in an OpenTelecom-Cloud S3 bucket called "xcube-examples" and the datacube is called "OLCI-SNS-RAW-CUBE-2.zarr".

Region [optional] is the region where the specified cloud provider is operating.

Styles [optional] influence the visualization of the xucbe dataset in the xcube viewer if specified in the server configuration file. The usage of *Styles* is described in section *styles*.

PlaceGroups [optional] allow to associate places (e.g. polygons or point-location) with a particular xcube dataset. Several different place groups may be connected to a xcube dataset, these different place groups are distinguished by the *PlaceGroupRef.* The configuration of *PlaceGroups* is described in section *place groups*.

AccessControl [optional] can only be used when providing *authentication*. Datasets may be protected by configuring the *RequiredScopes* entry whose value is a list of required scopes, e.g. "read:datasets".

Variables [optional] enforces the order of variables reported by xcube server. Is a list of wildcard patterns that determines the order of variables and the subset of variables to be reported.

The following example reports only variables whose name starts with "conc_":

```
Datasets:
- Path: demo.zarr
Variables:
- "conc_*"
```

The next example reports all variables but ensures that conc_chl and conc_tsm are the first ones:

```
Datasets:
  - Path: demo.zarr
  Variables:
        - conc_chl
        - conc_tsm
        _ "*"
```

Locally stored xcube Datasets

The following configuration snippet demonstrates how to publish static (persistent) xcube datasets stored in the local filesystem:

```
- Identifier: local
 Title: Local OLCI L2C cube for region SNS
 BoundingBox: [0.0, 50, 5.0, 52.5]
 FileSystem: file
 Path: cube-1-250-250.zarr
 Style: default
 TimeSeriesDataset: local_ts
 Augmentation:
   Path: compute_extra_vars.py
   Function: compute_variables
   InputParameters:
      factor_chl: 0.2
      factor_tsm: 0.7
 PlaceGroups:
    - PlaceGroupRef: inside-cube
    - PlaceGroupRef: outside-cube
 AccessControl:
   IsSubstitute: true
```

Most of the configuration of locally stored datasets is equal to the configuration of *remotely stored xcube datasets*.

FileSystem [mandatory] is set to "file" which lets xcube serve know, that the datacube is locally stored.

TimeSeriesDataset [optional] is not bound to local datasets, this parameter may be used for remotely stored datasets as well. By using this parameter a time optimized datacube will be used for generating the time series. The configuration of this time optimized datacube is shown below. By adding *Hidden* with *true* to the dataset configuration, the time optimized datacube will not appear among the displayed datasets in xcube viewer.

```
# Will not appear at all, because it is a "hidden" resource
- Identifier: local_ts
Title: 'local' optimized for time-series
BoundingBox: [0.0, 50, 5.0, 52.5]
FileSystem: file
Path: cube-5-100-200.zarr
Hidden: true
Style: default
```

Augmentation [optional] augments data cubes by new variables computed on-the-fly, the generation of the on-the-fly variables depends on the implementation of the python module specified in the *Path* within the *Augmentation* configuration.

AccessControl [optional] can only be used when providing *authentication*. By passing the *IsSubstitute* flag a dataset disappears for authorized requests. This might be useful for showing a demo dataset in the viewer for user who are not logged in.

Dynamic xcube Datasets

There is the possibility to define dynamic xcube datasets that are computed on-the-fly. Given here is an example that obtains daily or weekly averages of an xcube dataset named "local".

```
- Identifier: local_1w
 Title: OLCI weekly L3 cube for region SNS computed from local L2C cube
 BoundingBox: [0.0, 50, 5.0, 52.5]
 FileSystem: memory
 Path: resample_in_time.py
 Function: compute_dataset
 InputDatasets: ["local"]
 InputParameters:
   period: 1W
   incl_stdev: True
 Style: default
 PlaceGroups:
   - PlaceGroupRef: inside-cube
   - PlaceGroupRef: outside-cube
 AccessControl:
   IsSubstitute: True
```

FileSystem [mandatory] must be "memory" for dynamically generated datasets.

Path [mandatory] points to a Python module. Can be a Python file, a package, or a Zip file.

Function [mandatory, mutually exclusive with *Class*] references a function in the Python file given by *Path*. Must be suffixed by colon-separated module name, if *Path* references a package or Zip file. The function receives one or more datasets of type xarray.Dataset as defined by *InputDatasets* and optional keyword-arguments as given by *InputParameters*, if any. It must return a new xarray.Dataset with same spatial coordinates as the inputs. If "resample_in_time.py" is compressed among any other modules in a zip archive, the original module name must be indicated by the prefix to the function name:

```
Path: modules.zip
Function: resample_in_time:compute_dataset
InputDatasets: ["local"]
```

Class [mandatory, mutually exclusive with *Function*] references a callable in the Python file given by *Path*. Must be suffixed by colon-separated module name, if *Path* references a package or Zip file. The callable is either a class derived from xcube.core.mldataset.MultiLevelDataset or a function that returns an instance of xcube.core.mldataset.MultiLevelDataset. The callable receives one or more datasets of type xcube.core.mldataset.MultiLevelDataset as defined by *InputDatasets* and optional keyword-arguments as given by *InputParameters*, if any.

InputDatasets [mandatory] specifies the input datasets passed to Function or Class.

InputParameters [mandatory] specifies optional keyword arguments passed to *Function* or *Class*. In the example, *InputParameters* defines which kind of resampling should be performed.

Again, the dataset may be associated with place groups.

Place Groups [optional]

Place groups are specified in a similar manner compared to specifying datasets within a server. Place groups may be stored e.g. in shapefiles or a geoJson.

```
PlaceGroups:
- Identifier: outside-cube
Title: Points outside the cube
Path: places/outside-cube.geojson
PropertyMapping:
    image: ${resolve_config_path("images/outside-cube/${ID}.jpg")}
```

Identifier [mandatory] is a unique ID for each place group, it is the one xcube serve uses to associate a place group to a particular dataset.

Title [mandatory] should be understandable for humans and this is the title that will be displayed within the viewer for the place selection if the selected xcube dataset contains a place group.

Path [mandatory] defines where the file storing the place group is located. Please note that the paths within the example config are relative.

PropertyMapping [mandatory] determines which information contained within the place group should be used for selecting a certain location of the given place group. This depends very strongly of the data used. In the above example, the image URL is determined by a feature's ID property.

Property Mappings

The entry *PropertyMapping* is used to map a set of well-known properties (or roles) to the actual properties provided by a place feature in a place group. For example, the well-known properties are used to in xcube viewer to display information about the currently selected place. The possible well-known properties are:

- label: The property that provides a label for the place, if any. Defaults to to case-insensitive names label, title, name, id in xcube viewer.
- color: The property that provides a place's color. Defaults to the case-insensitive name color in xcube viewer.
- image: The property that provides a place's image URL, if any. Defaults to case-insensitive names image, img, picture, pic in xcube viewer.
- description: The property that provides a place's description text, if any. Defaults to case-insensitive names description, desc, abstract, comment in xcube viewer.

In the following example, a place's label is provided by the place feature's NAME property, while an image is provided by the place feature's IMG_URL property:

```
PlaceGroups:
    Identifier: my_group
    ...
    PropertyMapping:
        label: NAME
        image: IMG_URL
```

The values on the right side may either **be** feature property names or **contain** them as placeholders in the form \${PROPERTY}.

Styles [optional]

Within the *Styles* section, colorbars may be defined which should be used initially for a certain variable of a dataset, as well as the value ranges. For xcube viewer version 0.3.0 or higher the colorbars and the value ranges may be adjusted by the user within the xcube viewer.

```
Styles:
  - Identifier: default
    ColorMappings:
      conc_chl:
        ColorBar: plasma
        ValueRange: [0., 24.]
      conc_tsm:
        ColorBar: PuBuGn
        ValueRange: [0., 100.]
      kd489:
        ColorBar: jet
        ValueRange: [0., 6.]
      rgb:
        Red:
          Variable: conc_chl
          ValueRange: [0., 24.]
        Green:
          Variable: conc_tsm
          ValueRange: [0., 100.]
        Blue:
          Variable: kd489
          ValueRange: [0., 6.]
```

The *ColorMapping* may be specified for each variable of the datasets to be served. If not specified, xcube server will try to extract default values from attributes of dataset variables. The default value ranges are determined by:

- xcube-specific variable attributes color_value_min and color_value_max;
- The CF variable attributes valid_min, valid_max or valid_range.
- Or otherwise, the value range [0, 1] is assumed.

The colorbar name can be set using the

- xcube-specific variable attribute color_bar_name;
- Otherwise, the default colorbar name will be viridis.

The special name *rgb* may be used to generate an RGB-image from any other three dataset variables used for the individual *Red*, *Green* and *Blue* channels of the resulting image. An example is shown in the configuration above.

Colormaps may be reversed by using name suffix "_r". They also can have alpha blending indicated by name suffix "_alpha". Both, reversed and alpha blending is possible as well and can be configured by name suffix "_r_alpha".

```
Styles:
    - Identifier: default
    ColorMappings:
        conc_chl:
        ColorBar: plasma_r_alpha
        ValueRange: [0., 24.]
```

Example

```
xcube serve --port 8080 --config ./examples/serve/demo/config.yml --verbose
```

Server Demo Configuration File for DataStores

The server configuration file consists of various parts, some of them are necessary, others are optional. Here the demo stores configuration file used in the *example stores* is explained in detail.

This configuration file differs only in one part compared to section *Server Demo Configuration File: data stores*. The other main parts (*authentication, dataset attribution, place groups*, and *styles*) can be used in combination with *data stores*.

DataStores [mandatory]

Datasets, which are stored in the same location, may be configured in the configuration file using DataStores.

```
DataStores:
- Identifier: edc
StoreId: s3
StoreParams:
    root: xcube-dcfs/edc-xc-viewer-data
    max_depth: 1
    storage_options:
        anon: true
        # client_kwargs:
        # endpoint_url: https://s3.eu-central-1.amazonaws.com
Datasets:
        - Path: "*2.zarr"
        Style: default
        # ChunkCacheSize: 1G
```

Identifier [mandatory] is a unique ID for each DataStore.

StoreID [mandatory] can be file for locally stored datasets and s3 for datasets located in the cloud.

StoreParams [mandatory]

root [mandatory] states a common beginning part of the paths of the served datasets.

max_depth [optional] if wildcard is used in *Dataset Path* this indicated how far the server should step down and serve the discovered datasets.

storage_options [optional] is necessary when serving datasets from the cloud. With *anon* the accessibility is configured, if the datasets are public-read, *anon* is set to "true", "false" indicates they are protected. Credentials may be set by keywords *key* and *secret*.

Datasets [optional] if not specified, every dataset in the indicated location supported by xcube will be read and served by xcube serve. In order to filter certain datasets you can list Paths that shall be served by xcube serve. *Path* may contain wildcards. Each Dataset entry may have *Styles* and *PlaceGroups* associated with them, the same way as described in section *Server Demo Configuration File*.

Example Stores

xcube serve --port 8080 --config ./examples/serve/demo/config-with-stores.yml --verbose

Example Azure Blob Storage filesystem Stores

xcube server includes support for Azure Blob Storage filesystem by a data store *abfs*. This enables access to data cubes (*.zarr* or *.levels*) in Azure blob storage as shown here:

```
DataStores:
- Identifier: siec
StoreId: abfs
StoreParams:
    root: my_blob_container
    max_depth: 1
    storage_options:
    anon: true
    account_name: "xxx"
    account_key': "xxx"
    # or
    # connection_string: "xxx"
Datasets:
    - Path: "*.levels"
    Style: default
```

Web API

The xcube server has a dedicated self describing Web API Documentation. After starting the server, you can check the various functions provided by xcube Web API. To explore the functions, open

hase-url>/openapi.html.

The xcube server implements the OGC WMTS RESTful and KVP architectural styles of the OGC WMTS 1.0.0 specification. The following operations are supported:

- GetCapabilities: /xcube/wmts/1.0.0/WMTSCapabilities.xml
- GetTile: /xcube/wmts/1.0.0/tile/{DatasetName}/{VarName}/{TileMatrix}/{TileCol}/ {TileRow}.png
- GetFeatureInfo: in progress

CHAPTER

FIVE

PYTHON API

5.1 Data Store Framework

5.1.1 Functions

 $\texttt{xcube.core.store.new_data_store(data_store_id: str, extension_registry: ExtensionRegistry | None = None, **data_store_params) \rightarrow DataStore | MutableDataStore}$

Create a new data store instance for given *data_store_id* and *data_store_params*.

Parameters

- data_store_id (str) A data store identifier.
- **extension_registry** Optional extension registry. If not given, the global extension registry will be used.
- data_store_params Data store specific parameters.

Returns

A new data store instance

 $\begin{aligned} \texttt{xcube.core.store.new_fs_data_store}(\textit{protocol: str, root: str} = ", \textit{max_depth: int} | \textit{None} = 1, \textit{read_only: bool} \\ = \textit{False, includes: Sequence[str] | None} = \textit{None, excludes:} \\ & \textit{Sequence[str] | None} = \textit{None, storage_options: Dict[str, Any]} = \textit{None}) \\ & \rightarrow \textit{FsDataStore} \end{aligned}$

Create a new instance of a filesystem-based data store.

The data store is capable of filtering the data identifiers reported by get_data_ids(). For this purpose the optional keywords *excludes* and *includes* are used which can both take the form of a wildcard pattern or a sequence of wildcard patterns:

- excludes: if given and if any pattern matches the identifier, the identifier is not reported.
- includes: if not given or if any pattern matches the identifier, the identifier is reported.

Return type

FsDataStore

- **protocol** (str) The filesystem protocol, for example "file", "s3", "memory".
- **root** (str) Root or base directory. Defaults to "".
- **max_depth** Maximum recursion depth. None means limitless. Defaults to 1.
- **read_only** (bool) Whether this is a read-only store. Defaults to False.

- **includes** Optional sequence of wildcards that include certain filesystem paths. Affects the data identifiers (paths) returned by *get_data_ids()*. By default, all paths are included.
- **excludes** Optional sequence of wildcards that exclude certain filesystem paths. Affects the data identifiers (paths) returned by *get_data_ids()*. By default, no paths are excluded.
- **storage_options** Options specific to the underlying filesystem identified by *protocol*. Used to instantiate the filesystem.

Returns

A new data store instance of type :class:FsDataStore.

Find data store extensions using the optional filter function predicate.

Parameters

- **predicate** An optional filter function.
- extension_registry Optional extension registry. If not given, the global extension registry will be used.

Returns

List of data store extensions.

xcube.core.store.get_data_store_class($data_store_id: str, extension_registry: ExtensionRegistry | None = None$) \rightarrow Type[DataStore] | Type[MutableDataStore]

Get the class for the data store identified by *data_store_id*.

Parameters

- data_store_id (str) A data store identifier.
- **extension_registry** Optional extension registry. If not given, the global extension registry will be used.

Returns

The class for the data store.

xcube.core.store.get_data_store_params_schema($data_store_id: str, extension_registry:$ ExtensionRegistry | None = None) \rightarrow JsonObjectSchema

Get the JSON schema for instantiating a new data store identified by *data_store_id*.

Return type

JsonObjectSchema

Parameters

- data_store_id (str) A data store identifier.
- **extension_registry** Optional extension registry. If not given, the global extension registry will be used.

Returns

The JSON schema for the data store's parameters.

5.1.2 Classes

class xcube.core.store.DataStore

A data store represents a collection of data resources that can be enumerated, queried, and opened in order to obtain in-memory representations of the data. The same data resource may be made available using different data types. Therefore, many methods allow specifying a *data_type* parameter.

A store implementation may use any existing openers/writers, or define its own, or not use any openers/writers at all.

Store implementers should follow the conventions outlined in https://xcube.readthedocs.io/en/latest/storeconv. html .

The :class:DataStore is an abstract base class that both read-only and mutable data stores must implement.

classmethod get_data_store_params_schema() \rightarrow JsonObjectSchema

Get descriptions of parameters that must or can be used to instantiate a new DataStore object. Parameters are named and described by the properties of the returned JSON object schema. The default implementation returns JSON object schema that can have any properties.

abstract classmethod get_data_types() \rightarrow Tuple[str, ...]

Get alias names for all data types supported by this store. The first entry in the tuple represents this store's default data type.

Returns

The tuple of supported data types.

abstract get_data_types_for_data(*data_id: str*) \rightarrow Tuple[str, ...]

Get alias names for of data types that are supported by this store for the given data_id.

Parameters

data_id (str) – An identifier of data that is provided by this store

Returns

A tuple of data types that apply to the given *data_id*.

Raises

DataStoreError - If an error occurs.

abstract get_data_ids($data_type: str | None | type | DataType = None, include_attrs: Container[str] = None) <math>\rightarrow$ Iterator[str] | Iterator[Tuple[str, Dict[str, Any]]]

Get an iterator over the data resource identifiers for the given type *data_type*. If *data_type* is omitted, all data resource identifiers are returned.

If a store implementation supports only a single data type, it should verify that *data_type* is either None or compatible with the supported data type.

If *include_attrs* is provided, it must be a sequence of names of metadata attributes. The store will then return extra metadata for each returned data resource identifier according to the names of the metadata attributes as tuples (*data_id, attrs*).

Hence, the type of the returned iterator items depends on the value of *include_attrs*:

- If *include_attrs* is None (the default), the method returns an iterator of dataset identifiers *data_id* of type *str*.
- If *include_attrs* is a sequence of attribute names, even an empty one, the method returns an iterator of tuples (*data_id, attrs*) of type *Tuple[str, Dict]*, where *attrs* is a dictionary filled according to the names in *include_attrs*. If a store cannot provide a given attribute, it should simply ignore it. This may even yield to an empty dictionary for a given *data_id*.

The individual attributes do not have to exist in the dataset's metadata, they may also be generated on-the-fly. An example for a generic attribute name is "title". A store should try to resolve include_attrs=["title"] by returning items such as ("ESACCI-L4_GHRSST-SSTdepth-OSTIA-GLOB_CDR2.1-v02.0-fv01.0.zarr", {"title": "Level-4 GHRSST Analysed Sea Surface Temperature"}).

Parameters

- **data_type** If given, only data identifiers that are available as this type are returned. If this is omitted, all available data identifiers are returned.
- **include_attrs** A sequence of names of attributes to be returned for each dataset identifier. If given, the store will attempt to provide the set of requested dataset attributes in addition to the data ids. (added in xcube 0.8.0)

Returns

An iterator over the identifiers and titles of data resources provided by this data store.

Raises

DataStoreError – If an error occurs.

list_data_ids(*data_type: str* | *None* | *type* | *DataType* = *None*, *include_attrs:* Container[str] = None) \rightarrow List[str] | List[Tuple[str, Dict[str, Any]]]

Convenience version of get_data_ids() that returns a list rather than an iterator.

Parameters

- **data_type** If given, only data identifiers that are available as this type are returned. If this is omitted, all available data identifiers are returned.
- **include_attrs** A sequence of names of attributes to be returned for each dataset identifier. If given, the store will attempt to provide the set of requested dataset attributes in addition to the data ids. (added in xcube 0.8.0)

Returns

A list comprising the identifiers and titles of data resources provided by this data store.

Raises

DataStoreError – If an error occurs.

abstract has_data($data_id: str, data_type: str | None | type | DataType = None) \rightarrow bool$

Check if the data resource given by *data_id* is available in this store.

Return type

bool

Parameters

- data_id (str) A data identifier
- **data_type** An optional data type. If given, it will also be checked whether the data is available as the specified type. May be given as type alias name, as a type, or as a :class:DataType instance.

Returns

True, if the data resource is available in this store, False otherwise.

abstract describe_data(*data_id: str, data_type: str* | *None* | *type* | *DataType* = *None*) \rightarrow *DataDescriptor*

Get the descriptor for the data resource given by *data_id*.

Raises a :class:DataStoreError if *data_id* does not exist in this store or the data is not available as the specified *data_type*.

Return type

DataDescriptor

Parameters

- data_id (str) An identifier of data provided by this store
- **data_type** If given, the descriptor of the data will describe the data as specified by the data type. May be given as type alias name, as a type, or as a :class:DataType instance.

:return a data-type specific data descriptor :raise DataStoreError: If an error occurs.

abstract get_data_opener_ids($data_id: str = None, data_type: str | None | type | DataType = None) \rightarrow$ Tuple[str, ...]

Get identifiers of data openers that can be used to open data resources from this store.

If *data_id* is given, data accessors are restricted to the ones that can open the identified data resource. Raises if *data_id* does not exist in this store.

If *data_type* is given, only openers that are compatible with this data type are returned.

If a store implementation supports only a single data type, it should verify that *data_type* is either None or equal to that single data type.

Parameters

- data_id (str) An optional data resource identifier that is known to exist in this data store.
- **data_type** An optional data type that is known to be supported by this data store. May be given as type alias name, as a type, or as a :class:DataType instance.

Returns

A tuple of identifiers of data openers that can be used to open data resources.

Raises

DataStoreError – If an error occurs.

abstract get_open_data_params_schema($data_id: str = None, opener_id: str = None$) \rightarrow JsonObjectSchema

Get the schema for the parameters passed as *open_params* to :meth:open_data(data_id, open_params).

If *data_id* is given, the returned schema will be tailored to the constraints implied by the identified data resource. Some openers might not support this, therefore *data_id* is optional, and if it is omitted, the returned schema will be less restrictive. If given, the method raises if *data_id* does not exist in this store.

If *opener_id* is given, the returned schema will be tailored to the constraints implied by the identified opener. Some openers might not support this, therefore *opener_id* is optional, and if it is omitted, the returned schema will be less restrictive.

For maximum compatibility of stores, it is strongly encouraged to apply the following conventions on parameter names, types, and their interpretation.

Let P be the value of an optional, data constraining open parameter, then it should be interpreted as follows:

- _if P is None_ means, parameter not given, hence no constraint applies, hence no additional restrictions on requested data.
- _if not **P**_ means, we exclude data that would be included by default.
- _else_, the given constraint applies.

Given here are names, types, and descriptions of common, constraining open parameters for gridded datasets. Note, whether any of these is optional or mandatory depends on the individual data store. A

store may also define other open parameters or support only a subset of the following. Note all parameters may be optional, the Python-types given here refer to _given_, non-Null parameters:

- variable_names: List[str]: Included data variables. Available coordinate variables will be auto-included for any dimension of the data variables.
- bbox: Tuple[float, float, float]: Spatial coverage as xmin, ymin, xmax, ymax.
- crs: str: Spatial CRS, e.g. "EPSG:4326" or OGC CRS URI.
- spatial_res: float: Spatial resolution in coordinates of the spatial CRS.
- time_range: Tuple[Optional[str], Optional[str]]: Time range interval in UTC date/time units using ISO format. Start or end time may be missing which means everything until available start or end time.
- ``time_period: str`: Pandas-compatible period/frequency string, e.g. "8D", "2W".

E.g. applied to an optional variable_names parameter, this means

- variable_names is None include all data variables
- *variable_names* == [] do not include data variables (schema only)
- *variable_names* == ["<*var_1*>", "<*var_2*>", ...] only include data variables named "<*var_1*>", "<*var_2*>", ...

Return type

JsonObjectSchema

Parameters

- **data_id** (str) An optional data identifier that is known to exist in this data store.
- **opener_id** (str) An optional data opener identifier.

Returns

The schema for the parameters in *open_params*.

Raises

DataStoreError - If an error occurs.

abstract open_data($data_id: str, opener_id: str = None, **open_params$) \rightarrow Any

Open the data given by the data resource identifier *data_id* using the supplied *open_params*.

The data type of the return value depends on the data opener used to open the data resource.

If *opener_id* is given, the identified data opener will be used to open the data resource and *open_params* must comply with the schema of the opener's parameters. Note that some store implementations may not support using different openers or just support a single one.

Raises if *data_id* does not exist in this store.

Return type

Any

Parameters

- data_id (str) The data identifier that is known to exist in this data store.
- **opener_id** (str) An optional data opener identifier.
- **open_params** Opener-specific parameters.

Returns

An in-memory representation of the data resources identified by *data_id* and *open_params*.

Raises

DataStoreError – If an error occurs.

class xcube.core.store.MutableDataStore

A mutable data store is a data store that also allows for adding, updating, and removing data resources.

MutableDataStore is an abstract base class that any mutable data store must implement.

abstract get_data_writer_ids(*data_type: str* | *None* | *type* | *DataType* = *None*) → Tuple[str, ...]

Get identifiers of data writers that can be used to write data resources to this store.

If *data_type* is given, only writers that support this data type are returned.

If a store implementation supports only a single data type, it should verify that *data_type* is either None or equal to that single data type.

Parameters

data_type – An optional data type specifier that is known to be supported by this data store. May be given as type alias name, as a type, or as a :class:DataType instance.

Returns

A tuple of identifiers of data writers that can be used to write data resources.

Raises

DataStoreError - If an error occurs.

abstract get_write_data_params_schema(writer_id: str = None) \rightarrow JsonObjectSchema

Get the schema for the parameters passed as *write_params* to :meth:write_data(data, data_id, open_params).

If *writer_id* is given, the returned schema will be tailored to the constraints implied by the identified writer. Some writers might not support this, therefore *writer_id* is optional, and if it is omitted, the returned schema will be less restrictive.

Given here is a pseudo-code implementation for stores that support multiple writers and where the store has common parameters with the writer:

```
store_params_schema = self.get_data_store_params_schema() writer_params_schema
= get_writer(writer_id).get_write_data_params_schema() return sub-
tract_param_schemas(writer_params_schema, store_params_schema)
```

Return type

JsonObjectSchema

Parameters

writer_id (str) – An optional data writer identifier.

Returns

The schema for the parameters in *write_params*.

Raises

DataStoreError – If an error occurs.

abstract write_data(*data: Any, data_id: str* = None, writer_id: str = None, replace: bool = False, **write_params) \rightarrow str

Write a data in-memory instance using the supplied data_id

and write_params.

If data identifier *data_id* is not given, a writer-specific default will be generated, used, and returned.

If *writer_id* is given, the identified data writer will be used to write the data resource and *write_params* must comply with the schema of writers's parameters. Note that some store implementations may not support using different writers or just support a single one.

Given here is a pseudo-code implementation for stores that support multiple writers:

```
writer_id = writer_id or self.gen_data_id() path = self.resolve_data_id_to_path(data_id)
write_params = add_params(self.get_data_store_params(), write_params)
get_writer(writer_id).write_data(data, path, **write_params) self.register_data(data_id, data)
```

Raises if *data_id* does not exist in this store.

Return type

str

Parameters

- data (Any) The data in-memory instance to be written.
- data_id (str) An optional data identifier that is known to be unique in this data store.
- writer_id (str) An optional data writer identifier.
- replace (bool) Whether to replace an existing data resource.
- write_params Writer-specific parameters.

Returns

The data identifier used to write the data.

Raises

DataStoreError – If an error occurs.

abstract delete_data(data_id: str, **delete_params)

Delete the data resource identified by *data_id*.

Typically, an implementation would delete the data resource from the physical storage and also remove any registered metadata from an associated database.

Raises if *data_id* does not exist in this store.

Parameters

data_id (str) – An data identifier that is known to exist in this data store.

Raises

DataStoreError – If an error occurs.

abstract register_data(data_id: str, data: Any)

Register the in-memory representation of a data resource *data* using the given data resource identifier *data_id*.

This method can be used to register data resources that are already physically stored in the data store, but are not yet searchable or otherwise accessible by the given *data_id*.

Typically, an implementation would extract metadata from *data* and store it in a store-specific database. An implementation should just store the metadata of *data*. It should not write *data*.

Parameters

- data_id (str) A data resource identifier that is known to be unique in this data store.
- data (Any) An in-memory representation of a data resource.

Raises

DataStoreError – If an error occurs.

abstract deregister_data(data_id: str)

De-register a data resource identified by *data_id* from this data store.

This method can be used to de-register data resources so it will be no longer searchable or otherwise accessible by the given *data_id*.

Typically, an implementation would extract metadata from *data* and store it in a store-specific database. An implementation should only remove a data resource's metadata. It should not delete *data* from its physical storage space.

Raises if *data_id* does not exist in this store.

Parameters data_id (str) – A data resource identifier that is known to exist in this data store.

Raises

DataStoreError – If an error occurs.

class xcube.core.store.DataOpener

An interface that specifies a parameterized *open_data()* operation.

Possible open parameters are implementation-specific and are described by a JSON Schema.

Note this interface uses the term "opener" to underline the expected laziness of the operation. For example, when a xarray.Dataset is returned from a Zarr directory, the actual data is represented by Dask arrays and will be loaded only on-demand.

abstract get_open_data_params_schema($data_id: str = None$) \rightarrow JsonObjectSchema

Get the schema for the parameters passed as *open_params* to :meth:open_data(data_id, open_params). If *data_id* is given, the returned schema will be tailored to the constraints implied by the identified data resource. Some openers might not support this, therefore *data_id* is optional, and if it is omitted, the returned schema will be less restrictive.

Return type

JsonObjectSchema

Parameters

data_id (str) – An optional data resource identifier.

Returns

The schema for the parameters in *open_params*.

Raises

DataStoreError - If an error occurs.

abstract open_data($data_id: str, **open_params$) \rightarrow Any

Open the data resource given by the data resource identifier *data_id* using the supplied *open_params*.

Raises if *data_id* does not exist.

Return type

Any

Parameters

- data_id (str) The data resource identifier.
- open_params Opener-specific parameters.

Returns

An xarray.Dataset instance.

Raises

DataStoreError - If an error occurs.

class xcube.core.store.DataSearcher

Allow searching data in a data store.

```
abstract classmethod get_search_params_schema(data_type: str | None | type | DataType = None)

\rightarrow JsonObjectSchema
```

Get the schema for the parameters that can be passed as *search_params* to :meth:search_data(). Parameters are named and described by the properties of the returned JSON object schema.

Return type

JsonObjectSchema

Parameters

data_type – If given, the search parameters will be tailored to search for data for the given *data_type*.

Returns

A JSON object schema whose properties describe this store's search parameters.

abstract search_data(data_type: str | None | type | DataType = None, **search_params) \rightarrow Iterator[DataDescriptor]

Search this store for data resources. If *data_type* is given, the search is restricted to data resources of that type.

Returns an iterator over the search results which are returned as :class:DataDescriptor objects.

If a store implementation supports only a single data type, it should verify that *data_type* is either None or compatible with the supported data type specifier.

Parameters

• data_type - An optional data type that is known to be supported by this data store.

• **search_params** – The search parameters.

Returns

An iterator of data descriptors for the found data resources.

Raises

DataStoreError – If an error occurs.

class xcube.core.store.DataWriter

An interface that specifies a parameterized *write_data()* operation.

Possible write parameters are implementation-specific and are described by a JSON Schema.

$abstract get_write_data_params_schema() \rightarrow JsonObjectSchema$

Get the schema for the parameters passed as *write_params* to :meth:write_data(data resource, data_id, open_params).

Returns

The schema for the parameters in *write_params*.

Raises

DataStoreError – If an error occurs.

abstract write_data(*data: Any, data_id: str, replace: bool = False, **write_params*) \rightarrow str

Write a data resource using the supplied *data_id* and *write_params*.

Return type str

- data (Any) The data resource's in-memory representation to be written.
- data_id (str) A unique data resource identifier.
- **replace** (bool) Whether to replace an existing data resource.
- write_params Writer-specific parameters.

Returns

The data resource identifier used to write the data resource.

Raises

DataStoreError – If an error occurs.

class xcube.core.store.DataStoreError(message: str)

Raised on error in any of the data store, opener, or writer methods.

Parameters

message – The error message.

A generic descriptor for any data. Also serves as a base class for more specific data descriptors.

Parameters

- data_id An identifier for the data
- data_type A type specifier for the data
- crs A coordinate reference system identifier, as an EPSG, PROJ or WKT string
- **bbox** A bounding box of the data
- time_range Start and end time delimiting this data's temporal extent
- time_period The data's periodicity if it is evenly temporally resolved.
- open_params_schema A JSON schema describing the parameters that may be used to open this data.

classmethod get_schema() \rightarrow JsonObjectSchema

Get JSON object schema.

class xcube.core.store.**DatasetDescriptor**(*data_id: str*, *, *data_type: str* | *None* | *type* | *DataType* =

'dataset', crs: str = None, bbox: Tuple[float, float, float, float] = None, time_range: Tuple[str | None, str | None] = None, time_period: str = None, spatial_res: float = None, dims: Mapping[str, int] = None, coords: Mapping[str, VariableDescriptor] = None, data_vars: Mapping[str, VariableDescriptor] = None, attrs: Mapping[Hashable, any] = None, open_params_schema: JsonObjectSchema = None, **additional_properties)

A descriptor for a gridded, N-dimensional dataset represented by xarray.Dataset. Comprises a description of the data variables contained in the dataset.

Regrading *time_range* and *time_period* parameters, please refer to https://github.com/dcs4cop/xcube/blob/ master/docs/source/storeconv.md#date-time-and-duration-specifications

- data_id An identifier for the data
- data_type The data type of the data described
- crs A coordinate reference system identifier, as an EPSG, PROJ or WKT string
- **bbox** A bounding box of the data
- time_range Start and end time delimiting this data's temporal extent
- time_period The data's periodicity if it is evenly temporally resolved
- spatial_res The spatial extent of a pixel in crs units
- dims A mapping of the dataset's dimensions to their sizes
- **coords** mapping of the dataset's data coordinate names to instances of :class:VariableDescriptor
- **data_vars** A mapping of the dataset's variable names to instances of :class:VariableDescriptor
- attrs A mapping containing arbitrary attributes of the dataset
- **open_params_schema** A JSON schema describing the parameters that may be used to open this data

classmethod get_schema() \rightarrow JsonObjectSchema

Get JSON object schema.

A descriptor for a gridded, N-dimensional, multi-level, multi-resolution dataset represented by xcube.core.mldataset.MultiLevelDataset.

Parameters

- data_id An identifier of the multi-level dataset
- num_levels The number of levels of this multi-level dataset
- **data_type** A type specifier for the multi-level dataset

classmethod get_schema() \rightarrow JsonObjectSchema

Get JSON object schema.

class xcube.core.store.**DatasetDescriptor**(*data_id: str, *, data_type: str | None | type | DataType =*

'dataset', crs: str = None, bbox: Tuple[float, float, float, float] = None, time_range: Tuple[str | None, str | None] = None, time_period: str = None, spatial_res: float = None, dims: Mapping[str, int] = None, coords: Mapping[str, VariableDescriptor] = None, data_vars: Mapping[str, VariableDescriptor] = None, attrs: Mapping[Hashable, any] = None, open_params_schema: JsonObjectSchema = None, **additional_properties)

A descriptor for a gridded, N-dimensional dataset represented by xarray.Dataset. Comprises a description of the data variables contained in the dataset.

Regrading *time_range* and *time_period* parameters, please refer to https://github.com/dcs4cop/xcube/blob/ master/docs/source/storeconv.md#date-time-and-duration-specifications

Parameters

• data_id - An identifier for the data

- data_type The data type of the data described
- crs A coordinate reference system identifier, as an EPSG, PROJ or WKT string
- **bbox** A bounding box of the data
- time_range Start and end time delimiting this data's temporal extent
- time_period The data's periodicity if it is evenly temporally resolved
- spatial_res The spatial extent of a pixel in crs units
- dims A mapping of the dataset's dimensions to their sizes
- **coords** mapping of the dataset's data coordinate names to instances of :class:VariableDescriptor
- **data_vars** A mapping of the dataset's variable names to instances of :class:VariableDescriptor
- attrs A mapping containing arbitrary attributes of the dataset
- open_params_schema A JSON schema describing the parameters that may be used to open this data

<code>classmethod get_schema()</code> \rightarrow JsonObjectSchema

Get JSON object schema.

A descriptor for dataset variable represented by xarray.DataArray instances. They are part of dataset descriptor for an gridded, N-dimensional dataset represented by xarray.Dataset.

Parameters

- **name** The variable name
- **dtype** The data type of the variable.
- dims A list of the names of the variable's dimensions.
- chunks A list of the chunk sizes of the variable's dimensions
- attrs A mapping containing arbitrary attributes of the variable

property ndim: int

Number of dimensions.

$\texttt{classmethod get_schema()} \rightarrow JsonObjectSchema$

Get JSON object schema.

None, **kwargs)

A descriptor for a geo-vector dataset represented by a geopandas.GeoDataFrame instance.

- data_id An identifier of the geopandas.GeoDataFrame
- feature_schema A schema describing the properties of the vector data
- kwargs Parameters passed to super :class:DataDescriptor

```
\texttt{classmethod get\_schema()} \rightarrow JsonObjectSchema
```

Get JSON object schema.

5.2 Cube generation

 $\begin{aligned} \texttt{xcube.core.gen.gen.gen_cube}(input_paths: Sequence[str] = None, input_processor_name: str = None, \\ input_processor_params: Dict = None, input_reader_name: str = None, \\ input_reader_params: Dict[str, Any] = None, output_region: Tuple[float, float, float, float] = None, output_size: Tuple[int, int] = [512, 512], \\ output_resampling: str = 'Nearest', output_path: str = 'out.zarr', \\ output_writer_name: str = None, output_variables: List[Tuple[str, Dict[str, Any] | None]] = None, processed_variables: List[Tuple[str, Dict[str, Any] | None]] = None, profile_mode: bool = False, no_sort_mode: bool = False, monitor: \\ Callable[[...], None] = None) \to bool \end{aligned}$

Generate a xcube dataset from one or more input files.

Return type bool

- no_sort_mode (bool) -
- **input_paths** The input paths.
- **input_processor_name** (str) Name of a registered input processor (xcube.core.gen.inputprocessor.InputProcessor) to be used to transform the inputs.
- input_processor_params Parameters to be passed to the input processor.
- **input_reader_name** (str) Name of a registered input reader (xcube.core.util.dsio.DatasetIO).
- **input_reader_params** Parameters passed to the input reader.
- output_region Output region as tuple of floats: (lon_min, lat_min, lon_max, lat_max).
- **output_size** The spatial dimensions of the output as tuple of ints: (width, height).
- **output_resampling** (str) The resampling method for the output.
- **output_path** (str) The output directory.
- **output_writer_name** (str) Name of an output writer (xcube.core.util.dsio.DatasetIO) used to write the cube.
- **output_writer_params** Parameters passed to the output writer.
- **output_metadata** Extra metadata passed to output cube.
- output_variables Output variables.
- processed_variables Processed variables computed on-the-fly.
- profile_mode (bool) Whether profiling should be enabled.
- **append_mode** (bool) Deprecated. The function will always either insert, replace, or append new time slices.
- **dry_run** (bool) Doesn't write any data. For testing.

• **monitor** – A progress monitor.

Returns

True for success.

xcube.core.new.new_cube(title='Test Cube', width=360, height=180, x_name='lon', y_name='lat',

x_dtype='float64', y_dtype=None, x_units='degrees_east', y_units='degrees_north', x_res=1.0, y_res=None, x_start=-180.0, y_start=-90.0, inverse_y=False, time_name='time', time_dtype='datetime64[s]', time_units='seconds since 1970-01-01T00:00:00', time_calendar='proleptic_gregorian', time_periods=5, time_freq='D', time_start='2010-01-01T00:00:00', use_cftime=False, drop_bounds=False, variables=None, crs=None, crs_name=None)

Create a new empty cube. Useful for creating cubes templates with predefined coordinate variables and metadata. The function is also heavily used by xcube's unit tests.

The values of the *variables* dictionary can be either constants, array-like objects, or functions that compute their return value from passed coordinate indexes. The expected signature is::

def my_func(time: int, y: int, x: int) -> Union[bool, int, float]

- title (str) A title. Defaults to 'Test Cube'.
- width (int) Horizontal number of grid cells. Defaults to 360.
- height (int) Vertical number of grid cells. Defaults to 180.
- **x_name** (str) Name of the x coordinate variable. Defaults to 'lon'.
- y_name (str) Name of the y coordinate variable. Defaults to 'lat'.
- **x_dtype** (str) Data type of x coordinates. Defaults to 'float64'.
- y_dtype Data type of y coordinates. Defaults to 'float64'.
- **x_units** (str) Units of the x coordinates. Defaults to 'degrees_east'.
- y_units (str) Units of the y coordinates. Defaults to 'degrees_north'.
- **x_start** (float) Minimum x value. Defaults to -180.
- **y_start** (float) Minimum y value. Defaults to -90.
- **x_res** (float) Spatial resolution in x-direction. Defaults to 1.0.
- y_res Spatial resolution in y-direction. Defaults to 1.0.
- inverse_y (bool) Whether to create an inverse y axis. Defaults to False.
- time_name (str) Name of the time coordinate variable. Defaults to 'time'.
- time_periods (int) Number of time steps. Defaults to 5.
- time_freq (str) Duration of each time step. Defaults to `1D'.
- time_start (str) First time value. Defaults to '2010-01-01T00:00:00'.
- **time_dtype** (str) Numpy data type for time coordinates. Defaults to 'datetime64[s]'. If used, parameter 'use_cftime' must be False.
- time_units (str) Units for time coordinates. Defaults to 'seconds since 1970-01-01T00:00:00'.
- time_calendar (str) Calender for time coordinates. Defaults to 'proleptic_gregorian'.

- **use_cftime** (bool) If True, the time will be given as data types according to the 'cftime' package. If used, the time_calendar parameter must be also be given with an appropriate value such as 'gregorian' or 'julian'. If used, parameter 'time_dtype' must be None.
- **drop_bounds** (bool) If True, coordinate bounds variables are not created. Defaults to False.
- variables Dictionary of data variables to be added. None by default.
- crs pyproj-compatible CRS string or instance of pyproj.CRS or None
- **crs_name** Name of the variable that will hold the CRS information. Ignored, if *crs* is not given.

Returns

A cube instance

5.3 Cube computation

 $\begin{aligned} \texttt{xcube.core.compute.compute_cube}(cube_func: ~typing.Callable[[...], ~xarray.core.dataarray.DataArray | ~numpy.ndarray | ~typing.Sequence[~xarray.core.dataarray.DataArray | ~numpy.ndarray]], *input_cubes: ~xarray.core.dataset.Dataset, input_cube_schema: ~xcube.core.schema.CubeSchema = None, input_var_names: ~typing.Sequence[str] = None, input_params: ~typing.Dict[str, ~typing.Any] = None, output_var_name: str = 'output', output_var_dtype: ~typing.Dict[str, ~typing.Any] = None, vectorize: bool = None, cube_asserted: bool = False) \rightarrow Dataset \end{aligned}$

Compute a new output data cube with a single variable named *output_var_name* from variables named *in-put_var_names* contained in zero, one, or more input data cubes in *input_cubes* using a cube factory function *cube_func*.

For a more detailed description of the function usage, please refer to :func:compute_dataset.

Return type

Dataset

Parameters

- **cube_func** The cube factory function.
- **input_cubes** (Dataset) An optional sequence of input cube datasets, must be provided if *input_cube_schema* is not.
- input_cube_schema (CubeSchema) An optional input cube schema,

must be provided if *input_cubes* is not. Will be ignored if *input_cubes* is provided. :param input_var_names: A sequence of variable names :param input_params: Optional dictionary with processing parameters passed to *cube_func*. :param output_var_name: Optional name of the output variable, defaults to 'output'. :param output_var_dtype: Optional numpy datatype of the output variable, defaults to 'float32'. :param output_var_attrs: Optional metadata attributes for the output variable. :param vectorize: Whether all *input_cubes* have the same variables which are concatenated and passed as vectors

to cube_func. Not implemented yet.

Parameters

cube_asserted – If False, *cube* will be verified, otherwise it is expected to be a valid cube.

Returns

A new dataset that contains the computed output variable.

xcube.core.evaluate.evaluate_dataset(dataset: Dataset, processed_variables: List[Tuple[str, Dict[str, Any] |None]] = None, errors: str = 'raise') \rightarrow Dataset

Compute new variables or mask existing variables in *dataset* by the evaluation of Python expressions, that may refer to other existing or new variables. Returns a new dataset that contains the old and new variables, where both may be now masked.

Expressions may be given by attributes of existing variables in *dataset* or passed a via the *processed_variables* argument which is a sequence of variable name / attributes tuples.

Two types of expression attributes are recognized in the attributes:

- 1. The attribute expression generates a new variable computed from its attribute value.
- 2. The attribute valid_pixel_expression masks out invalid variable values.

In both cases the attribute value must be a string that forms a valid Python expression that can reference any other preceding variables by name. The expression can also reference any flags defined by another variable according the their CF attributes flag_meaning and flag_values.

Invalid variable values may be masked out using the value the valid_pixel_expression attribute whose value should form a Boolean Python expression. In case, the expression returns zero or false, the value of the _FillValue attribute or NaN will be used in the new variable.

Other attributes will be stored as variable metadata as-is.

Return type Dataset

Parameters

- dataset (Dataset) A dataset.
- processed_variables Optional list of variable name-attributes pairs that will processed in the given order.
- **errors** (str) How to deal with errors while evaluating expressions. May be be one of "raise", "warn", or "ignore".

Returns

new dataset with computed variables

5.4 Cube data extraction

xcube.core.extract.get_cube_values_for_points(cube: Dataset, points: Dataset | DataFrame |

 $\begin{array}{l} Mapping[str, ndarray \mid Array \mid DataArray \mid Series \mid \\ Sequence[int \mid float]], var_names: Sequence[str] = \\ None, include_coords: bool = False, include_bounds: \\ bool = False, include_indexes: bool = False, \\ index_name_pattern: str = '{name}_index', \\ include_refs: bool = False, ref_name_pattern: str = '{name}_ref', method: str = 'nearest', cube_asserted: \\ bool = False) \rightarrow Dataset \end{array}$

Extract values from *cube* variables at given coordinates in *points*.

Returns a new dataset with values of variables from *cube* selected by the coordinate columns provided in *points*. All variables will be 1-D and have the same order as the rows in points.

Return type

Dataset

Parameters

- **cube** (Dataset) The cube dataset.
- points Dictionary that maps dimension name to coordinate arrays.
- **var_names** An optional list of names of data variables in *cube* whose values shall be extracted.
- **include_coords** (bool) Whether to include the cube coordinates for each point in return value.
- **include_bounds** (bool) Whether to include the cube coordinate boundaries (if any) for each point in return value.
- **include_indexes** (bool) Whether to include computed indexes into the cube for each point in return value.
- **index_name_pattern** (str) A naming pattern for the computed index columns. Must include "{name}" which will be replaced by the index' dimension name.
- **include_refs** (bool) Whether to include point (reference) values from *points* in return value.
- **ref_name_pattern** (str) A naming pattern for the computed point data columns. Must include "{name}" which will be replaced by the point's attribute name.
- **method** (str) "nearest" or "linear".
- **cube_asserted** (bool) If False, *cube* will be verified, otherwise it is expected to be a valid cube.

Returns

A new data frame whose columns are values from *cube* variables at given *points*.

xcube.core.extract.get_cube_point_indexes(cube: ~xarray.core.dataset.Dataset, points:

~xarray.core.dataset.Dataset | ~pandas.core.frame.DataFrame | ~typing.Mapping[str, ~numpy.ndarray | ~dask.array.core.Array | ~xarray.core.dataarray.DataArray | ~pandas.core.series.Series | ~typing.Sequence[int | float]], dim_name_mapping: ~typing.Mapping[str, str] = None, index_name_pattern: str = '{name}_index', index_dtype=<class 'numpy.float64'>, cube_asserted: bool = False) \rightarrow Dataset

Get indexes of given point coordinates points into the given dataset.

Return type

Dataset

- **cube** (Dataset) The cube dataset.
- **points** A mapping from column names to column data arrays, which must all have the same length.
- **dim_name_mapping** A mapping from dimension names in *cube* to column names in *points*.
- **index_name_pattern** (str) A naming pattern for the computed indexes columns. Must include "{name}" which will be replaced by the dimension name.

- **index_dtype** Numpy data type for the indexes. If it is a floating point type (default), then *indexes* will contain fractions, which may be used for interpolation. For out-of-range coordinates in *points*, indexes will be -1 if *index_dtype* is an integer type, and NaN, if *index_dtype* is a floating point types.
- **cube_asserted** (bool) If False, *cube* will be verified, otherwise it is expected to be a valid cube.

A dataset containing the index columns.

 $\begin{aligned} \texttt{xcube.core.extract.get_cube_values_for_indexes}(\textit{cube: Dataset, indexes: Dataset} \mid \textit{DataFrame} \mid \\ & Mapping[str, Any], \textit{include_coords: bool} = False, \\ & \textit{include_bounds: bool} = False, \textit{data_var_names:} \\ & Sequence[str] = None, \textit{index_name_pattern: str} = \\ & '{name}_\textit{index', method: str} = 'nearest', \textit{cube_asserted:} \\ & bool = False) \rightarrow \text{Dataset} \end{aligned}$

Get values from the *cube* at given *indexes*.

Return type

Dataset

Parameters

- **cube** (Dataset) A cube dataset.
- **indexes** A mapping from column names to index and fraction arrays for all cube dimensions.
- **include_coords** (bool) Whether to include the cube coordinates for each point in return value.
- **include_bounds** (bool) Whether to include the cube coordinate boundaries (if any) for each point in return value.
- **data_var_names** An optional list of names of data variables in *cube* whose values shall be extracted.
- **index_name_pattern** (str) A naming pattern for the computed indexes columns. Must include "{name}" which will be replaced by the dimension name.
- **method** (str) "nearest" or "linear".
- **cube_asserted** (bool) If False, *cube* will be verified, otherwise it is expected to be a valid cube.

Returns

A new data frame whose columns are values from *cube* variables at given *indexes*.

~typing.Sequence[int | float], index_dtype=<class 'numpy.float64'>) → DataArray | ndarray

Compute the indexes and their fractions into a coordinate variable *coord_var_name* of a *dataset* for the given coordinate values *coord_values*.

The coordinate variable's labels must be monotonic increasing or decreasing, otherwise the result will be nonsense. For any value in *coord_values* that is out of the bounds of the coordinate variable's values, the index depends on the value of *index_dtype*. If *index_dtype* is an integer type, invalid indexes are encoded as -1 while for floating point types, NaN will be used.

Returns a tuple of indexes as int64 array and fractions as float64 array.

Parameters

- dataset (Dataset) A cube dataset.
- **coord_var_name** (str) Name of a coordinate variable.
- coord_values Array-like coordinate values.
- **index_dtype** Numpy data type for the indexes. If it is floating point type (default), then *indexes* contain fractions, which may be used for interpolation. If *dtype* is an integer type out-of-range coordinates are indicated by index -1, and NaN if it is a floating point type.

Returns

The indexes and their fractions as a tuple of numpy int64 and float64 arrays.

 $\begin{aligned} \texttt{xcube.core.timeseries.get_time_series}(\textit{cube: Dataset, grid_mapping: GridMapping} \mid \textit{None} = \textit{None,} \\ & geometry: \textit{BaseGeometry} \mid \textit{Dict[str, Any]} \mid \textit{str} \mid \textit{Sequence[float} \mid \\ & \textit{int]} \mid \textit{None} = \textit{None, var_names: Sequence[str]} \mid \textit{None} = \textit{None,} \\ & start_date: datetime64 \mid \textit{str} \mid \textit{None} = \textit{None, end_date: datetime64} \mid \\ & str \mid \textit{None} = \textit{None, agg_methods: str} \mid \textit{Sequence[str]} \mid \\ & AbstractSet[str] = 'mean', use_groupby: bool = False, \\ & cube_asserted: bool \mid \textit{None} = \textit{None} \end{pmatrix} \rightarrow \textit{Dataset} \mid \textit{None} \end{aligned}$

Get a time series dataset from a data cube.

geometry may be provided as a (shapely) geometry object, a valid GeoJSON object, a valid WKT string, a sequence of box coordinates (x1, y1, x2, y2), or point coordinates (x, y). If *geometry* covers an area, i.e. is not a point, the function aggregates the variables to compute a mean value and if desired, the number of valid observations and the standard deviation.

start_date and end_date may be provided as a numpy.datetime64 or an ISO datetime string.

Returns a time-series dataset whose data variables have a time dimension but no longer have spatial dimensions, hence the resulting dataset's variables will only have N-2 dimensions. A global attribute max_number_of_observations will be set to the maximum number of observations that could have been made in each time step. If the given *geometry* does not overlap the cube's boundaries, or if not output variables remain, the function returns None.

- cube (Dataset) The xcube dataset
- grid_mapping Grid mapping of *cube*.
- geometry Optional geometry
- var_names Optional sequence of names of variables to be included.
- **start_date** Optional start date.
- end_date Optional end date.
- **agg_methods** Aggregation methods. May be single string or sequence of strings. Possible values are 'mean', 'median', 'min', 'max', 'std', 'count'. Defaults to 'mean'. Ignored if geometry is a point.
- **use_groupby** (bool) Use group-by operation. May increase or decrease runtime performance and/or memory consumption.

• cube_asserted – Deprecated and ignored since xcube 0.11.0. No replacement.

5.5 Cube Resampling

Resample dataset according to an affine transformation.

The affine transformation will be applied only if the CRS of *source_gm* and the CRS of *target_gm* are both geographic or equal. Otherwise, a ValueError will be raised.

Return type

Dataset

Parameters

- dataset (Dataset) The source dataset
- **source_gm** (*GridMapping*) Source grid mapping of *dataset*. Must be regular. Must have same CRS as *target_gm*.
- target_gm (GridMapping) Target grid mapping. Must be regular. Must have same CRS as source_gm.
- var_configs Optional resampling configurations for individual variables.
- **encode_cf** (bool) Whether to encode the target grid mapping into the resampled dataset in a CF-compliant way. Defaults to True.
- **gm_name** Name for the grid mapping variable. Defaults to "crs". Used only if *encode_cf* is True.
- **reuse_coords** (bool) Whether to either reuse target coordinate arrays from target_gm or to compute new ones.

Returns

The resampled target dataset.

xcube.core.resampling.encode_grid_mapping(ds: Dataset, gm: GridMapping, gm_name: str | None = None, force: bool | None = None) → Dataset

Encode the given grid mapping *gm* into a copy of *ds* in a CF-compliant way and return the dataset copy. The function removes any existing grid mappings.

If the CRS of *gm* is geographic and the spatial dimension and coordinate names are "lat", "lon" and *force* is False, or *force* is None and no former grid mapping was encoded in *ds*, then nothing else is done and the dataset copy is returned without further action.

Otherwise, for every spatial data variable with dims=(..., y, x), the function sets the attribute "grid_mapping" to *gm_name*. The grid mapping CRS is encoded in a new 0-D variable named *gm_name*.

Return type

Dataset

Parameters

- **ds** (Dataset) The dataset.
- gm (GridMapping) The dataset's grid mapping.
- gm_name Name for the grid mapping variable. Defaults to "crs".
- force Whether to force encoding of grid mapping even if CRS is geographic and spatial dimension names are "lon", "lat". Optional value, if not provided, *force* will be assumed True if a former grid mapping was encoded in *ds*.

Returns

A copy of ds with gm encoded into it.

xcube.core.resampling.rectify_dataset(source_ds: Dataset, *, var_names: str | Sequence[str] = None, source_gm: GridMapping = None, xy_var_names: Tuple[str, str] = None, target_gm: GridMapping = None, encode_cf: bool = True, gm_name: str | None = None, tile_size: int | Tuple[int, int] = None, is_j_axis_up: bool = None, output_ij_names: Tuple[str, str] = None, compute_subset: bool = True, uv_delta: float = 0.001) → Dataset | None

Reproject dataset *source_ds* using its per-pixel x,y coordinates or the given *source_gm*.

The function expects *source_ds* or the given *source_gm* to have either one- or two-dimensional coordinate variables that provide spatial x,y coordinates for every data variable with the same spatial dimensions.

For example, a dataset may comprise variables with spatial dimensions var(..., y_dim, x_dim), then one the function expects coordinates to be provided in two forms:

- 1. One-dimensional x_var(x_dim) and y_var(y_dim) (coordinate) variables.
- 2. Two-dimensional x_var(y_dim, x_dim) and y_var(y_dim, x_dim) (coordinate) variables.

If *target_gm* is given and it defines a tile size or *tile_size* is given, and the number of tiles is greater than one in the output's x- or y-direction, then the returned dataset will be composed of lazy, chunked dask arrays. Otherwise, the returned dataset will be composed of ordinary numpy arrays.

- source_ds (Dataset) Source dataset grid mapping.
- var_names Optional variable name or sequence of variable names.
- **source_gm** (*GridMapping*) Source dataset grid mapping.
- **xy_var_names** Optional tuple of the x- and y-coordinate variables in *source_ds*. Ignored if *source_gm* is given.
- **target_gm** (*GridMapping*) Optional target geometry. If not given, output geometry will be computed to spatially fit *dataset* and to retain its spatial resolution.
- **encode_cf** (bool) Whether to encode the target grid mapping into the resampled dataset in a CF-compliant way. Defaults to **True**.
- **gm_name** Name for the grid mapping variable. Defaults to "crs". Used only if *encode_cf* is True.
- **tile_size** Optional tile size for the output.

- **is_j_axis_up** (bool) Whether y coordinates are increasing with positive image j axis.
- **output_ij_names** If given, a tuple of variable names in which to store the computed source pixel coordinates in the returned output.
- **compute_subset** (bool) Whether to compute a spatial subset from *dataset* using *out_put_geom*. If set, the function may return None in case there is no overlap.
- **uv_delta** (float) A normalized value that is used to determine whether x,y coordinates in the output are contained in the triangles defined by the input x,y coordinates. The higher this value, the more inaccurate the rectification will be.

a reprojected dataset, or None if the requested output does not intersect with *dataset*.

xcube.core.resampling.resample_in_space(dataset: Dataset, source_gm: GridMapping = None, target_gm: GridMapping = None, var_configs: Mapping[Hashable, Mapping[str, Any]] = None, encode_cf: bool = True, gm_name: str | None = None)

Resample a dataset in the spatial dimensions.

If the source grid mapping *source_gm* is not given, it is derived from *dataset*: source_gm = GridMapping. from_dataset(dataset).

If the target grid mapping *target_gm* is not given, it is derived from *source_gm*: target_gm = source_gm. to_regular().

If *source_gm* is almost equal to *target_gm*, this function is a no-op and *dataset* is returned unchanged.

Otherwise the function computes a spatially resampled version of *dataset* and returns it.

Using *var_configs*, the resampling of individual variables can be configured. If given, *var_configs* must be a mapping from variable names to configuration dictionaries which can have the following properties:

• spline_order (int) - The order of spline polynomials

used for interpolating. It is used for upsampling only. Possible values are 0 to 5. Default is 1 (bi-linear) for floating point variables, and 0 (= nearest neighbor) for integer and bool variables.

• aggregator (str) - An optional aggregating

function. It is used for downsampling only. Examples are numpy.nanmean, numpy.nanmin, numpy.nanmax. Default is numpy.nanmean for floating point variables, and None (= nearest neighbor) for integer and bool variables.

• recover_nan (bool) - whether a special algorithm

shall be used that is able to recover values that would otherwise yield NaN during resampling. Default is True for floating point variables, and False for integer and bool variables.

Note that *var_configs* is only used if the resampling involves an affine transformation. This is true if the CRS of *source_gm* and CRS of *target_gm* are equal and one of two cases is given:

- 1. *source_gm* is regular. In this case the resampling is the affine transformation. and the result is returned directly.
- 2. *source_gm* is not regular and has a lower resolution than *target_cm*. In this case *dataset* is downsampled first using an affine transformation. Then the result is rectified.

In all other cases, no affine transformation is applied and the resampling is a direct rectification.

- **dataset** (Dataset) The source dataset.
- **source_gm** (*GridMapping*) The source grid mapping.

- target_gm (GridMapping) The target grid mapping. Must be regular.
- var_configs Optional resampling configurations for individual variables.
- **encode_cf** (bool) Whether to encode the target grid mapping into the resampled dataset in a CF-compliant way. Defaults to True.
- **gm_name** Name for the grid mapping variable. Defaults to "crs". Used only if *encode_cf* is True.

The spatially resampled dataset.

Resample a dataset in the time dimension.

The argument *method* may be one or a sequence of 'all', 'any', 'argmax', 'argmin', 'count', 'first', 'last', 'max', 'min', 'mean', 'median', 'percentile_', 'std', 'sum', 'var'.

In value 'percentile_' is a placeholder, where '' must be replaced by an integer percentage value, e.g. 'percentile_90' is the 90%-percentile.

Important note: As of xarray 0.14 and dask 2.8, the methods 'median' and 'percentile_'` cannot be used if the variables in *cube* comprise chunked dask arrays. In this case, use the ``compute() or load() method to convert dask arrays into numpy arrays.

Return type

Dataset

Parameters

- **dataset** (Dataset) The xcube dataset.
- **frequency** (str) Temporal aggregation frequency. Use format "<count><offset>" where <offset> is one of 'H', 'D', 'W', 'M', 'Q', 'Y'.
- method Resampling method or sequence of resampling methods.
- offset Offset used to adjust the resampled time labels. Uses same syntax as *frequency*.
- **base** Deprecated since xcube 1.0.4. No longer used as of pandas 2.0.
- time_chunk_size If not None, the chunk size to be used for the "time" dimension.
- **var_names** Variable names to include.
- **tolerance** Time tolerance for selective upsampling methods. Defaults to *frequency*.
- **interp_kind** Kind of interpolation if *method* is 'interpolation'.
- metadata Output metadata.
- **cube_asserted** (bool) If False, *cube* will be verified, otherwise it is expected to be a valid cube.

Returns

A new xcube dataset resampled in time.

5.6 Cube Manipulation

Convert data variables into a dimension.

Parameters

- **cube** (Dataset) The xcube dataset.
- **dim_name** (str) The name of the new dimension and coordinate variable. Defaults to 'var'.
- var_name (str) The name of the new, single data variable. Defaults to 'data'.
- **cube_asserted** (bool) If False, *cube* will be verified, otherwise it is expected to be a valid cube.

Returns

A new xcube dataset with data variables turned into a new dimension.

xcube.core.chunk.chunk_dataset(dataset: Dataset, chunk_sizes: Dict[str, int] = None, format_name: str = None) \rightarrow Dataset

Chunk dataset using chunk_sizes and optionally update encodings for given format_name.

Return type Dataset

Parameters

- dataset (Dataset) input dataset
- chunk_sizes mapping from dimension name to new chunk size
- format_name (str) optional format, e.g. "zarr" or "netcdf4"

Returns

the (re)chunked dataset

Unchunk dataset variables in-place.

Parameters

- **dataset_path** (str) Path to ZARR dataset directory.
- var_names Optional list of variable names.
- **coords_only** (bool) Un-chunk coordinate variables only.

exception_type: ~typing.Type[Exception] = <class 'ValueError'>)

Optimize a dataset for faster access.

Reduces the number of metadata and coordinate data files in xcube dataset given by given by *dataset_path*. Consolidated cubes open much faster from remote locations, e.g. in object storage, because obviously much less HTTP requests are required to fetch initial cube meta information. That is, it merges all metadata files into a single top-level JSON file ".zmetadata".

If *unchunk_coords* is given, it also removes any chunking of coordinate variables so they comprise a single binary data file instead of one file per data chunk. The primary usage of this function is to optimize data cubes for cloud object storage. The function currently works only for data cubes using Zarr format. *unchunk_coords* can be a name, or list of names of the coordinate variable(s) to be consolidated. If boolean True is used, coordinate all variables will be consolidated.

Parameters

- **input_path** (str) Path to input dataset with ZARR format.
- **output_path** (str) Path to output dataset with ZARR format. May contain "{input}" template string, which is replaced by the input path's file name without file name extension.
- **in_place** (bool) Whether to modify the dataset in place. If False, a copy is made and *output_path* must be given.
- **unchunk_coords** The name of a coordinate variable or a list of coordinate variables whose chunks should be consolidated. Pass **True** to consolidate chunks of all coordinate variables.
- exception_type Type of exception to be used on value errors.

5.7 Cube Subsetting

xcube.core.select.select_variables_subset($dataset: Dataset, var_names: Collection[str] | None = None$) $\rightarrow Dataset$

Select data variable from given dataset and create new dataset.

Return type Dataset

Parameters

- dataset (Dataset) The dataset from which to select variables.
- var_names The names of data variables to select.

Returns

A new dataset. It is empty, if var_names is empty. It is dataset, if var_names is None.

xcube.core.geom.clip_dataset_by_geometry(dataset: Dataset, geometry: BaseGeometry | Dict[str, Any] | str | Sequence[float | int], save_geometry_wkt: str | bool = False) → Dataset | None

Spatially clip a dataset according to the bounding box of a given geometry.

Parameters

- dataset (Dataset) The dataset
- **geometry** A geometry-like object, see py:function:*convert_geometry*.
- **save_geometry_wkt** If the value is a string, the effective intersection geometry is stored as a Geometry WKT string in the global attribute named by *save_geometry*. If the value is True, the name "geometry_wkt" is used.

Returns

The dataset spatial subset, or None if the bounding box of the dataset has a no or a zero area intersection with the bounding box of the geometry.

5.8 Cube Masking

 $\begin{aligned} \texttt{xcube.core.geom.mask_dataset_by_geometry}(\textit{dataset: Dataset, geometry: BaseGeometry} \mid \textit{Dict[str, Any]} \mid \textit{str} \\ \mid \textit{Sequence[float | int], tile_size: int | Tuple[int, int] = None,} \\ excluded_vars: Sequence[str] = None, no_clip: bool = False, \\ all_touched: bool = False, save_geometry_mask: str | bool = False, \\ save_geometry_wkt: str | bool = False) \rightarrow Dataset | \\ None \end{aligned}$

Mask a dataset according to the given geometry. The cells of variables of the returned dataset will have NaNvalues where their spatial coordinates are not intersecting the given geometry.

Parameters

- dataset (Dataset) The dataset
- geometry A geometry-like object, see py:function:convert_geometry.
- **tile_size** If given, the unconditional spatial chunk sizes in x- and y-direction in pixels. May be given as integer scalar or x,y-pair of integers.
- **excluded_vars** Optional sequence of names of data variables that should not be masked (but still may be clipped).
- **no_clip** (bool) If True, the function will not clip the dataset before masking, this is, the returned dataset will have the same dimension size as the given *dataset*.
- **all_touched** (bool) If True, all pixels intersected by geometry outlines will be included in the mask. If False, only pixels whose center is within the polygon or that are selected by Bresenham's line algorithm will be included in the mask. The default value is set to *False*.
- **save_geometry_mask** If the value is a string, the effective geometry mask array is stored as a 2D data variable named by *save_geometry_mask*. If the value is True, the name "geometry_mask" is used.
- **save_geometry_wkt** If the value is a string, the effective intersection geometry is stored as a Geometry WKT string in the global attribute named by *save_geometry*. If the value is True, the name "geometry_wkt" is used.

Returns

The dataset spatial subset, or None if the bounding box of the dataset has a no or a zero area intersection with the bounding box of the geometry.

class xcube.core.maskset.MaskSet(flag_var: DataArray)

A set of mask variables derived from a variable *flag_var* with the following CF attributes:

- One or both of *flag_masks* and *flag_values*
- flag_meanings (always required)

See https://cfconventions.org/Data/cf-conventions/cf-conventions-1.9/cf-conventions.html#flags for details on the use of these attributes.

Each mask is represented by an *xarray.DataArray*, has the name of the flag, is of type *numpy.unit8*, and has the dimensions of the given *flag_var*.

Parameters

flag_var – an *xarray.DataArray* that defines flag values. The CF attributes *flag_meanings* and one or both of *flag_masks* and *flag_values* are expected to exist and be valid.

classmethod get_mask_sets(*dataset: Dataset*) → Dict[str, *MaskSet*]

For each "flag" variable in given *dataset*, turn it into a MaskSet, store it in a dictionary.

Parameters dataset (Dataset) – The dataset

Returns

A mapping of flag names to MaskSet. Will be empty if there are no flag variables in *dataset*.

5.9 Rasterisation of Features

 $\begin{aligned} \texttt{xcube.core.geom.rasterize_features}(\textit{dataset: Dataset, features: pandas.geodataframe.GeoDataFrame} \mid \\ Sequence[Mapping[str, Any]], feature_props: Sequence[str], \\ var_props: Dict[str, Mapping[str, Mapping[str, Any]]] = None, \\ tile_size: int | Tuple[int, int] = None, all_touched: bool = False, \\ in_place: bool = False) \rightarrow Dataset | None \end{aligned}$

Rasterize feature properties given by *feature_props* of vector-data *features* as new variables into *dataset*.

dataset must have two spatial 1-D coordinates, either lon and lat in degrees, reprojected coordinates, x and y, or similar.

feature_props is a sequence of names of feature properties that must exists in each feature of features.

features may be passed as pandas.GeoDataFrame`` or as an iterable of GeoJSON features.

Using the optional *var_props*, the properties of newly created variables from feature properties can be specified. It is a mapping of feature property names to mappings of variable properties. Here is an example variable properties mapping::

'name': 'land_class', # (str) - the variable's name, # default is the feature property name;

'dtype' np.int16, # (str|np.dtype) - the variable's dtype, # default is np.float64;

'fill_value': -999, # (bool|int|float|np.nparray) -

the variable's fill value, # default is np.nan;

'attrs': {}, # (Mapping[str, Any]) # the variable's fill value, default is { };

'converter': int, # (Callable[[Any], Any]) -

a converter function used to convert # from property feature value to variable # value, default is float. # Deprecated, no longer used.

}

{

Note that newly created variables will have data type *np.float64* because *np.nan* is used to encode missing values. *fill_value* and *dtype* are used to encode the variables when persisting the data.

Currently, the coordinates of the geometries in the given *features* must use the same CRS as the given *dataset*.

- dataset (Dataset) The xarray dataset.
- features A geopandas.GeoDataFrame instance or a sequence of GeoJSON features.

- feature_props Sequence of names of numeric feature properties to be rasterized.
- **var_props** Optional mapping of feature property name to a name or a 5-tuple (name, dtype, fill_value, attributes, converter) for the new variable.
- **tile_size** If given, the unconditional spatial chunk sizes in x- and y-direction in pixels. May be given as integer scalar or x,y-pair of integers.
- **all_touched** (bool) If True, all pixels intersected by a feature's geometry outlines will be included. If False, only pixels whose center is within the feature polygon or that are selected by Bresenham's line algorithm will be included in the mask. The default is False.
- **in_place** (bool) Whether to add new variables to *dataset*. If False, a copy will be created and returned.

dataset with rasterized feature_property

5.10 Cube Metadata

```
xcube.core.update.update_dataset_attrs(dataset: Dataset, global_attrs: Dict[str, Any] = None,
```

update_existing: bool = False, in_place: bool = False) \rightarrow

Dataset

Update spatio-temporal CF/THREDDS attributes given *dataset* according to spatio-temporal coordinate variables time, lat, and lon.

Return type

Dataset

Parameters

- dataset (Dataset) The dataset.
- global_attrs Optional global attributes.
- update_existing (bool) If True, any existing attributes will be updated.
- **in_place** (bool) If True, *dataset* will be modified in place and returned.

Returns

A new dataset, if *in_place* if False (default), else the passed and modified *dataset*.

```
xcube.core.update_update_dataset_spatial_attrs(dataset: Dataset, update_existing: bool = False, in place: bool = False) \rightarrow Dataset
```

Update spatial CF/THREDDS attributes of given dataset.

Return type

Dataset

Parameters

- **dataset** (Dataset) The dataset.
- update_existing (bool) If True, any existing attributes will be updated.
- **in_place** (bool) If True, *dataset* will be modified in place and returned.

Returns

A new dataset, if *in_place* if False (default), else the passed and modified *dataset*.

```
xcube.core.update.update_dataset_temporal_attrs(dataset: Dataset, update_existing: bool = False, in_place: bool = False) \rightarrow Dataset
```

Update temporal CF/THREDDS attributes of given dataset.

Return type

Dataset

Parameters

- dataset (Dataset) The dataset.
- update_existing (bool) If True, any existing attributes will be updated.
- **in_place** (bool) If True, *dataset* will be modified in place and returned.

Returns

A new dataset, if *in_place* is False (default), else the passed and modified *dataset*.

5.11 Cube verification

xcube.core.verify.**assert_cube**(*dataset: Dataset, name=None*) → Dataset

Assert that the given *dataset* is a valid xcube dataset.

Return type

Dataset

Parameters

- **dataset** (Dataset) The dataset to be validated.
- **name** Optional parameter name.

Raise

ValueError, if dataset is not a valid xcube dataset

xcube.core.verify.verify_cube(dataset: Dataset) \rightarrow List[str]

Verify the given *dataset* for being a valid xcube dataset.

The tool verifies that *dataset* * defines two spatial x,y coordinate variables, that are 1D, non-empty, using correct units; * defines a time coordinate variables, that are 1D, non-empty, using correct units; * has valid bounds variables for spatial x,y and time coordinate variables, if any; * has any data variables and that they are valid, e.g. min. 3-D, all have

same dimensions, have at least the dimensions dim(time), dim(y), dim(x) in that order.

Returns a list of issues, which is empty if *dataset* is a valid xcube dataset.

Parameters

dataset (Dataset) – A dataset to be verified.

Returns

List of issues or empty list.

5.12 Multi-Resolution Datasets

class xcube.core.mldataset.MultiLevelDataset

A multi-level dataset of decreasing spatial resolutions (a multi-resolution pyramid).

The pyramid level at index zero provides the original spatial dimensions. The size of the spatial dimensions in subsequent levels is computed by the formula size[index + 1] = (size[index] + 1) // 2 with size[index] being the maximum size of the spatial dimensions at level zero.

Any dataset chunks are assumed to be the same in all levels. Usually, the number of chunks is one in one of the spatial dimensions of the highest level.

abstract property ds_id: str

Returns

the dataset identifier.

abstract property grid_mapping: GridMapping

Returns

the CF-conformal grid mapping

abstract property num_levels: int

Returns

the number of pyramid levels.

property resolutions: Sequence[Tuple[float, float]]

Returns

the x,y resolutions for each level given in the spatial units of the dataset's CRS (i.e. self. grid_mapping.crs).

property avg_resolutions: Sequence[float]

Returns

the average x,y resolutions for each level given in the spatial units of the dataset's CRS (i.e. self.grid_mapping.crs).

property base_dataset: Dataset

Returns

the base dataset for lowest level at index 0.

property datasets: Sequence[Dataset]

Get datasets for all levels.

Calling this method will trigger any lazy dataset instantiation.

Returns

the datasets for all levels.

<code>abstract get_dataset(index: int) \rightarrow Dataset</code>

Return type Dataset

Parameters

index (int) – the level index

Returns

the dataset for the level at *index*.

close()

Close all datasets. Default implementation does nothing.

apply(function: Callable[[Dataset, Dict[str, Any]], Dataset], kwargs: Dict[str, Any] = None, ds_id: str = None) \rightarrow MultiLevelDataset

Apply function to all level datasets and return a new multi-level dataset.

derive_tiling_scheme(tiling_scheme: TilingScheme)

Derive a new tiling scheme for the given one with defined minimum and maximum level indices.

get_level_for_resolution(xy_res: float | Tuple[float, float]) \rightarrow int

Get the index of the level that best represents the given resolution.

Return type int

Parameters

xy_res – the resolution in x- and y-direction

Returns

a level ranging from 0 to self.num_levels - 1

class xcube.core.mldataset.BaseMultiLevelDataset(base_dataset: Dataset, grid_mapping: GridMapping

| None = None, num_levels: int | None = None, agg_methods: None | str | Mapping[str, None | str] = 'first', ds_id: str | None = None)

A multi-level dataset whose level datasets are created by down-sampling a base dataset.

Parameters

- **base_dataset** The base dataset for the level at index zero.
- grid_mapping Optional grid mapping for base_dataset.
- num_levels Optional number of levels.
- ds_id Optional dataset identifier.
- **agg_methods** Optional aggregation methods. May be given as string or as mapping from variable name pattern to aggregation method. Valid aggregation methods are None, "first", "min", "max", "mean", "median". If None, the default, "first" is used for integer variables and "mean" for floating point variables.

Dict[str, Any] | None = None)

A multi-level dataset that is a combination of other multi-level datasets.

- ml_datasets The multi-level datasets to be combined. At least two must be provided.
- ds_id Optional dataset identifier.
- **combiner_function** An optional function used to combine the datasets, for example xarray.merge. If given, it receives a list of datasets (xarray.Dataset instances) and *combiner_params* as keyword arguments. If not given or None is passed, a copy of the first dataset is made, which is then subsequently updated by the remaining datasets using xarray.Dataset.update().
- **combiner_params** Parameters to the *combiner_function* passed as keyword arguments.

close()

Close all datasets. Default implementation does nothing.

Size weights are used to distribute the cache size over the levels.

The identity.

A multi-level dataset where each level dataset is lazily retrieved, i.e. read or computed by the abstract method get_dataset_lazily(index, **kwargs).

Parameters

- ds_id Optional dataset identifier.
- **parameters** Optional keyword arguments that will be passed to the get_dataset_lazily method.

property ds_id: str

Returns

the dataset identifier.

property grid_mapping: GridMapping

Returns

the CF-conformal grid mapping

property num_levels: int

Returns

the number of pyramid levels.

property lock: RLock

Get the reentrant lock used by this object to synchronize lazy instantiation of properties.

get_dataset(*index: int*) \rightarrow Dataset

Get or compute the dataset for the level at given *index*.

Return type Dataset

Parameters index (int) – the level index

Returns

the dataset for the level at *index*.

set_dataset(index: int, level_dataset: Dataset)

Set the dataset for the level at given *index*.

Callers need to ensure that the given *level_dataset* has the correct spatial dimension sizes for the given level at *index*.

Parameters

- **index** (int) the level index
- **level_dataset** (Dataset) the dataset for the level at *index*.

close()

Close all datasets. Default implementation does nothing.

close()

Close all datasets. Default implementation does nothing.

5.13 Zarr Store

class xcube.core.zarrstore.ZarrStoreHolder(dataset: Dataset)

Represents a xarray dataset property zarr_store.

It is used to permanently associate a dataset with its Zarr store, which would otherwise not be possible.

In xcube server, we use the new property to expose datasets via the S3 emulation API.

For that concept to work, datasets must be associated with their Zarr stores explicitly. Therefore, the xcube data store framework sets the Zarr stores of datasets after opening them xr.open_zarr():

```
`python dataset = xr.open_zarr(zarr_store, **open_params) dataset.zarr_store.
set(zarr_store) `
```

Note, that the dataset may change after the Zarr store has been set, so that the dataset and its Zarr store are no longer in sync. This may be an issue and limit the application of the new property.

Parameters

dataset – The xarray dataset that is associated with a Zarr store.

get() \rightarrow MutableMapping

Get the Zarr store of a dataset. If no Zarr store has been set, the method will use GenericZarrStore. from_dataset() to create and set one.

The Zarr store.

set(*zarr_store: MutableMapping*) \rightarrow None

Set the Zarr store of a dataset. :type zarr_store: MutableMapping :param zarr_store: The Zarr store.

 $reset() \rightarrow None$

Resets the Zarr store.

```
class xcube.core.zarrstore.GenericZarrStore(*arrays: GenericArray | Dict[str, Any], attrs: Dict[str,
Any] | None = None, array_defaults: GenericArray |
Dict[str, Any] | None = None)
```

A Zarr store that maintains generic arrays in a flat, top-level hierarchy. The root of the store is a Zarr group conforming to the Zarr spec v2.

It is designed to serve as a Zarr store for xarray datasets that compute their data arrays dynamically.

See class GenericArray for specifying the arrays' properties.

The array data of this store's arrays are either retrieved from static (numpy) arrays or from a callable that provides the array's data chunks as bytes or numpy arrays.

Parameters

- arrays Arrays to be added. Typically, these will be instances of GenericArray.
- attrs Optional attributes of the top-level group. If given, it must be JSON serializable.
- **array_defaults** Optional array defaults for array properties not passed to add_array. Typically, this will be an instance of GenericArray.

Array

alias of GenericArray

add_array(array: GenericArray | Dict[str, Any] | $None = None, **array_kwargs) \rightarrow None$

Add a new array to this store.

Parameters

- **array** Optional array properties. Typically, this will be an instance of GenericArray.
- array_kwargs Keyword arguments form for the properties of GenericArray.

$\texttt{is_writeable()} \rightarrow \texttt{bool}$

Return False, because arrays in this store are generative.

listdir(*path:* str = ") \rightarrow List[str]

List a store path. :type path: str :param path: The path. :return: List of sorted directory entries.

rmdir(*path: str* = '') \rightarrow None

The general form removes store paths. This implementation can remove entire arrays only. :type path: str :param path: The array's name.

rename(*src_path: str*, *dst_path: str*) \rightarrow None

The general form renames store paths. This implementation can rename arrays only.

- **src_path** (str) Source array name.
- **dst_path** (str) Target array name.

$close() \rightarrow None$

Calls the "on_close" handlers, if any, of arrays.

classmethod from_dataset(*dataset: Dataset, array_defaults:* GenericArray | *Dict[str, Any]* | *None* = None) \rightarrow *GenericZarrStore*

Create a Zarr store for given *dataset*. to the *dataset*'s attributes. The following *array_defaults* properties can be provided (other properties are prescribed by the *dataset*):

- fill_value- defaults to None
- compressor- defaults to None
- filters- defaults to None
- order- defaults to "C"
- chunk_encoding defaults to "bytes"

Parameters

- **dataset** (Dataset) The dataset
- array_defaults Array default values.

Returns

A new Zarr store instance.

class xcube.core.zarrstore.GenericArray(array: Dict[str, any] | None = None, name: str | None = None, get_data: Callable[[Tuple[int]], bytes | ndarray] | None = None, get_data_params: Dict[str, Any] | None = None, data: ndarray | None = None, dtype: str | dtype | None = None, dims: str | Sequence[str] | None = None, shape: Sequence[int] | None = None, chunks: Sequence[int] | None = None, fill_value: bool | int | float | str | None = None, compressor: Codec | None = None, filters: Sequence[Codec] | None = None, order: str | None = None, attrs: Dict[str, Any] | None = None, on_close: Callable[[Dict[str, Any]], None] | None = None, chunk_encoding: str | None = None, **kwargs)

Represent a generic array in the GenericZarrStore as dictionary of properties.

Although all properties of this class are optional, some of them are mandatory when added to the GenericZarrStore.

When added to the store using GenericZarrStore.add_array(), the array *name* and *dims* must always be present. Other mandatory properties depend on the *data* and *get_data* properties, which are mutually exclusive:

- get_data is called for a requested data chunk of an array. It must return a bytes object or a numpy ndarray and is passed the chunk index, the chunk shape, and this array info dictionary. get_data requires the following properties to be present too: name, dims, dtype, shape. chunks is optional and defaults to shape.
- *data* must be a bytes object or a numpy nd-array. *data* requires the following properties to be present too: *name*, *dims*. *chunks* must be same as *shape*.

The function *get_data* receives only keyword-arguments which comprises the ones passed by *get_data_params*, if any, and two special ones which may occur in the signature of *get_data*:

• The keyword argument *chunk_info*, if given, provides a dictionary that holds information about the current chunk: - index: tuple[int, ...] - the chunk's index - shape: tuple[int, ...] - the chunk's shape - slices: tuple[slice, ...] - the chunk's array slices

• The keyword argument *array_info*, if given, provides a dictionary that holds information about the overall array. It contains all array properties passed to the constructor of GenericArray plus - ndim: int - number of dimensions - num_chunks: tuple[int, ...] - number of chunks in every dimension

GenericZarrStore will convert a Numpy array returned by *get_data* or given by *data* into a bytes object. It will also be compressed, if a *compressor* is given. It is important that the array chunks always See also https://zarr.readthedocs.io/en/stable/spec/v2.html#chunks

Note that if the value of a named keyword argument is None, it will not be stored.

Parameters

- array Optional array info dictionary
- **name** Optional array name
- **data** Optional array data. Mutually exclusive with *get_data*. Must be a bytes object or a numpy array.
- **get_data** Optional array data chunk getter. Mutually exclusive with *data*. Called for a requested data chunk of an array. Must return a bytes object or a numpy array.
- get_data_params Optional keyword-arguments passed to get_data.
- **dtype** Optional array data type. Either a string using syntax of the Zarr spec or a numpy. dtype. For string encoded data types, see https://zarr.readthedocs.io/en/stable/spec/v2. html#data-type-encoding
- **dims** Optional sequence of dimension names.
- **shape** Optional sequence of shape sizes for each dimension.
- chunks Optional sequence of chunk sizes for each dimension.
- **fill_value** Optional fill value, see https://zarr.readthedocs.io/en/stable/spec/v2.html# fill-value-encoding
- **compressor** Optional compressor. If given, it must be an instance of numcodecs.abc. Codec.
- **filters** Optional sequence of filters, see https://zarr.readthedocs.io/en/stable/spec/v2. html#filters.
- order Optional array endian ordering. If given, must be "C" or "F". Defaults to "C".
- **attrs** Optional array attributes. If given, must be JSON-serializable.
- on_close Optional array close handler. Called if the store is closed.
- **chunk_encoding** Optional encoding type of the chunk data returned for the array. Can be "bytes" (the default) or "ndarray" for array chunks that are numpy.ndarray instances.
- kwargs Other keyword arguments passed directly to the dictionary constructor.

finalize() \rightarrow *GenericArray*

Normalize and validate array properties and return a valid array info dictionary to be stored in the *Gener*-*icZarrStore*.

class xcube.core.zarrstore.CachedZarrStore(store: MutableMapping, cache: MutableMapping)

A read-only Zarr store that is faster than *store* because it uses a writable *cache* store.

The cache store is assumed to read values for a given key much faster than store.

Note that iterating keys and containment checks are performed on store only.

- **store** A Zarr store that is known to be slow in reading values.
- cache A writable Zarr store that can read values faster than store.

class xcube.core.zarrstore.DiagnosticZarrStore(store: MutableMapping)

A diagnostic Zarr store used for testing and investigating behaviour of Zarr and xarray's Zarr backend.

Parameters

store – Wrapped Zarr store.

keys() \rightarrow a set-like object providing a view on D's keys

5.14 Utilities

class xcube.core.gridmapping.GridMapping(size: int | Tuple[int, int], tile_size: int | Tuple[int, int] | None,

xy_bbox: Tuple[int | float, int | float, int | float, int | float], xy_res: int | float | Tuple[int | float, int | float], crs: CRS, xy_var_names: Tuple[str, str], xy_dim_names: Tuple[str, str], is_regular: bool | None, is_lon_360: bool | None, is_j_axis_up: bool | None, x_coords: DataArray | None = None, y_coords: DataArray | None = None, xy_coords: DataArray | None = None)

An abstract base class for grid mappings that define an image grid and a transformation from image pixel coordinates to spatial Earth coordinates defined in a well-known coordinate reference system (CRS).

This class cannot be instantiated directly. Use one of its factory methods to create instances:

- :meth:regular()
- :meth:from_dataset()
- :meth:from_coords()

Some instance methods can be used to derive new instances:

- :meth:derive()
- :meth:scale()
- :meth:transform()
- :meth:to_regular()

This class is thread-safe.

derive(xy_var_names: Tuple[str, str] = None, xy_dim_names: Tuple[str, str] = None, tile_size: int |
 Tuple[int, int] = None, is_j_axis_up: bool = None)

Derive a new grid mapping from this one with some properties changed.

Parameters

- **xy_var_names** The new x-, and y-variable names.
- **xy_dim_names** The new x-, and y-dimension names.
- tile_size The new tile size
- **is_j_axis_up** (bool) Whether j-axis points up.

Returns

A new, derived grid mapping.

scale(xy_scale: int | float | Tuple[int | float, int | float], tile_size: int | Tuple[int, int] = None) \rightarrow GridMapping

Derive a scaled version of this regular grid mapping.

Scaling factors lower than one correspond to up-scaling (pixels sizes decrease, image size increases).

Scaling factors larger than one correspond to down-scaling. (pixels sizes increase, image size decreases).

Parameters

• **xy_scale** – The x-, and y-scaling factors. May be a single number or tuple.

• tile_size – The new tile size

```
Returns
```

A new, scaled grid mapping.

property size: Tuple[int, int]

Image size (width, height) in pixels.

property width: int

Image width in pixels.

property height: int

Image height in pixels.

property tile_size: Tuple[int, int]

Image tile size (width, height) in pixels.

property is_tiled: bool

Whether the image is tiled.

property tile_width: int

Image tile width in pixels.

property tile_height: int

Image tile height in pixels.

property x_coords

The 1D or 2D x-coordinate array of shape (width,) or (height, width).

property y_coords

The 1D or 2D y-coordinate array of shape (width,) or (height, width).

property xy_coords: DataArray

The x,y coordinates as data array of shape (2, height, width). Coordinates are given in units of the CRS.

property xy_coords_chunks: Tuple[int, int, int]

Get the chunks for the *xy_coords* array.

property xy_var_names: Tuple[str, str]

The variable names of the x,y coordinates as tuple (x_var_name, y_var_name).

property xy_dim_names: Tuple[str, str]

The dimension names of the x,y coordinates as tuple (x_dim_name, y_dim_name).

property xy_bbox: Tuple[float, float, float, float]

The image's bounding box in CRS coordinates.

- property x_min: int | float Minimum x-coordinate in CRS units.
- property y_min: int | float Minimum y-coordinate in CRS units.
- property x_max: int | float

Maximum x-coordinate in CRS units.

property y_max: int | float

Maximum y-coordinate in CRS units.

property xy_res: Tuple[int | float, int | float]

Pixel size in x and y direction.

property x_res: int | float

Pixel size in CRS units per pixel in x-direction.

property y_res: int | float

Pixel size in CRS units per pixel in y-direction.

property crs: CRS

The coordinate reference system.

property is_lon_360: bool | None

Check whether x_max is greater than 180 degrees. Effectively tests whether the range x_min , x_max crosses the anti-meridian at 180 degrees. Works only for geographical coordinate reference systems.

property is_regular: bool | None

Do the x,y coordinates for a regular grid? A regular grid has a constant delta in both x- and y-directions of the x- and y-coordinates. :return None, if this property cannot be determined,

True or False otherwise.

property is_j_axis_up: bool | None

Does the positive image j-axis point up? By default, the positive image j-axis points down.

:return None, if this property cannot be determined,

True or False otherwise.

property ij_to_xy_transform: Tuple[Tuple[int | float, int | float, int | float], Tuple[int | float, int | float, int | float]]

The affine transformation matrix from image to CRS coordinates. Defined only for grid mappings with rectified x,y coordinates.

property xy_to_ij_transform: Tuple[Tuple[int | float, int | float, int | float], Tuple[int | float, int | float, int | float]]

The affine transformation matrix from CRS to image coordinates. Defined only for grid mappings with rectified x,y coordinates.

 $ij_transform_to(other: GridMapping) \rightarrow Tuple[Tuple[int | float, int | float, int | float], Tuple[int | float, int | float]]$

Get the affine transformation matrix that transforms image coordinates of *other* into image coordinates of this image geometry.

Defined only for grid mappings with rectified x,y coordinates.

Parameters

other – The other image geometry

Affine transformation matrix

ij_transform_from(*other*: GridMapping) → Tuple[Tuple[int | float, int | float, int | float], Tuple[int | float, int | float]]

Get the affine transformation matrix that transforms image coordinates of this image geometry to image coordinates of *other*.

Defined only for grid mappings with rectified x,y coordinates.

Parameters other – The other image geometry

Returns

Affine transformation matrix

property ij_bbox: Tuple[int, int, int]

The image's bounding box in pixel coordinates.

property ij_bboxes: ndarray

The image tiles' bounding boxes in image pixel coordinates.

property xy_bboxes: ndarray

The image tiles' bounding boxes in CRS coordinates.

Compute bounding box in i,j pixel coordinates given a bounding box *xy_bbox* in x,y coordinates.

Parameters

- **xy_bbox** Box (x_min, y_min, x_max, y_max) given in the same CS as x and y.
- **xy_border** (float) If non-zero, grows the bounding box *xy_bbox* before using it for comparisons. Defaults to 0.
- **ij_border** (int) If non-zero, grows the returned i,j bounding box and clips it to size. Defaults to 0.

Returns

Bounding box in (i_min, j_min, i_max, j_max) in pixel coordinates. Returns (-1, -1, -1, -1) if *xy_bbox* isn't intersecting any of the x,y coordinates.

ij_bboxes_from_xy_bboxes(xy_bboxes : ndarray, xy_border : float = 0.0, ij_border : int = 0, ij_bboxes : ndarray = None) \rightarrow ndarray

Compute bounding boxes in pixel coordinates given bounding boxes *xy_bboxes* [[x_min, y_min, x_max, y_max], ...] in x,y coordinates.

The returned array in i,j pixel coordinates has the same shape as *xy_bboxes*. The value ranges in the returned array [[i_min, j_min, i_max, j_max], ..]] are:

- i_min from 0 to width-1, i_max from 1 to width;
- j_min from 0 to height-1, j_max from 1 to height;

so the i,j pixel coordinates can be used as array index slices.

Return type ndarray

- **xy_bboxes** (ndarray) Numpy array of x,y bounding boxes [[x_min, y_min, x_max, y_max], ...] given in the same CS as x and y.
- **xy_border** (float) If non-zero, grows the bounding box *xy_bbox* before using it for comparisons. Defaults to 0.
- **ij_border** (int) If non-zero, grows the returned i,j bounding box and clips it to size. Defaults to 0.
- **ij_bboxes** (ndarray) Numpy array of pixel i,j bounding boxes [[x_min, y_min, x_max, y_max], ...]. If given, must have same shape as *xy_bboxes*.

Bounding boxes in [[i_min, j_min, i_max, j_max], ..]] in pixel coordinates.

to_dataset_attrs() \rightarrow Mapping[str, Any]

Get spatial dataset attributes as recommended by https://wiki.esipfed.org/Attribute_Convention_for_Data_ Discovery_1-3#Recommended

Returns

dictionary with dataset coordinate attributes.

to_coords (xy_var_names : $Tuple[str, str] = None, xy_dim_names$: $Tuple[str, str] = None, exclude_bounds$: $bool = False, reuse_coords$: $bool = False) \rightarrow Mapping[str, DataArray]$

Get CF-compliant axis coordinate variables and cell boundary coordinate variables.

Defined only for grid mappings with regular x,y coordinates.

Parameters

- xy_var_names Optional coordinate variable names (x_var_name, y_var_name).
- xy_dim_names Optional coordinate dimensions names (x_dim_name, y_dim_name).
- exclude_bounds (bool) If True, do not create bounds coordinates. Defaults to False.
- **reuse_coords** (bool) Whether to either reuse target coordinate arrays from target_gm or to compute new ones.

Returns

dictionary with coordinate variables

transform(*crs: str* | *CRS*, *, *tile_size: int* | *Tuple*[*int, int*] = *None*, *xy_var_names: Tuple*[*str, str*] = *None*, *tolerance: float* = 1e-05) \rightarrow *GridMapping*

Transform this grid mapping so it uses the given spatial coordinate reference system into another crs.

Parameters

- **crs** The new spatial coordinate reference system.
- **tile_size** Optional new tile size.
- xy_var_names Optional new coordinate names.
- **tolerance** (float) Absolute tolerance used when comparing coordinates with each other. Must be in the units of the *crs* and must be greater zero.

Returns

A new grid mapping that uses crs.

classmethod regular(*size: int* | *Tuple*[*int, int*], *xy_min: Tuple*[*float, float*], *xy_res: float* | *Tuple*[*float, float*], *crs: str* | *CRS*, *, *tile_size: int* | *Tuple*[*int, int*] = *None, is_j_axis_up: bool* = *False*) \rightarrow *GridMapping*

Create a new regular grid mapping.

Parameters

- **size** Size in pixels.
- xy_min Minimum x- and y-coordinates.
- xy_res Resolution in x- and y-directions.
- **crs** Spatial coordinate reference system.
- tile_size Optional tile size.
- **is_j_axis_up** (bool) Whether positive j-axis points up. Defaults to false.

Returns

A new regular grid mapping.

to_regular(*tile_size: int* | *Tuple[int, int]* = *None, is_j_axis_up: bool* = *False*) \rightarrow *GridMapping* Transform this grid mapping into one that is regular.

Parameters

- **tile_size** Optional tile size.
- **is_j_axis_up** (bool) Whether positive j-axis points up. Defaults to false.

Returns

A new regular grid mapping or this grid mapping, if it is already regular.

classmethod from_dataset(dataset: Dataset, *, crs: str | CRS = None, tile_size: int | Tuple[int, int] = None, prefer_is_regular: bool = True, prefer_crs: str | CRS = None, emit_warnings: bool = False, tolerance: float = 1e-05) \rightarrow GridMapping

Create a grid mapping for the given *dataset*.

Parameters

- **dataset** (Dataset) The dataset.
- crs Optional spatial coordinate reference system.
- tile_size Optional tile size
- **prefer_is_regular** (bool) Whether to prefer a regular grid mapping if multiple found. Default is True.
- prefer_crs The preferred CRS of a grid mapping if multiple found.
- emit_warnings (bool) Whether to emit warning for non-CF compliant datasets.
- **tolerance** (float) Absolute tolerance used when comparing coordinates with each other. Must be in the units of the *crs* and must be greater zero.

Returns

a new grid mapping instance.

classmethod from_coords(x_coords : DataArray, y_coords : DataArray, crs: str | CRS, *, tile_size: int | Tuple[int, int] = None, tolerance: float = 1e-05) \rightarrow GridMapping

Create a grid mapping from given x- and y-coordinates *x_coords*, *y_coords* and spatial coordinate reference system *crs*.

- **x_coords** (DataArray) The x-coordinates.
- y_coords (DataArray) The y-coordinates.
- crs The spatial coordinate reference system.

- **tile_size** Optional tile size.
- **tolerance** (float) Absolute tolerance used when comparing coordinates with each other. Must be in the units of the *crs* and must be greater zero.

A new grid mapping.

is_close(*other*: GridMapping, *tolerance*: *float* = *1e*-05) → bool

Tests whether this grid mapping is close to *other*.

Return type bool

Parameters

- **other** The other grid mapping.
- **tolerance** (float) Absolute tolerance used when comparing coordinates with each other. Must be in the units of the *crs* and must be greater zero.

Returns

True, if so, False otherwise.

xcube.core.geom.convert_geometry(geometry: BaseGeometry | Dict[str, Any] | str | Sequence[float | int] | None) \rightarrow BaseGeometry | None

Convert a geometry-like object into a shapely geometry object (shapely.geometry.BaseGeometry).

A geometry-like object may be any shapely geometry object, * a dictionary that can be serialized to valid GeoJ-SON, * a WKT string, * a box given by a string of the form "<x1>,<y1>,<x2>,<y2>"

or by a sequence of four numbers x1, y1, x2, y2,

• a point by a string of the form "<x>,<y>" or by a sequence of two numbers x, y.

Handling of geometries crossing the anti-meridian:

- If box coordinates are given, it is allowed to pass x1, x2 where x1 > x2, which is interpreted as a box crossing the anti-meridian. In this case the function splits the box along the anti-meridian and returns a multi-polygon.
- In all other cases, 2D geometries are assumed to _not cross the anti-meridian at all_.

Parameters

geometry – A geometry-like object

Returns

Shapely geometry object or None.

```
class xcube.core.schema.CubeSchema(shape: Sequence[int], coords: Mapping[str, DataArray], x_name: str
```

= 'lon', y_name: str = 'lat', time_name: str = 'time', dims:

Sequence[str] = None, chunks: Sequence[int] = None)

A schema that can be used to create new xcube datasets. The given *shape*, *dims*, and *chunks*, *coords* apply to all data variables.

- **shape** A tuple of dimension sizes.
- coords A dictionary of coordinate variables. Must have values for all dims.
- **dims** A sequence of dimension names. Defaults to ('time', 'lat', 'lon').

- chunks A tuple of chunk sizes in each dimension.
- property ndim: int

Number of dimensions.

property dims: Tuple[str, ...]

Tuple of dimension names.

property x_name: str

Name of the spatial x coordinate variable.

property y_name: str

Name of the spatial y coordinate variable.

property time_name: str

Name of the time coordinate variable.

property x_var: DataArray

Spatial x coordinate variable.

property y_var: DataArray

Spatial y coordinate variable.

property time_var: DataArray

Time coordinate variable.

property x_dim: str

Name of the spatial x dimension.

property y_dim: str

Name of the spatial y dimension.

property time_dim: str

Name of the time dimension.

property x_size: int

Size of the spatial x dimension.

property y_size: int

Size of the spatial y dimension.

property time_size: int

Size of the time dimension.

property shape: Tuple[int, ...]

Tuple of dimension sizes.

- property chunks: Tuple[int] | None
 Tuple of dimension chunk sizes.
- property coords: Dict[str, DataArray] Dictionary of coordinate variables.
- **classmethod new**(*cube: Dataset*) \rightarrow *CubeSchema* Create a cube schema from given *cube*.

Create a new Dask cluster.

Cloud resource tags can be specified in an environment variable XCUBE_DASK_CLUSTER_TAGS in the format tag_1=value_1:tag_2=value_2:...:tag_n=value_n. In case of conflicts, tags specified in resource_tags will override tags specified by the environment variable.

The cluster provider account name can be specified in an environment variable XCUBE_DASK_CLUSTER_ACCOUNT. If the account argument is given to new_cluster, it will override the value from the environment variable.

Return type

Cluster

Parameters

- **provider** (str) identifier of the provider to use. Currently, only 'coiled' is supported.
- **name** name to use as an identifier for the cluster
- **software** identifier for the software environment to be used.
- **n_workers** (int) number of workers in the cluster
- resource_tags tags to apply to the cloud resources forming the cluster
- **account** (str) cluster provider account name
- **kwargs further named arguments will be passed on to the cluster creation function
- **region** (str) default region where workers of the cluster will be deployed set to eu-central-1

5.15 Plugin Development

class xcube.util.extension.ExtensionRegistry

A registry of extensions. Typically used by plugins to register extensions.

has_extension(*point: str, name: str*) → bool

Test if an extension with given *point* and *name* is registered.

Return type bool

Parameters

- **point** (str) extension point identifier
- **name** (str) extension name

Returns

True, if extension exists

get_extension(point: str, name: str) \rightarrow Extension | None

Get registered extension for given *point* and *name*.

Parameters

• **point** (str) – extension point identifier

• **name** (str) – extension name

Returns

the extension or None, if no such exists

get_component(*point: str, name: str*) \rightarrow Any

Get extension component for given point and name. Raises a ValueError if no such extension exists.

Return type

Any

Parameters

- **point** (str) extension point identifier
- **name** (str) extension name

Returns

extension component

```
find_extensions(point: str, predicate: Callable[[Extension], bool] = None) \rightarrow List[Extension]
```

Find extensions for *point* and optional filter function *predicate*.

The filter function is called with an extension and should return a truth value to indicate a match or mismatch.

Parameters

- **point** (str) extension point identifier
- **predicate** optional filter function

Returns

list of matching extensions

find_components(point: str, predicate: Callable[[Extension], bool] = None) \rightarrow List[Any]

Find extension components for *point* and optional filter function *predicate*.

The filter function is called with an extension and should return a truth value to indicate a match or mismatch.

Parameters

- **point** (str) extension point identifier
- predicate optional filter function

Returns

list of matching extension components

add_extension(point: str, name: str, component: Any = None, loader: Callable[[Extension], Any] = None, **metadata) \rightarrow Extension

Register an extension *component* or an extension component *loader* for the given extension *point*, *name*, and additional *metadata*.

Either *component* or *loader* must be specified, but not both.

A given *loader* must be a callable with one positional argument *extension* of type *Extension* and is expected to return the actual extension component, which may be of any type. The *loader* will only be called once and only when the actual extension component is requested for the first time. Consider using the function *import_component()* to create a loader that lazily imports a component from a module and optionally executes it.

Return type

Extension

Parameters

- **point** (str) extension point identifier
- **name** (str) extension name
- component (Any) extension component
- loader extension component loader function
- metadata extension metadata

Returns

a registered extension

remove_extension(point: str, name: str)

Remove registered extension name from given point.

Parameters

- **point** (str) extension point identifier
- **name** (str) extension name

```
to_dict()
```

Get a JSON-serializable dictionary representation of this extension registry.

class xcube.util.extension.**Extension**(*point: str, name: str, component: Any = None, loader: Callable*[[Extension], Any] = None, **metadata)

An extension that provides a component of any type.

Extensions are registered in a *ExtensionRegistry*.

Extension objects are not meant to be instantiated directly. Instead, *ExtensionRegistry.add_extension()* is used to register extensions.

Parameters

- **point** extension point identifier
- **name** extension name
- **component** extension component
- loader extension component loader function
- metadata extension metadata

property is_lazy: bool

Whether this is a lazy extension that uses a loader.

property component: Any

Extension component.

property point: str

Extension point identifier.

property name: str

Extension name.

property metadata: Dict[str, Any]

Extension metadata.

 $\texttt{to_dict()} \rightarrow \textit{Dict[str, Any]}$

Get a JSON-serializable dictionary representation of this extension.

Return a component loader that imports a module or module component from *spec*. To import a module, *spec* should be the fully qualified module name. To import a component, *spec* must also append the component name to the fully qualified module name separated by a color (":") character.

An optional *transform* callable my be used to transform the imported component. If given, a new component is computed:

component = transform(component, extension)

If the *call* flag is set, the component is expected to be a callable which will be called using the given *call_args* and *call_kwargs* to produce a new component:

component = component(*call_kwargs, **call_kwargs)

Finally, the component is returned.

Parameters

- spec (str) String of the form "module_path" or "module_path:component_name"
- **transform** callable that takes two positional arguments, the imported component and the extension of type *Extension*
- call (bool) Whether to finally call the component with given *call_args* and *call_kwargs*
- call_args arguments passed to a callable component if call flag is set
- call_kwargs keyword arguments passed to callable component if call flag is set

Returns

a component loader

xcube.constants.EXTENSION_POINT_INPUT_PROCESSORS = 'xcube.core.gen.iproc'

The extension point identifier for input processor extensions

```
xcube.constants.EXTENSION_POINT_DATASET_IOS = 'xcube.core.dsio'
```

The extension point identifier for dataset I/O extensions

xcube.constants.EXTENSION_POINT_CLI_COMMANDS = 'xcube.cli'

The extension point identifier for CLI command extensions

Get populated extension registry.

xcube.util.plugin.get_plugins() → Dict[str, Dict]

Get mapping of "xcube_plugins" entry point names to JSON-serializable plugin meta-information.

CHAPTER

WEB API AND SERVER

xcube's RESTful web API is used to publish data cubes to clients. Using the API, clients can

- List configured xcube datasets;
- Get xcube dataset details including metadata, coordinate data, and metadata about all included variables;
- Get cube data;
- Extract time-series statistics from any variable given any geometry;
- Get spatial image tiles from any variable;
- Browse datasets and retrieve dataset data and metadata using the STAC API;
- Get places (GeoJSON features including vector data) that can be associated with xcube datasets;
- Perform compute operations on datasets, with the results calculated on demand and presented as new, dynamically generated datasets;
- Retrieve coverages backed by datasets using the OGC API Coverages API.

Later versions of API will also allow for xcube dataset management including generation, modification, and deletion of xcube datasets.

The complete description of all available functions is provided via openapi.html after starting the server locally. Please check out *Publishing xcube datasets* to learn how to do access it.

The web API is provided through the *xcube server* which is started using the *xcube serve* CLI command.

CHAPTER

SEVEN

VIEWER APP

The xcube viewer app is a simple, single-page web application to be used with the xcube server.

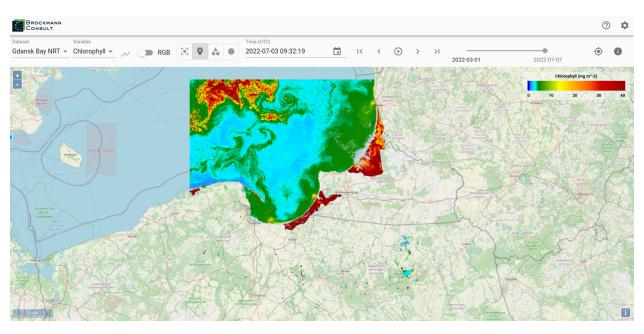
7.1 Demo

To test the viewer app, you can use the xcube viewer demo. This is our Brockmann Consult Demo xcube viewer. Via the viewer's settings it is possible to change the xcube server url which is used for displaying data. To do so open the viewer's settings panels, select "Server". A "Select Server" panel is opened, click the "+" button to add a new server. Here is demo server that you may add for testing:

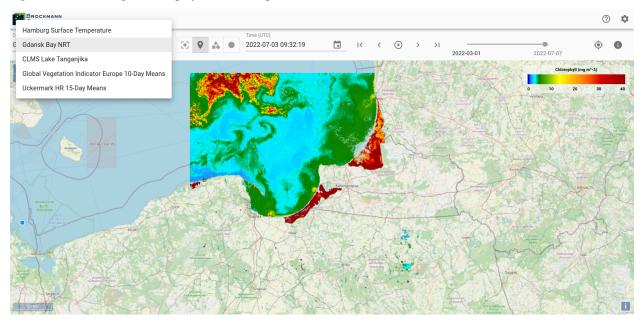
• Euro Data Cube Server (https://edc-api.brockmann-consult.de/api) has integrated amongst others a data cube with global essential climate variables (ECVs) variables from the ESA Earth System Data Lab Project. To access the Euro Data Cube viewer directly please visit https://edc-viewer.brockmann-consult.de .

7.2 Functionality

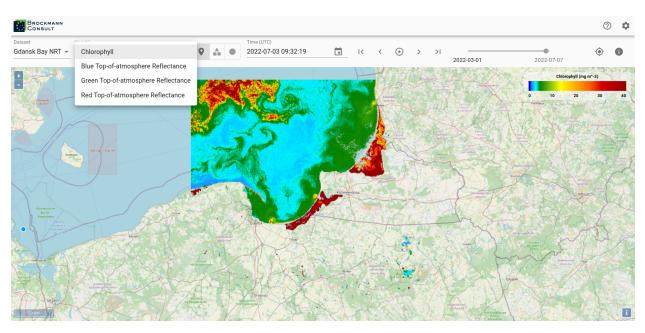
The xcube viewer functionality is described exemplary using the xcube viewer demo. The viewer visualizes data from the xcube datasets on top of a basemap. For zooming use the buttons in the top right corner of the map window or the zooming function of your computer mouse. A scale for the map is located in the lower right corner and in the upper left corner a corresponding legend to the mapped data of the data cube is available.



A xcube viewer may hold several xcube datasets which you can select via the drop-down menu *Dataset*. The viewed area automatically adjusts to a selected xcube dataset, meaning that if a newly selected dataset is located in a different region, the correct region is displayed on the map.



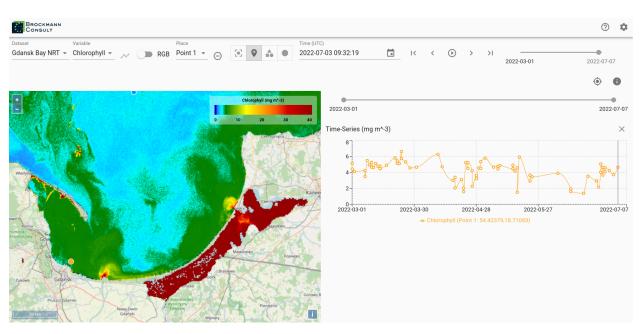
If more than one variable is available within a selected xcube dataset, you may change the variable by using the dropdown menu *Variable*.



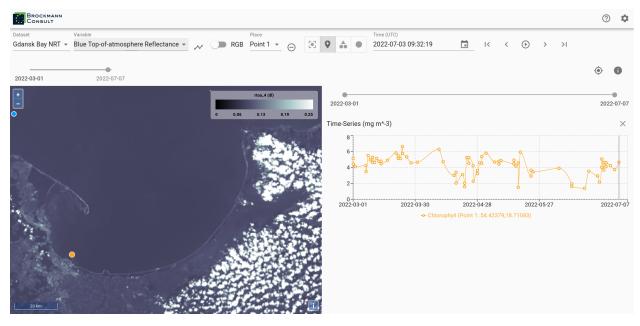
To see metadata for a dataset click on the *info*-icon on the right-hand side. Besides the dataset metadata you will see the metadata for the selected variable.

ROCKMANN Consult	0 🌣
Dataset Variable Place Place Circle 1 • O Image (UTC) 2022-04:3	22 09:37:54
	<u>ن</u>
Chorophyl (mg m-3)	1 1 * ? ×
0 10 20 30 40	Gdansk Bay NRT 🔳 🖒
	ID: bc-demo~Sweden/ws-s3-I2-gdansk-Sweden-v1.zarr
•	Dimension names lat, time, lon
	Dimension data types float64, datetime64[ns], float64
Denner A	Dimension lengths 1024, 273, 2048
Калининград	Geographical extent (x1, y1, x2, y2) 17.333333333333333333333333333333333333
	Spatial reference system EPSG:4326
	Chlorophyll Tr 🗉 <>
REAL Brankwo Satpatrichosex	Title Chlorophyll
Gdansk Grinko	Name Chlorophyll
Pruster Standal Before Standal Gorgeo Havestee	Units mg m^-3
Protect pourma 2 0 km Newy Dwor Editification Registrat	Data type float32

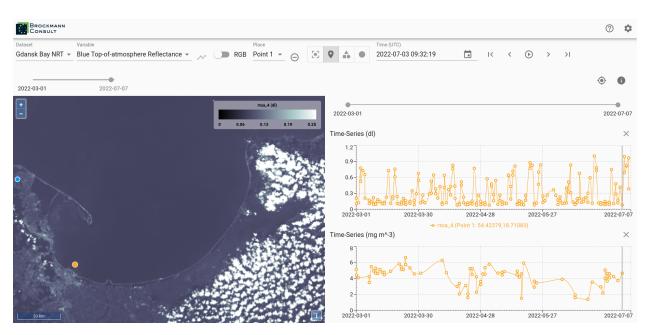
To obtain a time series set a point marker on the map and then select the *graph*-icon next to the *Variables* drop-down menu. You can select a different date by clicking into the time series graph on a value of interest. The data displayed in the viewer changes accordingly to the newly selected date.



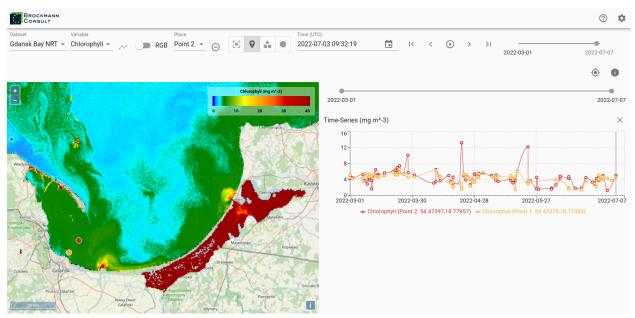
The current date is preserved when you select a different variable and the data of the variable is mapped for the date.



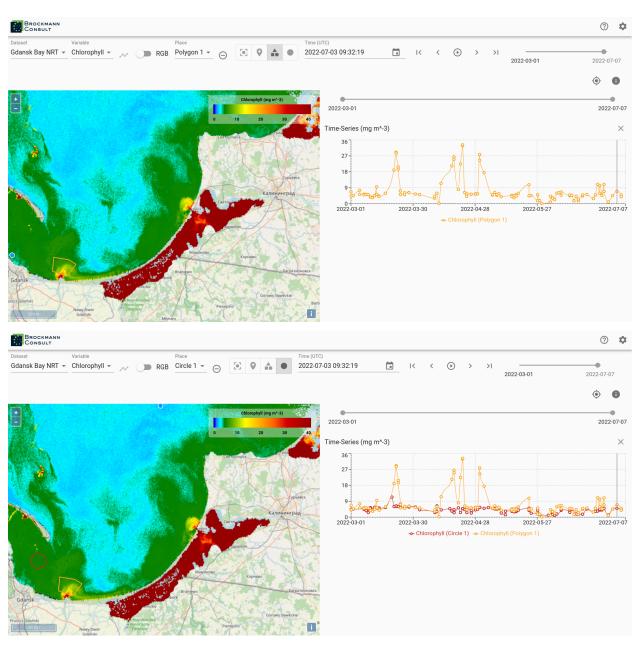
To generate a time series for the newly selected variable press the time series-icon again.



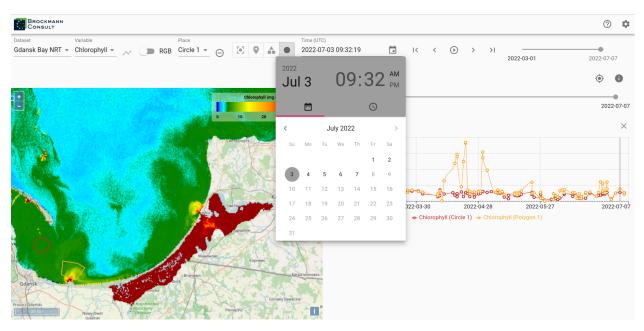
You may place multiple points on the map and you can generate time series for them. This allows a comparison between two locations. The color of the points corresponds to the color of the graph in the time series. You can find the coordinates of the point markers visualized in the time series beneath the graphs.



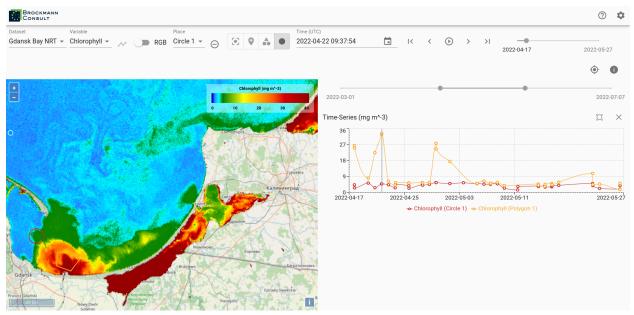
To delete a created location use the *remove*-icon next to the *Place* drop-down menu. Not only point location may be selected via the viewer, you can draw polygons and circular areas by using the icons on the right-hand side of the *Place* drop-down menu as well. You can visualize time series for areas, too.



In order to change the date for the data display use the calendar or step through the time line with the arrows on the right-hand side of the calendar.



When a time series is displayed two time-line tools are visible, the upper one for selecting the date displayed on the map of the viewer and the lower one may be used to narrow the time frame displayed in the time series graph. Just above the graph of the time series on the right-hand side is an *x*-icon for removing the time series from the view and to left of it is an icon which sets the time series back to the whole time extent.



To adjust the default settings select the *Settings*-icon on the very top right corner. There you have the possibility to change the server url, in order to view data which is available via a different server. You can choose a different language - if available - as well as set your preferences of displaying data and graph of the time series.

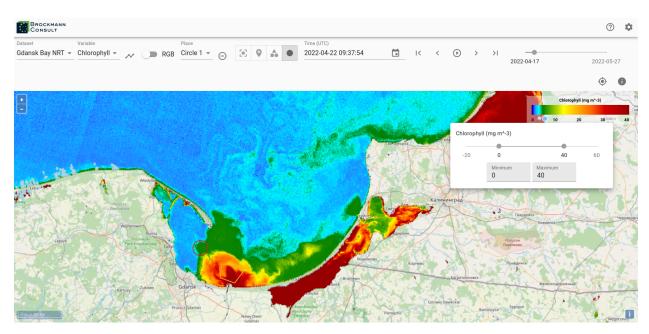
BROCKMANN						? 🌣
Dataset Gdansk Bay NRT →	Variable Chlorophyll - A RGB	Place Circle 1	Settings		>>>	
			General		2022-04-17	2022-05-27
+ o			Server BC DEMO Server			<u>ن</u>
		and and	Language English			2022-07-07
	and the second	i sente	Time interval of the player	1000 ms 👻		x Q
		10 Eggs.	Time-Series			
	t Reitig		Show graph after adding a place On	-	2	
A.			Show dots only, hide lines Off		2022-05-03 2022-05-11	2022-05-27
	and he		Show error bars On	-	II (Circle 1) Chlorophyll (Polygon 1)	
	1000 P	1-1-	Show median instead of mean (disables error bars) On	-		
Gdansk		Branlewo	Minimal number of data points in a time series update	20		
Pruszcz Gdański	thicko	1.	Мар			
20 km	Nowy Dwór Griański	100	Base map			

To see the map settings please scroll down in the settings window. There you can adjust the base map, switch the displayed projection between *Geographic* and *Mercator*. You can also choose to turn image smoothing on and to view the dataset boundaries.

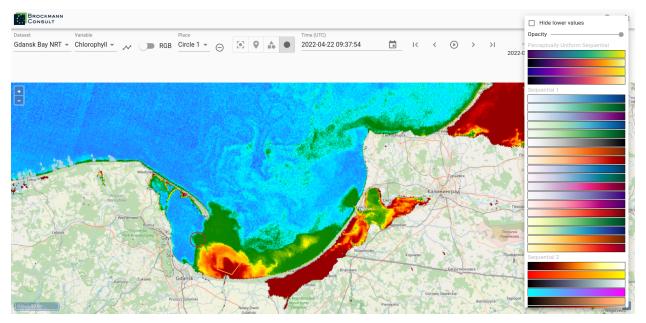
On the very bottom of the Settings pop-up window you can see information about the viewer and server version.

BROCKMANN	Show dots only, hide lines Off	> ()
Dataset Variable Place Gdansk Bay NRT Chlorophyll RGB Circle 1	Show error bars On	• > > <u>-</u> 2022-04-17 2022-05-27
	Show median instead of mean (disables error bars) On	• •
+ 0	Minimal number of data points in a time series update	2022-07-07
A State of the second sec	Мар	
and the second	Base map OpenStreetMap / OSM Mapnik	× ¤
	Projection Geographic O Merc	ator
	Image smoothing Off	
	Show dataset boundaries Off	2022-05-03 2022-05-11 2022-05-27 II (Circle 1) -> Chlorophyll (Polygon 1)
S Reader	System Information	
	xcube Viewer version 0.11.1	
Gdanist and a state of the stat	xcube Server version 0.11.2	
Prusez Bduński 20 km Nowy Dwár Gdański	Penietto	

Back to the general view, if you would like to change the value ranges of the displayed variable you can do it by clicking into the area of the legend where the value ticks are located or you can enter the desired values in the *Minimum* and/or *Maximum* text field.



You can change the color mapping as well by clicking into the color range of the legend. There you can also decide to hide lower values and it is possible to adjust the opacity.



The xcube viewer app is constantly evolving and enhancements are added, therefore please be aware that the above described features may not always be completely up-to-date.

7.3 Build and Deploy

You can also build and deploy your own viewer instance. In the latter case, visit the xcube-viewer repository on GitHub and follow the instructions provides in the related README file.

CHAPTER

EIGHT

DATA ACCESS

In xcube, data cubes are raster datasets that are basically a collection of N-dimensional geo-physical variables represented by xarray.Dataset Python objects (see also *xcube Dataset Convention*). Data cubes may be provided by a variety of sources and may be stored using different data formats. In the simplest case you have a NetCDF file or a Zarr directory in your local filesystem that already represents a data cube. Data cubes may also be stored on AWS S3 or Google Cloud Storage using the Zarr format. Sometimes a set of NetCDF or GeoTIFF files in some storage must first be concatenated to form a data cube. In other cases, data cubes can be generated on-the-fly by suitable requests to some cloud-hosted data API such as the ESA Climate Data Centre or Sentinel Hub.

8.1 Data Store Framework

The *xcube data store framework* provides a simple and consistent Python interface that is used to open xarray.Dataset and other data objects from *data stores* which abstract away the individual data sources, protocols, formats and hides involved data processing steps. For example, the following two lines open a data cube from the ESA Climate Data Centre comprising the essential climate variable Sea Surface Temperature (SST):

Often, and in the example above, data stores create data cube *views* on a given data source. That is, the actual data arrays are subdivided into chunks and each chunk is fetched from the source in a "lazy" manner. In such cases, the xarray.Dataset's variables are backed by Dask arrays. This allows data cubes to be virtually of any size.

Data stores can provide the data using different Python in-memory representations or data types. The most common representation for a data cube is an xarray.Dataset instance, multi-resolution data cubes would be represented as a xcube MultiLevelDataset instance (see also xcube Multi-Level Dataset Convention). Vector data is usually provided as an instance of geopandas.GeoDataFrame.

Data stores can also be writable. All read-only data stores share the same functional interface share the same functional interface and so do writable data stores. Of course, different data stores will have different configuration parameters. Also, the parameters passed to the open_data() method, or respectively the write_data() method, may change based on the store's capabilities. Depending on what is offered by a given data store, also the parameters passed to the open_data() method, may change.

The xcube data store framework is exported from the xcube.core.store package, see also its API reference.

The DataStore abstract base class is the primary user interface for accessing data in xcube. The most important operations of a data store are:

- list_data_ids() enumerate the datasets of a data store by returning their data identifiers;
- describe_data(data_id) describe a given dataset in terms of its metadata by returning a specific DataDescriptor, e.g., a DatasetDescriptor;

- search_data(...) search for datasets in the data store and return a DataDescriptor iterator;
- open_data(data_id, ...) open a given dataset and return, e.g., an xarray.Dataset instance.

The MutableDataStore abstract base class represents a writable data store and extends DataStore by the following operations:

- write_data(dataset, data_id, ...) write a dataset to the data store;
- delete_data(data_id) delete a dataset from the data store;

Above, the ellipses . . . are used to indicate store-specific parameters that are passed as keyword-arguments. For a given data store instance, it is not obvious what are parameters are allowed. Therefore, data stores provide a programmatic way to describe the allowed parameters for theoperations of a given data store by the means of a parameter schema:

- get_open_data_params_schema() describes parameters of open_data();
- get_search_data_params_schema() describes parameters of search_data();
- get_write_data_params_schema() describes parameters of write_data().

All operations return an instance of a JSON Object Schema. The JSON object's properties describe the set of allowed and required parameters as well as the type and value range of each parameter. The schemas are also used internally to validate the parameters passed by the user.

xcube comes with a predefined set of writable, filesystem-based data stores, Since data stores are xcube extensions, additional data stores can be added by xcube plugins. The data store framework provides a number of global functions that can be used to access the available data stores:

- find_data_store_extensions() -> list[Extension] get a list of xcube data store extensions;
- new_data_store(store_id, ...) -> DataStore instantiate a data store with store-specific parameters;
- get_data_store_params_schema(store_id) -> Schema describe the store-specific parameters that must/can be passed to new_data_store() as JSON Object Schema.

The following example outputs all installed data stores:

```
from xcube.core.store import find_data_store_extensions
for ex in find_data_store_extensions():
    store_id = ex.name
    store_md = ex.metadata
    print(store_id, "-", store_md.get("description"))
```

If one of the installed data stores is, e.g. sentinelhub, you could further introspect its specific parameters and datasets as shown in the following example:

(continues on next page)

(continued from previous page)

8.2 Available Data Stores

This sections lists briefly the official data stores available for xcube. We provide the store identifier, list the store parameters, and list the common parameters used to open data cubes, i.e., xarray.Dataset instances.

Note that some data stores the open parameters may differ by from dataset to dataset depending on the actual dataset layout, coordinate references system or data type. Some data stores may also provide vector data.

For every data store we also provide a dedicated example Notebook that demonstrates its specific usage in examples/notebooks/datastores.

8.2.1 Filesystem-based data stores

The following filesystem-based data stores are available in xcube:

- "file" for the local filesystem;
- "s3" for AWS S3 compatible object storage;
- "abfs" for Azure blob storage;
- "memory" for mimicking an in-memory filesystem.

All filesystem-based data store have the following parameters:

- root: str The root directory of the store in the filesystem. Defaults to ''.
- max_depth: int Maximum directory traversal depth. Defaults to 1.
- read_only: bool Whether this store is read-only. Defaults to False.
- includes: list[str] A list of paths to include into the store. May contain wildcards * and ?. Defaults to UNDEFINED.
- excludes: list[str] A list of paths to exclude from the store. May contain wildcards * and ?. Defaults to UNDEFINED.
- storage_options: dict[str, any] Filesystem-specific options.

The parameter storage_options is filesystem-specific. Valid storage_options for all filesystem data stores are:

use_listings_cache: bool

- listings_expiry_time: float
- max_paths: int
- skip_instance_cache: bool
- asynchronous: bool

The following storage_options can be used for the file data store:

• auto_mkdirs: bool - Whether, when opening a file, the directory containing it should be created (if it doesn't already exist).

The following storage_options can be used for the s3 data store:

- anon: bool Whether to anonymously connect to AWS S3.
- key: str AWS access key identifier.
- secret: str AWS secret access key.
- token: str Session token.
- use_ssl: bool Whether to use SSL in connections to S3; may be faster without, but insecure. Defaults to True.
- requester_pays: bool If "RequesterPays" buckets are supported. Defaults to False.
- s3_additional_kwargs: dict parameters that are used when calling S3 API methods. Typically, used for things like "ServerSideEncryption".
- client_kwargs: dict Parameters for the botocore client.

The following storage_options can be used for the abfs data store:

- anon: bool Whether to anonymously connect to Azure Blob Storage.
- account_name: str Azure storage account name.
- account_key: str Azure storage account key.
- connection_string: str Connection string for Azure blob storage.

All filesystem data stores can open datasets from various data formats. Datasets in Zarr, GeoTIFF / COG, or NetCDF format will be provided either by xarray.Dataset or xcube MultiLevelDataset instances. Datasets stored in GeoJSON or ESRI Shapefile will yield geopandas.GeoDataFrame instances.

Common parameters for opening xarray.Dataset instances:

- cache_size: int Defaults to UNDEFINED.
- group: str Group path. (a.k.a. path in zarr terminology.). Defaults to UNDEFINED.
- chunks: dict[str, int | str] Optional chunk sizes along each dimension. Chunk size values may be None, "auto" or an integer value. Defaults to UNDEFINED.
- decode_cf: bool Whether to decode these variables, assuming they were saved according to CF conventions. Defaults to True.
- mask_and_scale: bool If True, replace array values equal to attribute "_FillValue" with NaN. Use "scale_factor" and "add_offset" attributes to compute actual values.. Defaults to True.
- decode_times: bool If True, decode times encoded in the standard NetCDF datetime format into datetime objects. Otherwise, leave them encoded as numbers.. Defaults to True.
- decode_coords: bool If True, decode the "coordinates" attribute to identify coordinates in the resulting dataset. Defaults to True.

- drop_variables: list[str] List of names of variables to be dropped. Defaults to UNDEFINED.
- consolidated: bool Whether to open the store using Zarr's consolidated metadata capability. Only works for stores that have already been consolidated. Defaults to False.
- log_access: bool Defaults to False.

8.2.2 ESA Climate Data Centre cciodp

The data store cciodp provides the datasets of the ESA Climate Data Centre.

This data store is provided by the xcube plugin xcube-cci. You can install it using conda install -c conda-forge xcube-cci.

Data store parameters:

- endpoint_url: str Defaults to 'https://archive.opensearch.ceda.ac.uk/opensearch/ request'.
- endpoint_description_url: str Defaults to 'https://archive.opensearch.ceda.ac.uk/ opensearch/description.xml?parentIdentifier=cci'.
- enable_warnings: bool Whether to output warnings. Defaults to False.
- num_retries: int Number of retries when requesting data fails. Defaults to 200.
- retry_backoff_max: int Defaults to 40.
- retry_backoff_base: float Defaults to 1.001.

Common parameters for opening xarray.Dataset instances:

- variable_names: list[str] List of variable names. Defaults to all.
- bbox: (float, float, float) Bounding box in geographical coordinates.
- time_range: (str, str) Time range.
- normalize_data: bool Whether to normalize and sanitize the data. Defaults to True.

8.2.3 ESA Climate Data Centre ccizarr

A subset of the datasets of the cciodp store have been made available using the Zarr format using the data store ccizarr. It provides much better data access performance.

It has no dedicated data store parameters.

Its common dataset open parameters for opening xarray.Dataset instances are the same as for the filesystem-based data stores described above.

8.2.4 ESA SMOS

A data store for ESA SMOS data is currently under development and will be released soon.

This data store is provided by the xcube plugin xcube-smos. Once available, you will be able to install it using conda install -c conda-forge xcube-smos.

8.2.5 Copernicus Climate Data Store cds

The data store cds provides datasets of the Copernicus Climate Data Store.

This data store is provided by the xcube plugin xcube-cds. You can install it using conda install -c conda-forge xcube-cds.

Data store parameters:

- cds_api_key: str User API key for Copernicus Climate Data Store.
- endpoint_url: str API endpoint URL.
- num_retries: int Defaults to 200.
- normalize_names: bool Defaults to False.

Common parameters for opening xarray.Dataset instances:

- bbox: (float, float, float) Bounding box in geographical coordinates.
- time_range: (str, str) Time range.
- variable_names: list[str] List of names of variables to be included. Defaults to all.
- spatial_res: float Spatial resolution. Defaults to 0.1.

8.2.6 Copernicus Marine Service cmems

The data store cmems provides datasets of the Copernicus Marine Service.

This data store is provided by the xcube plugin xcube-cmems. You can install it using conda install -c conda-forge xcube-cmems.

Data store parameters:

- cmems_username: str CMEMS API username
- cmems_password: str CMEMS API password
- cas_url: str Defaults to 'https://cmems-cas.cls.fr/cas/login'.
- csw_url: str Defaults to 'https://cmems-catalog-ro.cls.fr/geonetwork/srv/eng/ csw-MYOCEAN-CORE-PRODUCTS?'.
- databases: str One of ['nrt', 'my'].
- server: str Defaults to 'cmems-du.eu/thredds/dodsC/'.

Common parameters for opening xarray.Dataset instances:

- variable_names: list[str] List of variable names.
- time_range: [str, str] Time range.

8.2.7 Sentinel Hub API

The data store sentinelhub provides the datasets of the Sentinel Hub API.

This data store is provided by the xcube plugin xcube-sh. You can install it using conda install -c conda-forge xcube-sh.

Data store parameters:

- client_id: str Sentinel Hub API client identifier
- client_secret: str Sentinel Hub API client secret
- api_url: str Sentinel Hub API URL. Defaults to 'https://services.sentinel-hub.com'.
- oauth2_url: str Sentinel Hub API authorisation URL. Defaults to 'https://services. sentinel-hub.com/oauth'.
- enable_warnings: bool Whether to output warnings. Defaults to False.
- error_policy: str Policy for errors while requesting data. Defaults to 'fail'.
- num_retries: int Number of retries when requesting data fails. Defaults to 200.
- retry_backoff_max: int Defaults to 40.
- retry_backoff_base: number Defaults to 1.001.

Common parameters for opening xarray.Dataset instances:

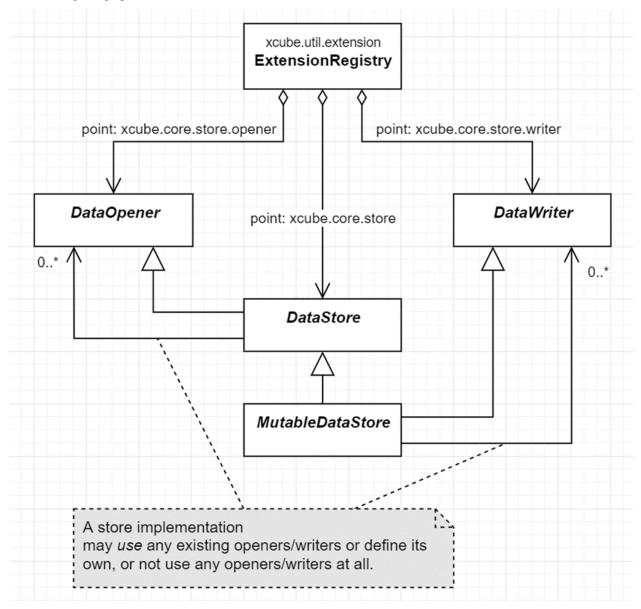
- bbox: (float, float, float) Bounding box in coordinate units of crs. Required.
- crs: str Defaults to 'WGS84'.
- time_range: (str, str) Time range. Required.
- variable_names: list[str] List of variable names. Defaults to all.
- variable_fill_values: list[float] List of fill values according to variable_names
- variable_sample_types: list[str] List of sample types according to variable_names
- variable_units: list[str] List of sample units according to variable_names
- tile_size: (int, int) Defaults to (1000, 1000).
- spatial_res: float Required.
- upsampling: str Defaults to 'NEAREST'.
- downsampling: str Defaults to 'NEAREST'.
- mosaicking_order: str Defaults to 'mostRecent'.
- time_period: str Defaults to '1D'.
- time_tolerance: str Defaults to '10M'.
- collection_id: str Name of the collection.
- four_d: bool Defaults to False.
- extra_search_params: dict Extra search parameters passed to a catalogue query.
- max_cache_size: int Maximum chunk cache size in bytes.

8.3 Developing new data stores

8.3.1 Implementing the data store

New data stores can be developed by implementing the xcube DataStore interface for read-only data store, or the MutableDataStore interface for writable data stores, and should follow the *xcube Data Store Conventions*.

If a data store supports combinations of Python data types, external storagestypes, and/or data formats it should consider the following design pattern:



Here, we implement a dedicated DataOpener for a suitable combination of supported Python data types, external storages types, and/or data formats. The DataStore, which implements the DataOpener interface delegates to specialized DataOpener implementations based on the open parameters passed to the open_data() method. The same holds for the DataWriter implementations for a MutableDataStore.

New data stores that are backed by some cloud-based data API can make use the xcube GenericZarrStore to implement the lazy fetching of data array chunks from the API.

8.3.2 Registering the data store

To register the new data store with xcube, it must be provided as a Python package. Based on the package's name there are to ways to register it with xcube. If your package name matches the pattern xcube_*, then you would need to provide a function init_plugin() in the package's plugin module (hence {package}.plugin.init_plugin()).

Alternatively, the package can have any name, but then it must register a setuptools entry point in the slot "xcube_plugins". In this case the function init_plugin() can also be placed anywhere in your code. If you use setup.cfg:

```
[options.entry_points]
xcube_plugins =
    {your_name} = {your_package}.plugin:init_plugin
```

If you are (still) using setup.py:

```
from setuptools import setup
setup(
    # ...,
    entry_points={
        'xcube_plugins': [
           '{your_name} = {your_package}.plugin:init_plugin',
        ]
    }
)
```

The function init_plugin will be implemented as follows:

```
from xcube.constants import EXTENSION_POINT_DATA_OPENERS
from xcube.constants import EXTENSION_POINT_DATA_STORES
from xcube.constants import EXTENSION_POINT_DATA_WRITERS
from xcube.util import extension
def init_plugin(ext_registry: extension.ExtensionRegistry):
    # register your DataStore extension
    ext_registry.add_extension(
        loader=extension.import_component(
            '{your_package}.store:{YourStoreClass}'),
        point=EXTENSION_POINT_DATA_STORES,
        name="{your_store_id}",
        description='{your store description}'
   )
    # register any extra DataOpener (EXTENSION_POINT_DATA_OPENERS)
    # or DataWriter (EXTENSION_POINT_DATA_WRITERS) extensions (optional)
    ext_registry.add_extension(
        loader=extension.import_component(
            '{your_package}.opener:{YourOpenerClass}'),
        point=EXTENSION_POINT_DATA_OPENERS,
        name="{your_opener_id}",
        description='{your opener description}'
   )
```

CHAPTER

THE XCUBE GENERATOR

9.1 Introduction

The *generator* is an xcube feature which allows users to create, manipulate, and write xcube datasets according to a supplied configuration. The same configuration can be used to generate a dataset on the user's local computer or remotely, using an online server.

The generator offers two main user interfaces: A Python API, configured using Python objects; and a command-line interface, configured using YAML or JSON files. The Python and file-based configurations have the same structure and are interconvertible.

The online generator service interfaces with the xcube client via a well-defined REST API; it is also possible for thirdparty clients to make use of this API directly, though it is expected that the Python and command-line interfaces will be more convenient in most cases.

9.2 Further documentation

This document aims to provide a brief overview of the generation process and the available configuration options. More details are available in other documents and in the code itself:

- Probably the most thorough documentation is available in the Jupyter demo notebooks in the xcube repository. These can be run in any JupyterLab environment containing an xcube installation. They combine explanation with interactive worked examples to demonstrate practical usage of the generator in typical use cases.
- For the Python API in particular, the xcube API documentation is generated from the docstrings included in the code itself and serves as a detailed low-level reference for individual Python classes and methods. The docstrings can also be read from a Python environment (e.g. using the ? postfix in IPython or JupyterLab) or, of course, by browsing the source code itself.
- For the YAML/JSON configuration syntax used with the command-line interface, there are several examples available in the examples/gen2/configs subdirectory of the xcube repository.
- For the REST API underlying the Python and command-line interfaces, there is a formal definition on Swagger-Hub, and one of the example notebooks demonstrates its usage with the Python *requests* library.

9.3 The generation process

The usual cube generation process is as follows:

- 1. The generator opens the input data store using the store identifier and parameters in the *input configuration*.
- 2. The generator reads from the input store the data specified in the *cube configuration* and uses them to create a data cube, often with additional manipulation steps such as resampling the data.
- 3. If an optional *code configuration* has been given, the user-supplied code is run on the created data cube, potentially modifying it.
- 4. The generator writes the generated cube to the data store specified in the *output configuration*.

9.4 Invoking the generator from a Python environment

The configurations for the various parts of the generator are used to initialize a GeneratorRequest, which is then passed to xcube.core.gen2.generator.CubeGenerator.generate_cube. The generate_cube method returns a *cube reference* which can be used to open the cube from the output data store.

The generator can also be directly invoked with a configuration file from a Python environment, using the xcube. core.gen2.generator.CubeGenerator.from_file method.

9.5 Invoking the generator from the command line

The generator can be invoked from the command line using the xcube gen2 subcommand. (Note: the subcommand xcube gen invokes an earlier, deprecated generator feature which is not compatible with the generator framework described here.)

9.6 Configuration syntax

All Python configuration classes are defined in the xcube.core.gen2 package, except for CodeConfig, which is in xcube.core.byoa.

The types in the parameter tables are given in an ad-hoc, semi-formal notation whose corresponding Python and JSON representations should be obvious. For the formal Python type definitions, see the signatures of the __init__ methods of the configuration classes; for the formal JSON type definitions, see the JSON schemata (in JSON Schema format) produced by the get_schema methods of the configuration classes.

9.6.1 Remote generator service configuration

The command-line interface allows a *service configuration* for the remote generator service to be provided as a YAML or JSON file. This file defines the endpoint and access credentials for an online generator service. If it is provided, the specified remote service will be used to generate the cube. If it is omitted, the cube will be generated locally. The configuration file defines three values: endpoint_url, client_id, and client_secret. A typical service configuration YAML file might look as follows:

```
endpoint_url: "https://xcube-gen.brockmann-consult.de/api/v2/"
client_id: "93da366d7c39517865e4f141ddf1dd81"
client_secret: "d210aG91dCByZXN0cmljdGlvbiwgaW5jbHVkaW5nIHd"
```

9.6.2 Store configuration

In the command-line interface, an additional YAML or JSON file containing one or more *store configurations* may be supplied. A store configuration encapsulates a data store ID and an associated set of store parameters, which can then be referenced by an associated *store configuration identifier*. This identifier can be used in the input configuration, as described below. A typical YAML store configuration might look as follows:

```
sentinelhub_eu:
  title: SENTINEL Hub (Central Europe)
  description: Datasets from the SENTINEL Hub API deployment in Central Europe
  store id: sentinelhub
  store_params:
   api_url: https://services.sentinel-hub.com
    client_id: myid123
    client_secret: 0c5892208a0a82f1599df026b5e19017
cds:
  title: C3S Climate Data Store (CDS)
  description: Selected datasets from the Copernicus CDS API
  store_id: cds
  store_params:
   normalize_names: true
   num_retries: 3
my_data_bucket:
  title: S3 output bucket
  description: An S3 bucket for output data sets
  store_id: s3
  store_params:
   root: cube-outputs
    storage_options:
      key: gwerty12345
      secret: 7ff889c0aea254d5e00440858289b85c
      client_kwargs:
        endpoint_url: https://my-endpoint.some-domain.org/
```

9.6.3 Input configuration

The input configuration defines the data store from which data for the cube are to be read, and any additional parameters which this data store requires.

The Python configuration object is InputConfig; the corresponding YAML configuration section is input_configs.

Parameter	Required?	Туре	Description
store_id	Ν	str	Identifier for the data store
opener_id	Ν	str	Identifier for the data opener
data_id	Y	str	Identifier for the dataset
store_params	Ν	$map(str \rightarrow *)$	Parameters for the data store
open_params	Ν	$map(str \rightarrow *)$	Parameters for the data opener

store_id is a string identifier for a particular xcube data store, defined by the data store itself. If a store configuration file has been supplied (see above), a store configuration identifier can also be supplied here in place of a 'plain' store

identifier. Store configuration identifiers must be prefixed by an @ symbol. If a store configuration identifier is supplied in place of a store identifier, store_params values will be supplied from the predefined store configuration and can be omitted from the input configuration.

data_id is a string identifier for the dataset within a particular store.

The format and content of the store_params and open_params dictionaries is defined by the individual store or opener.

The generator service does not yet provide a remote interface to list available data stores, datasets, and store parameters (i.e. allowed values for the parameters in the table above). In a local xcube Python environment, you can list the currently available store identifiers with the expression list(map(lambda e: e.name, xcube.core.store.find_data_store_extensions())). You can create a local store object for an identifier store_id with xcube.core.store.get_data_store_instance(store_id).store. The store object provides methods list_data_ids, get_data_store_params_schema, and get_open_data_params_schema to describe the allowed values for the corresponding parameters. Note that the available stores and datasets on a remote xcube generator server may not be the same as those available in your local xcube environment.

9.6.4 Cube configuration

This configuration element defines the characteristics of the cube that should be generated. The Python configuration class is called CubeConfig, and the YAML section cube_config. All parameters are optional and will be filled in with defaults if omitted; the default values are dependent on the data store and dataset.

Parameter	Туре	Units/Description
variable_name	[str,]	Available variables are data store dependent.
crs	str	PROJ string, JSON string with PROJ parameters, CRS WKT string, or authority string
bbox	[float, float, float, float]	Bounding-box (min_x, min_y, max_x, max_y) CRS-dependent, usually degrees
spatial_res	float or [float, float]	CRS-dependent, usually degrees
tile_size	int or [int, int]	pixels
time_range	str or [str, str]	ISO 8601 subset
time_period	str	integer + unit
chunks	$map(str \rightarrow null/int)$	maps variable names to chunk sizes

The crs parameter string is interpreted using `CRS.from_string in the pyproj package <https://pyproj4.github.io/ pyproj/dev/api/crs/crs.html#pyproj.crs.CRS.from_string>`___ and therefore accepts the same specifiers.

time_range specified the start and end of the requested time range. can be specified either as a date in the format YYYY-MM-DD or as a date and time in the format YYYY-MM-DD HH:MM:SS. If the time is omitted, it is taken to be 00:00:00 (the start of the day) for the start specifier and 24:00:00 (the end of the day) for the specifier. The end specifier may be omitted; in this case the current time is used.

time_period specified the duration of a single time step in the requested cube, which determines the temporal resolution. It consists of an integer denoting the number of time units, followed by single upper-case letter denoting the time unit. Valid time unit specifiers are D (day), W (week), M (month), and Y (year). Examples of time_period values: 1Y (one year), 2M (two months), 10D (ten days).

The value of the chunks mapping determines how the generated data is chunked for storage. The chunking has no effect on the data itself, but can have a dramatic impact on data access speeds in different scenarios. The value of chunks is structured a map from variable names (corresponding to those specified by the variable_names parameter) to chunk sizes.

9.6.5 Code configuration

The code configuration supports multiple ways to define a *dataset processor* – fundamentally, a Python function which takes a dataset and returns a processed version of the input dataset. Since the code configuration can work directly with instantiated Python objects (which can't be stored in a YAML file), there are some differences in code configuration between the Python API and the YAML format.

Parameter	Туре	Units/description
_callable †	Callable	Function to be called to process the datacube. Only available via Python API
callable_ref	str (non-empty)	A reference to a Python class or function, in the format <module>:< function_or_class></module>
callable_para	$map(str \rightarrow *)$	Parameters to be passed to the specified callable
inline_code †	str (non-empty)	An inline snippet of Python code
file_set†	FileSet (Python) / map (YAML)	A bundle of Python modules or packages (see details below)
install_requi	boolean	If set, indicates that file_set contains modules or packages to be installed.

All parameters are optional (as is the entire code configuration itself). The three parameters marked † are mutually exclusive: at most one of them may be given.

_callable provides the dataset processor directly and is only available in the Python API. It must be either a function or a class.

- If a function, it takes a Dataset and optional additional named parameters, and returns a Dataset. Any additional parameters are supplied in the callable_params parameter of the code configuration.
- If an object, it must implement a method process_dataset, which is treated like the function described above, and may optionally implement a class method get_process_params_schema, which returns a JsonObjectSchema describing the additional parameters. For convenience and clarity, the object may extend the abstract base class DatasetProcessor, which declares both these methods.

callable_ref is a string with the structure <module>:<function_or_class>, and specifies the function or class to call when inline_code or file_set is provided. The specified function or class is handled like the _callable parameter described above.

callable_params specifies a dictionary of named parameters which are passed to the processor function or method.

inline_code is a string containing Python source code. If supplied, it should contain the definition of a function or object as described for the _callable parameter. The module and class identifiers for the callable in the inline code snippet should be specified in callable_ref parameter.

file_set specifies a set of files which should be read from an fsspec file system and which contain a definition of a dataset processor. As with inline_code, the parameter callable_ref should also be supplied to tell the generator which class or function in the file set is the actual processor. The parameters of file_set are identical with those of the constructor of the corresponding Python FileSet class, and are as follows:

Parameter	Туре	Description
path	str	fsspec-compatible root path specifier
sub_path	str	optional sub-path to append to main path
includes	[str]	include files matching any of these patterns
excludes	[str]	exclude files matching any of these patterns
storage_params	$map(str \rightarrow *)$	FS-specific parameters (passed to fsspec)

9.6.6 Output configuration

This configuration element determines where the generated cube should be written to. The Python configuration class is called OutputConfig, and the YAML section output_config.

Parameter	Туре	Units/description
store_id	str	Identifier of output store
writer_id	str	Identifier of data writer
data_id	str	Identifier under which to write the cube
store_params	$map(str \rightarrow *)$	Store-dependent parameters for output store
write_params	$map(str \rightarrow *)$	Writer-dependent parameters for output writer
replace	bool	If true, replace any existing data with the same identifier.

CHAPTER

TEN

XCUBE DATASET CONVENTION

Version 1.0 Draft, last updated April 28, 2023

10.1 Introduction

The purpose of this document is to define a convention for how geospatial data cubes accessed and generated by xcube should look. The goal of the convention is to ease access to data, make it compliant to existing data standards, make it comprehensive and understandable, and minimize the effort to ingest the data into users' processing, visualisation, and analysis code. In short, the goal is to make xcube datasets analysis-ready. The intention is neither to invent any new data formats nor to try to establish new "standards". Instead, the xcube conventions solely build on existing, popular, well-established data formats and data standards.

This convention applies to gridded (array) datasets only.

In this document, the verbs *must*, *should*, and *may* have a special meaning when used to describe conventions. A *must* is a mandatory requirement that must be fulfilled, *should* is optional but always recommended, and *may* is optional and recommended in cases (like good to have).

10.2 Table of Contents

- Data Model and Format
- Metadata
- Spatial Reference
- Temporal Reference
- Encoding of Units
- Encoding of Flags
- Encoding of Missing Values
- Encoding of Scale/Offset Values
- Multi-Resolution Datasets
- Multi-Band Datasets
- Metadata Consolidation
- Zip Archives
- Dataset Examples

10.3 Data Model and Format

The data model for xcube gridded datasets is a subset of the Unidata Common Data Model (CDM). The CDM has originally been established for NetCDF (Network Common Data Form) but is implemented also for the OPenDAP protocol and the HDF and Zarr formats. Other common gridded data models can be easily translated into the CDM, for example GeoTIFF.

xcube gridded datasets must fully comply to the *Climate and Forecast Metadata Conventions*, CF Conventions for short. The following sections in this chapter describe how these conventions are tailored and applied to xcube datasets. By default, xcube gridded datasets are made available in the Zarr format v2 as this format is ideal for Cloud storage, e.g. Object Storage such as AWS S3, and can be easily converted to other formats such as NetCDF or TIFF. We consider the popular xarray Python package to be the primary in-memory representation for xcube gridded datasets. The xarray.Dataset data model represents a subset of the CDM, interprets CF compliant data correctly, and we will follow this simplified CDM model for xcube gridded datasets in both structure and terminology:

- A dataset is a container for variables. A dataset has metadata in form of a key-value mapping (global attributes).
- A variable has N dimensions in a prescribed order, and each dimension has a name and a size. A variable contains the actual gridded data in the form of an N-dimensional data array that has the shape of the dimension sizes. The data type of the array should be numeric, however also text strings and complex types are allowed. A variable has metadata in form of a key-value mapping (variable attributes). There are two types of variables, *coordinates* and *data variables*.
- **Coordinates** usually contain the labels for a dimension. For example, the coordinate lon could have a 1-D array that contains 3600 longitude values for the dimension lon of size 3600. However, coordinates may also be 2-D. For example. a dataset using satellite geometry may provide its lon coordinate as a 2-D array for the dimensions **x** and **y** (for which co-ordinates should also exist).
- All variables that are not coordinates are **data variables**. They provide the actual dataset's data. For each of the dimensions used by a data variable, a corresponding coordinate must exist. For example, for a data variable NDVI with dimensions time, lat, lon, the coordinates time, lat, lon must exist too.

10.4 Metadata

Global and variable metadata must follow the recommendations of the ESIP Attribute Convention for Data Discovery v1.3. If other metadata attributes are used, they should be covered by the CF Conventions v1.8.

10.5 Spatial Reference

The spatial dimensions of a data variable must be the innermost dimensions, namely lat, lon for geographic (EPGS:4326, WGS-84) grids, and y, x for other grids – in this order. Datasets that use a geographic (EPGS:4326, WGS-84) grid must provide the 1-D coordinates lat for dimension lat and lon for dimension lon. Datasets that use a non-geographic grid must provide the 1-D coordinates y for dimension y and x for dimension x.

- In case the grid refers to a known spatial reference system (projected CRS), the dataset must make use of the CRS encoding described in the CF Conventions on Grid Mapping, i.e. add a variable crs.
- In case the grid is referring to satellite viewing geometry, the dataset must provide 2-D coordinates lat and lon both having the dimensions y, x in exactly this order, and apply the CF Conventions on 2-D Lat and Lon.

It is expected that the 1-D coordinates have a uniform spacing, i.e. there should be a unique linear mapping between coordinate values and the image grid. Ideally, the spacing should also be the same in x- and y-direction. Note, it is always a good practice to add geographic coordinates to non-geographic, projected grids. Therefore, a dataset may

also provide the 2-D coordinates lat and lon in this case. Spatial coordinates must follow the CF Conventions on Coordinates.

10.6 Temporal Reference

The temporal dimension of a data variables should be named time. It should be the variable's outermost dimension. A corresponding coordinate named time must exist and the CF Conventions on the Time Coordinate must be applied. It is recommended to use the unit seconds since 1970.01.01.

Data variables may contain other dimensions in between the temporal dimension and the spatial dimensions, i.e. a variable's dimensions may be time, ..., lat, lon (geographic CRS), or time, ..., y, x (non-geographic CRS) – both in exactly this order, where the ellipsis ... refers to such extra dimensions. If there is a requirement to store individual time stamps per pixel value, this could still be done by adding a variable to the dataset which would then be e.g. pixel_time with dimensions time, lat, and lon.

10.7 Encoding of Units

All variables that represent quantities must set the attributeunits according to the CF Convention on Units. Even if the quantity is dimensionless, the units attribute must be set to indicate its dimensionless-ness by using the value "1".

10.8 Encoding of Flags

For data variables that are encoded as flags using named bits or bit masks/ranges, we strictly follow the CF Convention on Flags.

10.9 Encoding of Missing Values

According to the CF Convention on Missing Data, missing data should be indicated by the variable attribute _FillValue. However, Zarr does not define or interpret (decode) array attributes at all. The Zarr equivalent of the CF attribute _FillValue is the array property fill_value (not an attribute). fill_value can and should be set for all data types including integers, also because it is given in raw units, that is, before scaling_factor and add_offset are applied (by xarray). Zarr's fill_value has the advantage that data array chunks comprising only fill_value values can and should be dropped entirely. This can reduce number of chunks dramatically and improve data access performance a lot for many no-data chunks. In our case we should use fill_value to indicate that a data cube's grid cell does not contain a value at all, it does not exist, it does not make sense, it had no input, etc.

However, when reading a Zarr with Python xarray using decode_cf=True (the new default), xarray will also encode variable attribute _FillValue and missing_value and mask the data accordingly. It will unfortunately ignore valid_min, valid_max, and valid_range.

Missing values shall therefore be indicated by the Zarr array property fill_value. In addition valid_min, valid_max, valid_range variable attributes may be given, but they will not be decoded. However, applications might still find them useful, e.g. xcube Viewer uses them, if provided as value range for mapping color bars.

10.10 Encoding of Scale/Offset Values

Scale and offset encoding of bands / variables follows the CF Convention on Packed Data. In practice, affected variables must have attributes scaling_factor (default is 1) and add_offset (with default 0).

10.11 Multi-Resolution Datasets

Spatial multi-resolution datasets are described in detail in a dedicated document xcube Multi-Resolution Datasets.

10.12 Multi-Band Datasets

Individual spectral bands of a dataset should be stored as variables. For example, this is the case for multi-band products, like Sentinel-2, where each wavelength will be represented by a separate variable named B01, B02, etc.

10.13 Metadata Consolidation

Datasets using Zarr format should provide consolidated dataset metadata in their root directories. This allows reading the metadata of datasets with many variables more efficiently, especially when data is stored in Object Storage and every metadata file would need to fetched via an individual HTTP request.

The consolidated metadata file should be named .zmetadata, which is also the default name used by the Zarr format.

The format of the consolidated metadata must be JSON. It is a JSON object using the following structure:

```
{
    "zarr_consolidated_format": 1,
    "metadata": {
        ".zattrs": {...},
        ".zgroup": {...},
        "band_1/.zarray": {...},
        "band_2/.zarray": {...},
        "band_2/.zattrs": {...},
        ...},
    }
}
```

10.13.1 Zip Archives

Zarr datasets may also be provided as Zip archives. Such Zip archives should contain the contents of the archived Zarr directory in their root. In other words, the keys of the entries in such an archive should not have a common prefix as is often the case when directories are zipped for convenience. The name of a zipped Zarr dataset should be the original Zarr dataset name plus the .zip extension. For example:

Desired:

```
${dataset-name}.zarr.zip/
   .zarray
   .zgroup
   .zmetadata
   band_1/
   time/
   ...
```

Zipping the Zarr datasets this way allows opening the Zarr datasets directly from the Zip, e.g., for xarray this is xarray. open_zarr("./dataset.zarr.zip").

10.13.2 Dataset Examples

TODO (forman)

- Global WGS-84
- Projected CRS
- Satellite Viewing Geometry
- Multi-resolution

CHAPTER

ELEVEN

COMMON DATA STORE CONVENTIONS

This document is a work in progress.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

Useful references related to this document include:

- The JSON Schema Specification and the book Understanding JSON Schema
- xcube Issue #330 ('Establish common data store conventions')
- The existing xcube store plugins xcube-sh, xcube-cci, and xcube-cds
- The xcube.util.jsonschema source code

11.1 Naming Identifiers

This section explains various identifiers used by the xcube data store framework and defines their format.

In the data store framework, identifiers are used to denote data sources, data stores, and data accessors. Data store, data opener, and data writer identifiers are used to register the component as extension in a package's plugin.py. Identifiers MUST be unambiguous in the scope of the data store. They SHOULD be unambiguous across the entirety of data stores.

There are no further restrictions for data source and data store identifiers.

A data accessor identifier MUST correspond to the following scheme:

```
<data_type>:<format>:<storage>[:<version>]
```

<data_type> identifies the in-memory data type to represent the data, e.g., dataset (or xarray.Dataset), geodataframe (or geopandas.GeoDataFrame). <format> identifies the data format that may be accessed, e.g., zarr, netcdf, geojson. <storage> identifies the kind of storage or data provision the accessor can access. Example values are file (the local file system), s3 (AWS S3-compatible object storage), or sentinelhub (the Sentinel Hub API), or cciodp (the ESA CCI Open Data Portal API). The <version> finally is an optional notifier about a data accessor's version. The version MUST follow the Semantic Versioning.

Examples for valid data accessors identifiers are:

- dataset:netcdf:file
- dataset:zarr:sentinelhub
- geodataframe:geojson:file
- geodataframe:shapefile:cciodp:0.4.1

11.2 Open Parameters

This section aims to provide an overview of the interface defined by an xcube data store or opener in its open parameters schema, and how this schema may be used by a UI generator to automatically construct a user interface for a data opener.

11.2.1 Specification of open parameters

Every implementation of the xcube.core.store.DataOpener or xcube.core.store.DataStore abstract base classes MUST implement the get_open_data_params_schema method in order to provide a description of the allowed arguments to open_data for each dataset supported by the DataOpener or DataStore. The description is provided as a JsonObjectSchema object corresponding to a JSON Schema. The intention is that this description should be full and detailed enough to allow the automatic construction of a user interface for access to the available datasets. Note that, under this system:

- 1. Every dataset provided by an opener can support a different set of open parameters.
- 2. The schema does not allow the representation of interdependencies between values of open parameters within a dataset. For instance, the following interdependencies between two open parameters *sensor_type* and *variables* would not be representable in an open parameters schema:

sensor_type: A or B*variables*: [temperature, humidity] for sensor type A; [temperature, pressure] for sensor type B

To work around some of the restrictions of point (2) above, a dataset MAY be presented by the opener as multiple "virtual" datasets with different parameter schemas. For instance, the hypothetical dataset described above MAY be offered not as a single dataset envdata but as two datasets envdata:sensor-a (with a fixed *sensor_type* of A) and envdata:sensor-b, (with a fixed *sensor_type* of B), offering different sets of permitted variables. Sometimes, the interdependencies between parameters are too complex to be fully represented by splitting datasets in this manner. In these cases:

- 1. The JSON Schema SHOULD describe the smallest possible superset of the allowed parameter combinations.
- 2. The additional restrictions on parameter combinations MUST be clearly documented.
- 3. If illegal parameter combinations are supplied, the opener MUST raise an exception with an informative error message, and the user interface SHOULD present this message clearly to the user.

11.2.2 Common parameters

While an opener is free to define any open parameters for any of its datasets, there are some common parameters which are likely to be used by the majority of datasets. Furthermore, there are some parameters which are fundamental for the description of a dataset and therefore MUST be included in a schema (these parameters are denoted explicitly in the list below). In case that an opener does not support varying values of one of these parameters, a constant value must defined. This may be achieved by the JSON schema's const property or by an enum property value whose is a one-element array.

Any dataset requiring the specification of these parameters MUST use the standard parameter names, syntax, and semantics defined below, in order to keep the interface consistent. For instance, if a dataset allows a time aggregation period to be specified, it MUST use the time_period parameter with the format described below rather than some other alternative name and/or format. Below, the parameters are described with their Python type annotations.

- variable_names: List[str]A list of the identifiers of the requested variables. This parameter MUST be included in an opener parameters schema.
- bbox: Union[str,Tuple[float, float, float, float]] The bounding box for the requested data, in the order xmin, ymin, xmax, ymax. Must be given in the units of the specified spatial coordinate reference system crs. This parameter MUST be included in an opener parameters schema.

- crs: strThe identifier for the spatial coordinate reference system of geographic data.
- spatial_res: floatThe requested spatial resolution (x and y) of the returned data. Must be given in the units of the specified spatial coordinate reference system crs. This parameter MUST be included in an opener parameters schema.
- time_range: Tuple[Optional[str], Optional[str]]The requested time range for the data to be returned. The first member of the tuple is the start time; the second is the end time. See section '*Date, time, and duration specifications*'. This parameter MUST be included in an opener parameters schema. If a date without a time is given as the start time, it is interpreted as 00:00 on the specified date. If a date without a time is given as the end time, it is interpreted as 24:00 on the specified date (identical with 00:00 on the date following the specified date). If the end time is specified as None, it is interpreted as the current time.
- time_period: strThe requested temporal aggregation period for the data. See section '*Date, time, and duration specifications*'. This parameter MUST be included in an opener parameters schema.
- force_cube: boolWhether to return results as a specification-compliant xcube. If a store supports this parameter and if a dataset is opened with this parameter set to True, the store MUST return a specification-compliant xcube dataset. If this parameter is not supported or if a dataset is opened with this parameter set to False, the caller MUST NOT assume that the returned data conform to the xcube specification.

11.2.3 Semantics of list-valued parameters

The variables parameter takes as its value a list, with no duplicated members and the values of its members drawn from a predefined set. The values of this parameter, and other parameters whose values also follow such a format, are interpreted by xcube as a *restriction*, much like a bounding box or time range. That is:

- By default (if the parameter is omitted or if a None value is supplied for it), *all* the possible member values MUST be included in the list. In the case of variables, this will result in a dataset containing all the available variables.
- If a list containing *some* of the possible members is given, a dataset corresponding to those members only MUST be returned. In the case of variables, this will result in a dataset containing only the requested variables.
- A special case of the above: if an empty list is supplied, a dataset containing *no data* MUST be returned but with the requested spatial and temporal dimensions.

11.2.4 Date, time, and duration specifications

In the common parameter time_range, times can be specified using the standard JSON Schema formats date-time or date. Any additional time or date parameters supported by an xcube opener dataset SHOULD also use these formats, unless there is some good reason to prefer a different format.

The formats are described in the JSON Schema Validation 2019 draft, which adopts definitions from RFC 3339 Section 5.6. The JSON Schema date-time format corresponds to RFC 3339's date-time production, and JSON Schema's date format to RFC 3339's full-date production. These formats are subsets of the widely adopted ISO 8601 format.

The date format corresponds to the pattern YYYY-MM-DD (four-digit year - month - day), for example 1995-08-20. The date-time format consists of a date (in the date format), a time (in HH:MM:SS format), and timezone (Z for UTC, or +HH:MM or -HH:MM format). The date and time are separated by the letter T. Examples of date-time format include 1961-03-23T12:22:45Z and 2018-04-01T21:12:00+08:00. Fractions of a second MAY also be included, but are unlikely to be relevant for xcube openers.

The format for durations, as used for aggregation period, does **not** conform to the syntax defined for this purpose in the ISO 8601 standard (which is also quoted as Appendix A of RFC 3339). Instead, the required format is a small subset of the pandas time series frequency syntax, defined by the following regular expression:

^([1-9][0-9]*)?[HDWMY]\$

That is: an optional positive integer followed by one of the letters H (hour), D (day), W (week), M (month), and Y (year). The letter specifies the time unit and the integer specifies the number of units. If the integer is omitted, 1 is assumed.

11.2.5 Time limits: an extension to the JSON Schema

JSON Schema itself does not offer a way to impose time limits on a string schema with the date or date-time format. This is a problem for xcube generator UI creation, since it might be reasonably expected that a UI will show and enforce such limits. The xcube opener API therefore defines an unofficial extension to the JSON string schema: a JsonStringSchema object (as returned as part of a JsonSchema by a call to get_open_data_params_schema) MAY, if it has a format property with a value of date or date-time, also have one or both of the properties min_datetime and max_datetime. These properties must also conform to the date or date-time format. xcube provides a dedicated JsonDatetimeSchema for this purpose. Internally, it extends JsonStringSchema by adding the required properties to the JSON string schema.

11.2.6 Generating a UI from a schema

With the addition of the time limits extension described above, the JSON Schema returned by get_open_data_params_schema is expected to be extensive and detailed enough to fully describe a UI for cube generation.

Order of properties in a schema

Sub-elements of a JsonObjectSchema are passed to the constructor using the properties parameter with type signature Mapping[str, JsonSchema]. Openers SHOULD provide an ordered mapping as the value of properties, with the elements placed in an order suitable for presentation in a UI, and UI generators SHOULD lay out the UI in the provided order, with the exception of the common parameters discussed below. Note that the CPython dict object preserves the insertion order of its elements as of Python 3.6, and that this behaviour is officially guaranteed as of Python 3.7, so additional classes like OrderedDict are no longer necessary to fulfil this requirement.

Special handling of common parameters

Any of the common parameters listed above SHOULD, if present, be recognized and handled specially. They SHOULD be presented in a consistent position (e.g. at the top of the page for a web GUI), in a consistent order, and with user-friendly labels and tooltips even if the title and description annotations (see below) are absent. The UI generator MAY provide special representations for these parameters, for instance an interactive map for the bbox parameter.

An opener MAY provide title, description, and/or examples annotations for any of the common parameters, and a UI generator MAY choose to use any of these to supplement or modify its standard presentation of the common parameters.

Schema annotations (title, description, examples, and default)

For JSON Schemas describing parameters other than the common parameters, an opener SHOULD provide the title and description annotations. A UI generator SHOULD make use of these annotations, for example by taking the label for a UI control from title and the tooltip from description. The opener and UI generator MAY additionally make use of the examples annotation to record and display example values for a parameter. If a sensible default value can be envisaged, the opener SHOULD record this default as the value of the default annotation and the UI generator SHOULD set the default value in the UI accordingly. If the title annotation is absent, the UI generator SHOULD use the key corresponding to the parameter's schema in the parent schema as a fallback.

Generalized conversion of parameter schemas

For parameters other than the common parameters, the UI can be generated automatically from the schema structure. In the case of a GUI, a one-to-one conversion of values of JSON Schema properties into GUI elements will generally be fairly straightforward. For instance:

- A schema of type boolean can be represented as a checkbox.
- A schema of type string without restrictions on allowed items can be represented as an editable text field.
- A schema of type string with an enum keyword giving a list of allowed values can be represented as a drop-down menu.
- A schema of type string with the keyword setting "format": "date" can be represented as a specialized date selector.
- A schema of type array with the keyword setting "uniqueItems": true and an items keyword giving a fixed list of allowed values can be represented as a list of checkboxes.

CHAPTER

TWELVE

XCUBE DEVELOPER GUIDE

Version 0.2, draft

IMPORTANT NOTE: Any changes to this doc must be reviewed by dev-team through pull requests.

12.1 Table of Contents

- Versioning
- Coding Style
- Main Packages
 - Package xcube.core
 - Package xcube.cli
 - Package xcube.webapi
 - Package xcube.util
- Development Process

12.2 Versioning

We adhere to PEP-440. Therefore, the xcube software version uses the format <major>.<micro> for released versions and <major>.<micro>.dev<n> for versions in development.

- <major> is increased for major enhancements. CLI / API changes may introduce incompatibilities with former version.
- <minor> is increased for new features and and minor enhancements. CLI / API changes are backward compatible with former version.
- <micro> is increased for bug fixes and micro enhancements. CLI / API changes are backward compatible with former version.
- <n> is increased whenever the team (internally) deploys new builds of a development snapshot.

The current software version is in xcube/version.py.

12.3 Coding Style

We follow PEP-8, including its recommendation of PEP-484 syntax for type hints.

12.3.1 Updating code style in the existing codebase

A significant portion of the existing codebase does not adhere to our current code style guidelines. It is of course a goal to bring these parts into conformance with the style guide, but major style changes should not be bundled into pull requests focused on other improvements or bug fixes, because they obscure the significant code changes and make reviews difficult. Large-scale style and formatting updates should instead be made via dedicated pull requests.

12.3.2 Line length

As recommended in PEP-8, all lines should be limited to a maximum of 79 characters, including docstrings and comments.

12.3.3 Quotation marks for string literals

In general, single quotation marks should always be used for string literals. Double quotation marks should only be used if there is a compelling reason to do so in a particular case.

12.4 Main Packages

- xcube.core Hosts core API functions. Code in here should be maintained w.r.t. backward compatibility. Therefore think twice before adding new or change existing core API.
- xcube.cli Hosts CLI commands. CLI command implementations should be lightweight. Move implementation code either into core or util.CLI commands must be maintained w.r.t. backward compatibility. Therefore think twice before adding new or change existing CLI commands.
- xcube.webapi Hosts Web API functions. Web API command implementations should be lightweight. Move implementation code either into core or util.Web API interface must be maintained w.r.t. backward compatibility. Therefore think twice before adding new or change existing web API.
- xcube.util Mainly implementation helpers. Comprises classes and functions that are used by cli, core, webapi in order to maximize modularisation and testability but to minimize code duplication. The code in here must not be dependent on any of cli, core, webapi. The code in here may change often and in any way as desired by code implementing the cli, core, webapi packages.

The following sections will guide you through extending or changing the main packages that form xcube's public interface.

12.4.1 Package xcube.cli

Checklist

Make sure your change

- 1. is covered by unit-tests (package test/cli);
- 2. is reflected by the CLI's doc-strings and tools documentation (currently in README.md);
- 3. follows existing xcube CLI conventions;
- 4. follows our *code style* conventions;
- 5. is reflected in API and WebAPI, if desired;
- 6. is reflected in CHANGES.md.

Hints

Make sure your new CLI command is in line with the others commands regarding command name, option names, as well as metavar arguments names. The CLI command name shall ideally be a verb.

Avoid introducing new option arguments if similar options are already in use for existing commands.

In the following common arguments and options are listed.

Input argument:

```
@click.argument('input')
```

If input argument is restricted to an xcube dataset:

```
@click.argument('cube')
```

Output (directory) option:

Output format:

Output parameters:

Variable names:

For parsing CLI inputs, use helper functions that are already in use. In the CLI command implementation code, raise click.ClickException(message) with a clear message for users.

Common xcube CLI options like -f for FORMAT should be lower case letters and specific xcube CLI options like -S for SIZE in xcube gen are recommended to be uppercase letters.

Extensively validate CLI inputs to avoid that API functions raise ValueError, TypeError, etc. Such errors and their message texts are usually hard to understand by users. They are actually dedicated to to developers, not CLI users.

There is a global option --traceback flag that user can set to dump stack traces. You don't need to print stack traces from your code.

12.4.2 Package xcube.core

Checklist

Make sure your change

- 1. is covered by unit-tests (package test/core);
- 2. is covered by API documentation;
- 3. follows existing xcube API conventions;
- 4. follows our *code style* conventions;
- 5. is reflected in xarray extension class xcube.core.xarray.DatasetAccessor;
- 6. is reflected in CLI and WebAPI if desired;
- 7. is reflected in CHANGES.md.

Hints

Create new module in xcube.core and add your functions. For any functions added make sure naming is in line with other API. Add clear doc-string to the new API. Use Sphinx RST format.

Decide if your API methods requires *xcube datasets* as inputs, if so, name the primary dataset argument cube and add a keyword parameter cube_asserted: bool = False. Otherwise name the primary dataset argument dataset.

Reflect the fact, that a certain API method or function operates only on datasets that conform with the xcube dataset specifications by using cube in its name rather than dataset. For example compute_dataset can operate on any xarray datasets, while get_cube_values_for_points expects a xcube dataset as input or read_cube ensures it will return valid xcube datasets only.

In the implementation, if not cube_asserted, we must assert and verify the cube is a cube. Pass True to cube_asserted argument of other API called later on:

```
from xcube.core.verify import assert_cube

def frombosify_cube(cube: xr.Dataset, ..., cube_asserted: bool = False):
    if not cube_asserted:
        assert_cube(cube)
    ...
    result = bibosify_cube(cube, ..., cube_asserted=True)
    ...
```

If import xcube.core.xarray is imported in client code, any xarray.Dataset object will have an extra property xcube whose interface is defined by the class xcube.core.xarray.DatasetAccessor. This class is an xarray extension that is used to reflect xcube.core functions and make it directly applicable to the xarray.Dataset object.

Therefore any xcube API shall be reflected in this extension class.

12.4.3 Package xcube.webapi

Checklist

Make sure your change

- 1. is covered by unit-tests (package test/webapi);
- 2. is covered by Web API specification and documentation (currently in webapi/res/openapi.yml);
- 3. follows existing xcube Web API conventions;
- 4. follows our *code style* conventions;
- 5. is reflected in CLI and API, if desired;
- 6. is reflected in CHANGES.md.

12.4.4 Hints

- The Web API is defined in webapi. app which defines mapping from resource URLs to handlers
- All handlers are implemented in webapi.handlers. Handler code just delegates to dedicated controllers.
- All controllers are implemented in webapi.controllers.*. They might further delegate into core.*

12.5 Development Process

- 1. Make sure there is an issue ticket for your code change work item.
- 2. Select issue; priorities are as follows:
 - 1. "urgent" and ("important" and "bug")
 - 2. "urgent" and ("important" or "bug")
 - 3. "urgent"
 - 4. "important" and "bug"
 - 5. "important" or "bug"
 - 6. others
- 3. Make sure issue is assigned to you; if unclear, agree with team first.
- 4. Add issue label "in progress".
- 5. Clone the GitHub repository to your local machine, if you haven't already done so.
- 6. Check out the master branch and make sure it's up to date (git pull).
- 7. Create a development branch named "<developer>-<issue>-<title>" (see below).
- 8. Develop, having in mind the checklists and implementation hints above.
 - 1. In your first commit, refer the issue so it will appear as link in the issue history.
 - 2. It is useful to push your first commit to the GitHub repository at once and create a draft pull request for your branch, to make it easier to find online. Also, it will run continuous integration and check the compatibility of your branch with the main branch.
 - 3. Develop, test, and push to the remote branch as desired.

- 4. In your last commit, utilize checklists above. (You can include the line "closes #<issue>" in your commit message to auto-close the issue once the PR is merged.)
- 9. When your branch is ready to merge (as determined by the checklist), either create a pull request, or, if you had created a draft in the previous step, remove its draft status and invite one or more reviewers. Exactly who should review depends on how potentially dangerous your changes are, who's available, who knows that part of the codebase well, etc. if unsure, discuss with the team. You can also mention other team members with the @username syntax in a PR comment to make sure that they're aware of the PR even if they don't need to review it.
- 10. Wait for reviewers to review your PR and respond to comments and suggestions. Reviewers can also commit to your branch with additional changes.
- 11. Once all reviewers have given an "accept" review, the PR should be merged. The last reviewer to give an "accept" should perform the merge; if they forget, the PR author should merge.
- 12. Remove issue label "in progress".
- 13. Delete the feature branch.
- 14. If the PR completely solves the associated issue: close the issue, if it wasn't already automatically closed by a keyword in the pull request description, and make sure that there is a link from the issue to the PR (e.g. by adding a "fixed by PR #123" comment on the issue).
- 15. If the PR is only partly solving an issue:
 - 1. Make sure the issue contains a to-do list (checkboxes) to complete the issue.
 - 2. Do not include the line "closes #<issue>" in your last commit message.
 - 3. Add "relates to issue#" in PR.
 - 4. Make sure to check the corresponding to-do items (checkboxes) after the PR is merged.
 - 5. Remove issue label "in progress".
 - 6. Leave issue open.

12.6 Branches and Releases

12.6.1 Target Branch

The master branch contains latest developments, including new features and fixes. Its software version string is always <major>.<miror>.dev<n>. The branch is used to generate major, minor, or maintenance releases. That is, either <major>, <minor>, or <fix> is increased. Before a release, the last thing we do is to remove the .dev<n> suffix, after a release, the first thing we do is to increase the micro version and add the .dev<n> suffix.

12.6.2 Development Branches

Development branches should be named <developer>-<issue>-<title> where

- <developer> is the github name of the code author
- <issue> is the number of the issue in the github issue tracker that is targeted by the work on this branch
- <title> is either the name of the issue or an abbreviated version of it

12.7 Release Process

12.7.1 Release on GitHub

This describes the release process for xcube. For a plugin release, you need to adjust the paths accordingly.

- Check issues in progress, close any open issues that have been fixed.
- Make sure that all unit tests pass and that test coverage is 100% (or as near to 100% as practicable).
- In xcube/version.py remove the .dev suffix from version name.
- Adjust version in Dockerfile accordingly.
- Make sure CHANGES.md is complete. Remove the suffix (in development) from the last version headline.
- Push changes to either master or a new maintenance branch (see above).
- Await results from Travis CI and ReadTheDocs builds. If broken, fix.
- Go to xcube/releases and press button "Draft a new Release".
 - Tag version is: v\${version} (with a "v" prefix)
 - Release title is: \${version} (without a "v" prefix)
 - Paste latest changes from CHANGES.md into field "Describe this release"
 - Press "Publish release" button
- After the release on GitHub, rebase sources, if the branch was master, create a new maintenance branch (see above)
- In xcube/version.py increase version number and append a .dev0 suffix to the version name so that it is still PEP-440 compatible.
- Adjust version in Dockerfile accordingly.
- In CHANGES.md add a new version headline and attach (in development) to it.
- Push changes to either master or a new maintenance branch (see above).
- Activate new doc version on ReadTheDocs.

Go through the same procedure for all xcube plugin packages dependent on this version of xcube.

12.7.2 Release on Conda-Forge

These instructions are based on the documentation at conda-forge.

Conda-forge packages are produced from a github feedstock repository belonging to the conda-forge organization. A repository's feedstock is usually located at https://github.com/conda-forge/<repo-name>-feedstock, e.g., https://github.com/conda-forge/xcube-feedstock.

Usually, a conda-forge bot will create a feedstock pull request automatically for each new GitHub release, and the maintainers only need to merge it (potentially after some manual improvements). The manual procedure below is only needed if there isn't time to wait for the bot, or if an additional conda-forge build has to be done without a new GitHub release. The package is updated by

- · forking the repository
- creating a new branch for the changes

• creating a pull request to merge this branch into conda-forge's feedstock repository (this is done automatically if the build number is 0).

In detail, the steps are:

- 1. Fork https://github.com/conda-forge/<repo-name>-feedstock in your personal GitHub account.
- 2. Clone the repository locally and create a new branch. The name of the branch is not strictly prescribed, but it's sensible to choose an informative name like update_0_5_3.
- 3. In case the build number is 0, a bot will render the feedstock during the pull request. Otherwise, conduct the following steps: Rerender the feedstock using conda-smithy. This updates common conda-forge feedstock files. It's probably easiest to install conda-smithy in a fresh environment for this:

```
conda install -c conda-forge conda-smithy
```

```
conda smithy rerender -c auto
```

- 4. Update recipe/meta.yaml for the new version. Mainly this will involve the following steps:
 - 1. Update the value of the version variable (or, if the version number has not changed, increment the build number).
 - 2. If the version number has changed, ensure that the build number is set to 0.
 - 3. Update the sha256 hash of the source archive prepared by GitHub.
 - 4. If the dependencies have changed, update the list of dependencies in the -run subsection to match those in the environment.yml file.
- 5. Commit the changes and push them to GitHub. A pull request at the feedstock repository on conda-forge will be automatically created by a bot if the build number is 0. If it is higher, you will have to create the pull request yourself.
- 6. Once conda-forge's automated checks have passed, merge the pull request.
- 7. Delete your personal fork of the xcube-feedstock-repository.

Once the pull request has been merged, the updated package should usually become available from conda-forge within a couple of hours.

TODO: Describe deployment of xcube Docker image after release

If any changes apply to **xcube** serve and the xcube Web API:

Make sure changes are reflected in xcube/webapi/res/openapi.yml. If there are changes, sync xcube/webapi/res/openapi.yml with xcube Web API docs on SwaggerHub.

Check if changes affect the xcube-viewer code. If so make sure changes are reflected in xcube-viewer code and test viewer with latest xcube Web API. Then release a new xcube viewer.

12.7.3 xcube Viewer

- Cd into viewer project directory (.../xcube-viewer/.).
- Remove the -dev suffix from version property in package.json.
- Remove the -dev suffix from version constant in src/config.ts.
- Make sure CHANGES.md is complete. Remove the suffix (in development) from the last version headline.
- Build the app and test the build using a local http-server, e.g.:
 - \$ npm install -g http-server \$ cd build \$ http-server -p 3000 -c-1
- Push changes to either master or a new maintenance branch (see above).

- Goto xcube-viewer/releases and press button "Draft a new Release".
 - Tag version is: v\${version} (with a "v" prefix).
 - Release title is: \${version}.
 - Paste latest changes from CHANGES.md into field "Describe this release".
 - Press "Publish release" button.
- After the release on GitHub, if the branch was master, create a new maintenance branch (see above).
- Increase version property and version constant in package.json and src/config.ts and append -dev.0 suffix to version name so it is SemVer compatible.
- In CHANGES.md add a new version headline and attach (in development) to it.
- Push changes to either master or a new maintenance branch (see above).
- Deploy builds of master branches to related web content providers.

CHAPTER

THIRTEEN

PLUGINS

xcube's functionality can be extended by plugins. A plugin contributes extensions to specific extension points defined by xcube. Plugins are detected and dynamically loaded, once the available extensions need to be inquired.

13.1 Installing Plugins

Plugins are installed by simply installing the plugin's package into xcube's Python environment.

In order to be detected by xcube, an plugin package's name must either start with xcube_ or the plugin package's setup.py file must specify an entry point in the group xcube_plugins. Details are provided below in section *plu-gin_development*.

13.2 Available Plugins

13.2.1 SENTINEL Hub

The xcube_sh plugin adds support for the SENTINEL Hub Cloud API. It extends xcube by a new Python API function xcube_sh.cube.open_cube to create data cubes from SENTINEL Hub on-the-fly. It also adds a new CLI command xcube sh gen to generate and write data cubes created from SENTINEL Hub into the file system.

13.2.2 ESA CCI Open Data Portal

The xcube_cci plugin provides support for the ESA CCI Open Data Portal.

13.2.3 Copernicus Climate Data Store

The xcube_cds plugin provides support for the Copernicus Climate Data Store.

13.2.4 Cube Generation

xcube's GitHub organisation currently hosts a few plugins that add new *input processor* extensions (see below) to xcube's data cube generation tool *xcube gen*. They are very specific but are a good starting point for developing your own input processors:

- xcube_gen_bc adds new input processors for specific Ocean Colour Earth Observation products derived from the Sentinel-3 OLCI measurements.
- xcube_gen_rbins adds new input processors for specific Ocean Colour Earth Observation products derived from the SEVIRI measurements.
- xcube_gen_vito adds new input processors for specific Ocean Colour Earth Observation products derived from the Sentinel-2 MSI measurements.

13.3 Plugin Development

13.3.1 Plugin Definition

An xcube plugin is a Python package that is installed in xcube's Python environment. xcube can detect plugins either

- 1. by naming convention (more simple);
- 2. by entry point (more flexible).

By naming convention: Any Python package named xcube_<name> that defines a plugin *initializer function* named init_plugin either defined in xcube_<name>/plugin.py (preferred) or xcube_<name>/__init__.py is an xcube plugin.

By entry point: Any Python package installed using Setuptools that defines a non-empty entry point group xcube_plugins is an xcube plugin. An entry point in the xcube_plugins group has the format <name> = <fully-qualified-module-path>:<init-func-name>, and therefore specifies where plugin *initializer function* named <init-func-name> is found. As an example, refer to the xcube standard plugin definitions in xcube's setup.py file.

For more information on Setuptools entry points refer to section Creating and discovering plugins in the Python Packing User Guide and Dynamic Discovery of Services and Plugins in the Setuptools documentation.

13.3.2 Initializer Function

xcube plugins are initialized using a dedicated function that has a single *extension registry* argument of type *xcube*. *util.extension.ExtensionRegistry*, that is used by plugins's to register their extensions to xcube. By convention, this function is called init_plugin, however, when using entry points, it can have any name. As an example, here is the initializer function of the SENTINEL Hub plugin xcube_sh/plugin.py::

13.3.3 Extension Points and Extensions

When a plugin is loaded, it adds its extensions to predefined *extension points* defined by xcube. xcube defines the following extension points:

- xcube.core.gen.iproc: input processor extensions
- xcube.core.dsio: dataset I/O extensions
- xcube.cli: Command-line interface (CLI) extensions

An extension is added to the extension registry's add_extension method. The extension registry is passed to the plugin initializer function as its only argument.

13.3.4 Input Processor Extensions

Input processors are used the xcube gen CLI command and gen_cube API function. An input processor is responsible for processing individual time slices after they have been opened from their sources and before they are appended to or inserted into the data cube to be generated. New input processors are usually programmed to support the characteristics of specific xcube gen inputs, mostly specific Earth Observation data products.

By default, xcube uses a standard input processor named default that expects inputs to be individual NetCDF files that conform to the CF-convention. Every file is expected to contain a single spatial image with dimensions lat and lon and the time is expected to be given as global attributes.

If your input files do not conform with the default expectations, you can extend xcube and write your own input processor. An input processor is an implementation of the xcube.core.gen.iproc.InputProcessor or xcube. core.gen.iproc.XYInputProcessor class.

As an example take a look at the implementation of the default input processor xcube.core.gen.iproc.DefaultInputProcessor or the various input processor plugins mentioned above.

The extension point identifier is defined by the constant xcube.constants. EXTENSION_POINT_INPUT_PROCESSORS.

13.3.5 Dataset I/O Extensions

More coming soon...

The extension point identifier is defined by the constant xcube.constants.EXTENSION_POINT_DATASET_IOS.

13.3.6 CLI Extensions

CLI extensions enhance the xcube command-line tool by new sub-commands. The xcube CLI is implemented using the click library, therefore the extension components must be click commands or command groups.

The extension point identifier is defined by the constant xcube.constants.EXTENSION_POINT_CLI_COMMANDS.

CHAPTER

FOURTEEN

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

in

(class

Α

ComputedMultiLevelDataset add_array() (xcube.core.zarrstore.GenericZarrStore con method), 85 $\texttt{add_extension()} \ (\textit{xcube.util.extension.ExtensionRegistry}^{\texttt{COO}}$ crs method), 97 Cub affine_transform_dataset() (in module xcube.core.resampling), 71 D (xcube.core.mldataset.MultiLevelDataset apply() Data method), 82 Array (*xcube.core.zarrstore.GenericZarrStore attribute*), Data Data 85 Data assert_cube() (in module xcube.core.verify), 80 avg_resolutions(xcube.core.mldataset.MultiLevelDatasedat property), 81 Data

В

<pre>base_dataset (xcube.core.mldatase</pre>	et.MultiLevel	Dataset
property), 81		
BaseMultiLevelDataset	(class	in
xcube.core.mldataset), 82		

С

CachedZarrStore (<i>class in xcube.co</i> , chunk_dataset() (<i>in module xcube.</i>		<i>,</i> .
chunks (xcube.core.schema.CubeSche	ema prope	riy), 95
<pre>clip_dataset_by_geometry()</pre>	(in	module
xcube.core.geom), 76		
<pre>close() (xcube.core.mldataset.Combi </pre>	inedMulti	LevelDatas
method), 82		
<pre>close() (xcube.core.mldataset.Lazy</pre>	vMultiLev	elDataset
method), 84		
<pre>close() (xcube.core.mldataset.Mappe</pre>	edMultiLe	evelDataset
method), 84		
close() (xcube.core.mldataset	.MultiLev	elDataset
<i>method</i>), 81		
close() (xcube.core.zarrstore	e.Generic.	ZarrStore
method), 85		
CombinedMultiLevelDataset	(class	in
xcube.core.mldataset), 82		
component (xcube.util.extension.Exten	nsion prop	verty), 98
<pre>compute_cube() (in module xcube.com</pre>	pre.compi	ıte), 66

xcube.core.mldataset), 83
<pre>vert_geometry() (in module xcube.core.geom), 94</pre>
rds (xcube.core.schema.CubeSchema property), 95
(xcube.core.gridmapping.GridMapping property), 90
eSchema (class in xcube.core.schema), 94
aDescriptor (class in xcube.core.store), 61
aOpener (class in xcube.core.store), 59
aSearcher (class in xcube.core.store), 60
asetDescriptor (<i>class in xcube.core.store</i>), 61, 62
asets (xcube.core.mldataset.MultiLevelDataset
property), 81
aStore (class in xcube.core.store), 53

DataStoreError (class in xcube.core.store), 61

DataWriter (class in xcube.core.store), 60

- delete_data() (xcube.core.store.MutableDataStore method), 58
- deregister_data() (xcube.core.store.MutableDataStore method), 58
- derive() (xcube.core.gridmapping.GridMapping method), 88
- derive_tiling_scheme() (xcube.core.mldataset.MultiLevelDataset method), 82
- describe_data() (xcube.core.store.DataStore method), 54

DiagnosticZarrStore (class in xcube.core.zarrstore), 88

- dims (xcube.core.schema.CubeSchema property), 95
- ds id (xcube.core.mldataset.LazyMultiLevelDataset property), 83
- ds_id (xcube.core.mldataset.MultiLevelDataset property), 81

F

- module encode_grid_mapping() (in xcube.core.resampling), 71
- evaluate_dataset() (in module xcube.core.evaluate), 67

Extension (class in xcube.util.extension), 98

EXTENSION_POINT_CLI_COMMANDS (in modu xcube.constants), 99	le g
EXTENSION_POINT_DATASET_IOS (in modu xcube.constants), 99	le g
EXTENSION_POINT_INPUT_PROCESSORS (in modu xcube.constants), 99	le g
ExtensionRegistry (class in xcube.util.extension), 96	6
F	ç
finalize() (xcube.core.zarrstore.GenericArra method), 87	ay g
<pre>find_components() (xcube.util.extension.ExtensionRe</pre>	egist
<pre>find_data_store_extensions() (in modu</pre>	le 9
<pre>find_extensions() (xcube.util.extension.ExtensionRe</pre>	egist
<pre>from_coords() (xcube.core.gridmapping.GridMappin</pre>	ng 🤉
<pre>from_dataset() (xcube.core.gridmapping.GridMappi</pre>	ing ç
<pre>from_dataset() (xcube.core.zarrstore.GenericZarrSta</pre>	ore g
FsMultiLevelDataset (class in xcube.core.mldataset 83	t), g
0	

G

<pre>gen_cube() (in module xcube.core.gen.gen), 64 GenericArray (class in xcube.core.zarrstore),</pre>	
GenericZarrStore (<i>class in xcube.core.zarrst</i>	ore), 85
GeoDataFrameDescriptor (<i>class in xcube.cd</i>	
63	
get() (xcube.core.zarrstore.ZarrStoreHolder 84	method),
<pre>get_component() (xcube.util.extension.Extens</pre>	ionRegistry
<pre>get_cube_point_indexes() (in</pre>	module
xcube.core.extract), 68	
<pre>get_cube_values_for_indexes() (in</pre>	module
xcube.core.extract), 69	
<pre>get_cube_values_for_points() (in</pre>	module
xcube.core.extract), 67	
<pre>get_data_ids() (xcube.core.store.DataStore</pre>	method),
53	
<pre>get_data_opener_ids() (xcube.core.store.l</pre>	DataStore
<pre>get_data_store_class() (in</pre>	module
xcube.core.store), 52	
<pre>get_data_store_params_schema() (in</pre>	module
xcube.core.store), 52	
get_data_store_params_schema()	
(xcube.core.store.DataStore class	method),
53	

get_data_types() (xcube.core.store.DataStore class method), 53 get_data_types_for_data() (xcube.core.store.DataStore method), 53 get_data_writer_ids() (xcube.core.store.MutableDataStore method), 57 get_dataset() (xcube.core.mldataset.LazyMultiLevelDataset method), 83 get_dataset() (xcube.core.mldataset.MultiLevelDataset method), 81 get_dataset_indexes() (in module xcube.core.extract), 69 get_extension() (xcube.util.extension.ExtensionRegistry method), 96 get_extension_registry() (in module xcube.util.plugin), 99 get_level_for_resolution() (xcube.core.mldataset.MultiLevelDataset method), 82 get_mask_sets() (xcube.core.maskset.MaskSet class method), 77 get_open_data_params_schema() (xcube.core.store.DataOpener method), 59 get_open_data_params_schema() (xcube.core.store.DataStore method), 55 get_plugins() (in module xcube.util.plugin), 99 get_schema() (xcube.core.store.DataDescriptor class method), 61 get_schema() (xcube.core.store.DatasetDescriptor class method), 62, 63 get_schema() (xcube.core.store.GeoDataFrameDescriptor class method), 63 get_schema() (xcube.core.store.MultiLevelDatasetDescriptor class method), 62 get_schema() (xcube.core.store.VariableDescriptor class method), 63 get_search_params_schema() (xcube.core.store.DataSearcher class method), 60 get_time_series() (in module xcube.core.timeseries), 70 get_write_data_params_schema() (xcube.core.store.DataWriter method), 60 get_write_data_params_schema() (xcube.core.store.MutableDataStore method), 57 grid_mapping(xcube.core.mldataset.LazyMultiLevelDataset property), 83 grid_mapping (xcube.core.mldataset.MultiLevelDataset property), 81 GridMapping (class in xcube.core.gridmapping), 88

Н

Н		lock (xcube.core.mldataset.LazyMultil	LevelDataset prop-
has_dat	a() (xcube.core.store.DataStore method), 54	<i>erty</i>), 83	
has_ext	<pre>sension() (xcube.util.extension.ExtensionRegistr method), 96</pre>		
height	(xcube.core.gridmapping.GridMapping prop- erty), 89	<pre>MappedMultiLevelDataset</pre>	(class in
I		<pre>mask_dataset_by_geometry()</pre>	(in module
Identit	xyMultiLevelDataset (class in	MaskSet (class in xcube.core.maskset)	
	xcube.core.mldataset), 83	<pre>metadata (xcube.util.extension.Extens MultiLevelDataset (class in xcube.class)</pre>	
1]_bbox	(<i>xcube.core.gridmapping.GridMapping prop-</i> <i>erty</i>), 91	MultiLevelDatasetDescriptor	(class in
ii bbox	c_from_xy_bbox()	xcube.core.store), 62	× ·
5	(xcube.core.gridmapping.GridMapping	MutableDataStore (class in xcube.co	pre.store), 57
	method), 91	Ν	
ij_bbox	xes (xcube.core.gridmapping.GridMapping property), 91	name (xcube.util.extension.Extension p	roperty) 98
ii bhox	<pre>ces_from_xy_bboxes()</pre>	ndim (xcube.core.schema.CubeSchema	
1)_0001	(xcube.core.gridmapping.GridMapping	ndim (xcube.core.store.VariableDescri	
	method), 91	<pre>new() (xcube.core.schema.CubeSchem</pre>	na class method),
ij_to_x	xy_transform(xcube.core.gridmapping.GridMap		1 1) 05
	property), 90	<pre>new_cluster() (in module xcube.util new_cube() (in module xcube.core.ne</pre>	
ij_tran	<pre>usform_from() (xcube.core.gridmapping.GridMapping</pre>	new_data_store() (in module xcube.core.ne	
	(xcube.core.griamapping.oriamapping method), 91	new_fs_data_store() (in module xcusc)	
ij_tran	usform_to() (xcube.core.gridmapping.GridMapp		
-	method), 90	property), 83	
<pre>import_</pre>	<pre>component() (in module xcube.util.extension), 99</pre>	num_levels (xcube.core.mldataset. property), 81	MultiLevelDataset
is_clos	se() (xcube.core.gridmapping.GridMapping method), 94	0	
	<pre>kis_up (xcube.core.gridmapping.GridMapping property), 90</pre>	<pre>open_data() (xcube.core.store.DataC open_data() (xcube.core.store.DataS</pre>	tore method), 56
is_lazy is_lon_		optimize_dataset() (in module xci 75	ıbe.core.optimize),
ic more	property), 90	Р	
is_regu	<pre>ilar (xcube.core.gridmapping.GridMapping property), 90</pre>	<pre>point (xcube.util.extension.Extension)</pre>	property), 98
is_tile	ed (xcube.core.gridmapping.GridMapping prop- erty), 89	R	
is_writ	<pre>seable() (xcube.core.zarrstore.GenericZarrStore method), 85</pre>	<pre>rasterize_features() (in module</pre>	xcube.core.geom),
K		rectify_dataset() (in value core resempting) 72	module
keys()	(xcube.core.zarrstore.DiagnosticZarrStore method), 88	xcube.core.resampling), 72 register_data() (xcube.core.store. method), 58	MutableDataStore
	тетои), во	regular() (xcube.core.gridmapping.	GridMapping class
L		method), 92	_
LazyMul	tiLevelDataset (class in xcube.core.mldataset), 83	<pre>remove_extension() (xcube.util.exte method), 98</pre>	ension.ExtensionRegistr

- list_data_ids() (xcube.core.store.DataStore method), rename() 54
- listdir() (xcube.core.zarrstore.GenericZarrStore resample_in_space() method), 85

xcube.core.resampling), 73

method), 85

(xcube.core.zarrstore.GenericZarrStore

(in

module

resample_in_time() (in module	U
xcube.core.resampling), 74 resample_ndimage() (in module	<pre>unchunk_dataset() (in module xcube.core.unchunk), 75</pre>
<pre>xcube.core.resampling), 71 reset() (xcube.core.zarrstore.ZarrStoreHolder</pre>	update_dataset_attrs() (in module
method), 85	xcube.core.update),79 update_dataset_spatial_attrs() (in module
resolutions (xcube.core.mldataset.MultiLevelDataset property), 81	<pre>xcube.core.update), 79 update_dataset_temporal_attrs() (in module</pre>
<pre>rmdir() (xcube.core.zarrstore.GenericZarrStore method), 85</pre>	xcube.core.update), 79
S	V
<pre>scale() (xcube.core.gridmapping.GridMapping</pre>	VariableDescriptor (<i>class in xcube.core.store</i>), 63
method), 88	<pre>vars_to_dim() (in module xcube.core.vars2dim), 75 verify_cube() (in module xcube.core.verify), 80</pre>
<pre>search_data() (xcube.core.store.DataSearcher method), 60</pre>	W
<pre>select_variables_subset() (in module</pre>	width (xcube.core.gridmapping.GridMapping property),
<pre>set() (xcube.core.zarrstore.ZarrStoreHolder method),</pre>	<pre>89 write_data() (xcube.core.store.DataWriter method),</pre>
set_dataset() (xcube.core.mldataset.LazyMultiLevelDa	60 taset write_data() (xcube.core.store.MutableDataStore
method), 84 shape (xcube.core.schema.CubeSchema property), 95	method), 57
size (xcube.core.gridmapping.GridMapping property), 89	Х
<pre>size_weights(xcube.core.mldataset.FsMultiLevelDataset</pre>	_{ct} x_coords (xcube.core.gridmapping.GridMapping prop- erty), 89
property), 83	x_dim (xcube.core.schema.CubeSchema property), 95
Т	x_max (xcube.core.gridmapping.GridMapping property),
<pre>tile_height (xcube.core.gridmapping.GridMapping</pre>	90 x_min (<i>xcube.core.gridmapping.GridMapping property</i>), 89
<pre>tile_size (xcube.core.gridmapping.GridMapping</pre>	x_name (xcube.core.schema.CubeSchema property), 95
tile_width (xcube.core.gridmapping.GridMapping property), 89	x_res (<i>xcube.core.gridmapping.GridMapping property</i>), 90
time_dim (xcube.core.schema.CubeSchema property),	<pre>x_size (xcube.core.schema.CubeSchema property), 95</pre> <pre>x_var (xcube.core.schema.CubeSchema property), 95</pre>
95 time_name (xcube.core.schema.CubeSchema property),	xy_bbox (xcube.core.gridmapping.GridMapping prop-
95	erty), 89 xy_bboxes (xcube.core.gridmapping.GridMapping
time_size (xcube.core.schema.CubeSchema property), 95	property), 91
time_var (xcube.core.schema.CubeSchema property), 95	xy_coords (xcube.core.gridmapping.GridMapping property), 89
to_coords() (xcube.core.gridmapping.GridMapping method), 92	xy_coords_chunks (xcube.core.gridmapping.GridMapping property), 89
to_dataset_attrs() (xcube.core.gridmapping.GridMap	xy_dim_names (xcube.core.gridmapping.GridMapping property), 89
method), 92 to_dict() (xcube.util.extension.Extension method), 98	xy_res (xcube.core.gridmapping.GridMapping prop-
to_dict() (xcube.util.extension.ExtensionRegistry	<pre>erty), 90 xy_to_ij_transform(xcube.core.gridmapping.GridMapping</pre>
<i>method</i>), 98 to_regular() (<i>xcube.core.gridmapping.GridMapping</i>	property), 90
method), 93	xy_var_names (xcube.core.gridmapping.GridMapping property), 89
transform() (xcube.core.gridmapping.GridMapping method), 92	

Y

y_coords (xcube.core.gridmapping.GridMapping property), 89 y_dim (xcube.core.schema.CubeSchema property), 95 y_max (xcube.core.gridmapping.GridMapping property), 90 y_min (xcube.core.gridmapping.GridMapping property), 90 y_name (xcube.core.schema.CubeSchema property), 95 y_res (xcube.core.gridmapping.GridMapping property), 90 y_size (xcube.core.schema.CubeSchema property), 95 y_var (xcube.core.schema.CubeSchema property), 95

Ζ

ZarrStoreHolder (class in xcube.core.zarrstore), 84