xcube

Release 0.3.0

Brockmann Consult GmbH

Nov 21, 2019

GETTING STARTED

> **Warning:** This documentation is a work in progress and currently less than a draft.

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

CHAPTER

ONE

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.

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 multi-dimensional, 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 Specification*.

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 Specification* 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>
Dimensions:        (lat: 720, lon: 1440, time: 1)
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 ... 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
    title:                       Global SST & Sea Ice Anomaly, L4 OSTIA, 0.25 ...
```

(continues on next page)

(continued from previous page)

```
    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
    product_version:              2.0
    uuid:                         5c1665b7-06e8-499d-a281-857dcbfd07e2
    gds_version_id:               2.0
    netcdf_version_id:            3.6
    date_created:                 20170606T061737Z
    start_time:                   20170605T000000Z
    time_coverage_start:          20170605T000000Z
    stop_time:                    20170606T000000Z
    time_coverage_end:            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...
    standard_name_vocabulary:     NetCDF Climate and Forecast (CF) Metadata Con...
    westernmost_longitude:        0.0
    easternmost_longitude:        360.0
    southernmost_latitude:        -90.0
    northernmost_latitude:        90.0
    spatial_resolution:           0.25 degree
    geospatial_lat_units:         degrees_north
    geospatial_lat_resolution:    0.25 degree
    geospatial_lon_units:         degrees_east
    geospatial_lon_resolution:    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
```

(continues on next page)

(continued from previous page)

```
  * 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:
    long_name:      analysed sea surface temperature
    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:

```
$ xcube gen -o "your/output/path/demo_SST_xcube.zarr" -c examples/gen/config_files/
↪xcube_sst_demo_config.yml --sort examples/gen/data/*.nc
```

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       (lat, bnds) float64 dask.array<shape=(5632, 2), chunksize=(5632, 2)>
  * 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),␣
↪chunksize=(1, 704, 640)>
Attributes:
    acknowledgment:             Data Cube produced based on data provided by ...
    comment:
    contributor_name:
    contributor_role:
    creator_email:              info@brockmann-consult.de
    creator_name:               Brockmann Consult GmbH
    creator_url:                https://www.brockmann-consult.de
    date_modified:              2019-09-25T08:50:32.169031
    geospatial_lat_max:         62.666666666666664
    geospatial_lat_min:         48.0
    geospatial_lat_resolution:  0.002604166666666666
    geospatial_lat_units:       degrees_north
    geospatial_lon_max:         10.666666666666664
    geospatial_lon_min:         -16.0
    geospatial_lon_resolution:  0.0026041666666666665
    geospatial_lon_units:       degrees_east
    history:                    xcube/reproj-snap-nc
    id:                         demo-bc-sst-sns-l2c-v1
    institution:                Brockmann Consult GmbH
    keywords:
    license:                    terms and conditions of the DCS4COP data dist...
    naming_authority:           bc
    processing_level:           L2C
    project:                    xcube
    publisher_email:            info@brockmann-consult.de
    publisher_name:             Brockmann Consult GmbH
    publisher_url:              https://www.brockmann-consult.de
    references:                 https://dcs4cop.eu/
    source:                     CMEMS Global SST & Sea Ice Anomaly Data Cube
    standard_name_vocabulary:
    summary:
    time_coverage_end:          2017-06-08T00:00:00.000000000
```

(continues on next page)

(continued from previous page)

```
    time_coverage_start:        2017-06-05T00:00:00.000000000
    title:                      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     (time) datetime64[ns] 2017-06-05T12:00:00 ... 2017-06-07T12:00:00
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, check the various functions provided by xcube Web API.

- **Datasets:**
  - Get datasets
  - Get dataset details

  - Get dataset coordinates
- **Color bars:**
  - Get color bars
  - Get color bars (HTML)
- **WMTS:**
  - Get WMTS KVP Capabilities (XML)
  - Get WMTS KVP local tile (PNG)
  - Get WMTS KVP remote tile (PNG)
  - Get WMTS REST Capabilities (XML)
  - Get WMTS REST local tile (PNG)
  - Get WMTS REST remote tile (PNG)
- **Tiles**
  - Get tile (PNG)
  - Get tile grid for OpenLayers 4.x
  - Get tile grid for Cesium 1.x
  - Get legend for layer (PNG)
- **Time series service (preliminary & unstable, will likely change soon)**
  - Get time stamps per dataset
  - Get time series for single point
- **Places service (preliminary & unstable, will likely change soon>'_**
  - Get all features
  - Get all features of collection "inside-cube"
  - Get all features for dataset "local"
  - Get all features of collection "inside-cube" for dataset "local"

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 `--show` 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 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):
```

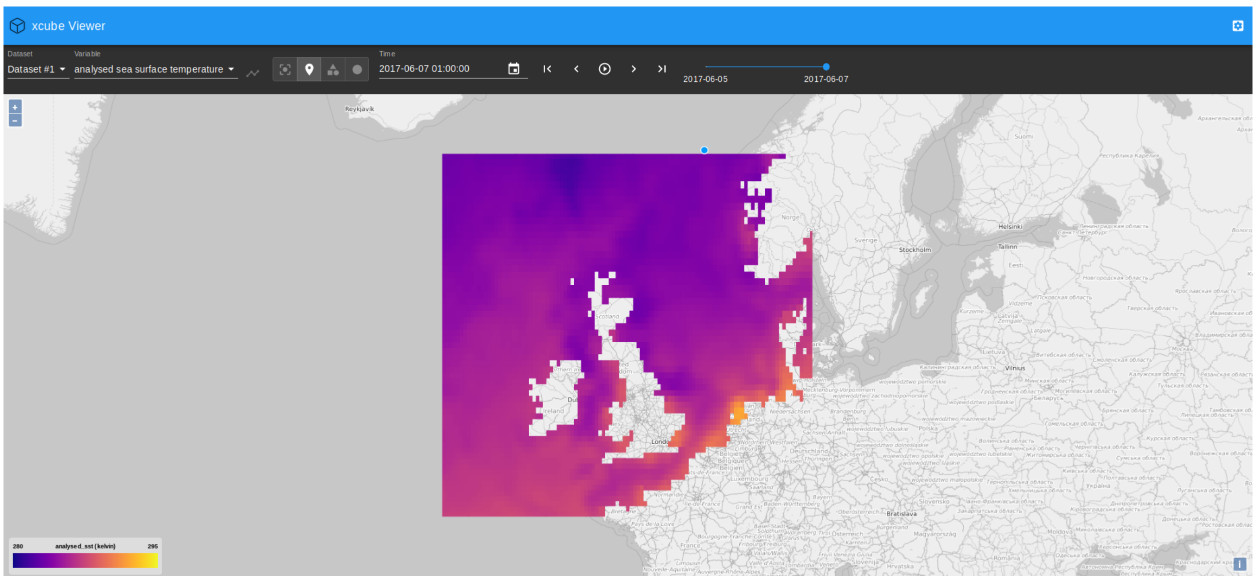
```
> SET XCUBE_VIEWER_PATH=/abs/path/to/xcube-viewer/build
```

4. Then run `xcube serve --show`:

```
$ xcube serve --show --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 --show --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 Installation using conda

Into existing conda environment (>= Python 3.7)

```
$ conda install -c conda-forge xcube
```

Into new conda environment

```
$ conda create -c conda-forge -n xcube python3
$ conda install -c conda-forge xcube
```

3.2 Installation from sources

First

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

Then

```
$ activate xcube
$ python setup.py develop
```

Update

```
$ activate xcube
$ git pull --force
$ python setup.py develop
```

Run tests

```
$ pytest
```

with coverage

```
$ pytest --cov=xcube
```

with coverage report in HTML

```
$ pytest --cov-report html --cov=xcube
```

3.3 Docker

To start a demo using docker use the following commands

```
$ docker build -t [your name] .
$ docker run -d -p [host port]:8000 [your name]
```

Example:

```
$  docker build -t xcube:0.1.0dev6 .
$  docker run -d -p 8001:8000 xcube:0.1.0dev6
$  docker ps
```

CHAPTER

FOUR

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
```

(continues on next page)

(continued from previous page)

```
                                 about input processors can be accessed by
                                 calling xcube gen --info . Defaults to
                                 "default", an input processor that can deal
                                 with simple datasets whose variables have
                                 dimensions ("lat", "lon") and conform with
                                 the CF conventions.
  -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.
  -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
                                 "<width>,<height>".
  -R, --region REGION            Output region using format "<lon-min>,<lat-
                                 min>,<lon-max>,<lat-max>"
  --variables, --vars VARIABLES  Variables to be included in output. Comma-
                                 separated list of names which may contain
                                 wildcard characters "*" and "?".
  --resampling␣
↪[Average|Bilinear|Cubic|CubicSpline|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 option --proc:
  default                          Single-scene NetCDF/CF inputs in xcube standard␣
↪format
```

(continues on next page)

(continued from previous page)

```
  rbins-seviri-highroc-scene-l2     RBINS SEVIRI HIGHROC single-scene Level-2 NetCDF␣
↪inputs
  rbins-seviri-highroc-daily-l2     RBINS SEVIRI HIGHROC daily Level-2 NetCDF inputs
  snap-olci-highroc-l2              SNAP Sentinel-3 OLCI HIGHROC Level-2 NetCDF inputs
  snap-olci-cyanoalert-l2           SNAP Sentinel-3 OLCI CyanoAlert Level-2 NetCDF␣
↪inputs
  vito-s2plus-l2                    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 option --format:
  csv                      (*.csv)       CSV file format
  mem                      (*.mem)       In-memory dataset I/O
  netcdf4                  (*.nc)        NetCDF-4 file format
  zarr                     (*.zarr)      Zarr file format (http://zarr.readthedocs.io)
```

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 INPUT. 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`.

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:

```
$ xcube gen -S 2000,1000 -R 0,50,5,52.5 --vars conc_chl,conc_tsm,kd489,c2rcc_flags,
↪quality_flags -o hiroc-cube.zarr /data/eo-data/SST/2018/**/*.nc
```

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)
      HEIGHT(L) = HEIGHT_0 * 2 ^ 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.
```

(continues on next page)

(continued from previous page)

```
Options:
  --help  Show this message and exit.

Commands:
  abox    Adjust a bounding box to a fixed Earth grid.
  levels  List levels for a resolution or a tile size.
  res     List resolutions close to a target resolution.
```

Example: Find suitable target resolution for a ~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_RES (%)
540     7       69120   384     0.0026041666666666665   289.9   -3.4
4140    4       66240   368     0.0027173913043478260   302.5   0.8
8100    3       64800   360     0.0027777777777777778   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       0.3333333333333333      37106.5
1       1080    6       0.16666666666666666     18553.2
2       2160    12      0.08333333333333333     9276.6
...
11      1105920 6144    0.00016276041666666666  18.1
12      2211840 12288   8.138020833333333e-05   9.1
13      4423680 24576   4.0690104166666664e-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
Orig. box WKT    = POLYGON ((0.0 50.0, 5.0 50.0, 5.0 52.5, 0.0 52.5, 0.0 50.0))
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 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
  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":
```

(continues on next page)

(continued from previous page)

```
    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.
  -p, --params PARAMS            Parameters passed as 'input_params' dict to
                                 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.
  --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
```

(continues on next page)

(continued from previous page)

```
                                 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".

  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 data points from an xcube dataset.

  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
```

(continues on next page)

(continued from previous page)

```
  of CUBE according to CHUNKS and writes the result to OUTPUT.

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"
  --help               Show this message and exit.
```

Example

```
$ xcube chunk input_not_chunked.zarr -o output_rechunked.zarr --chunks "time=1,
↪lat=270,lon=270"
```

Python API

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

4.6.2 `xcube edit`

Synopsis

Edit metadata of an xcube dataset.

```
$ xcube edit --help
```

```
Usage: xcube edit [OPTIONS] CUBE

  Edit the metadata of an xcube dataset. Edits the metadata of a given CUBE.
  The command currently works only for data cubes using ZARR format.

Options:
  -o, --output OUTPUT      Output path. The placeholder "{input}" will be
                           replaced by the input's filename without extension
                           (such as ".zarr"). Defaults to
                           "{input}-edited.zarr".
  -M, --metadata METADATA  The metadata of the cube is edited. The metadata to
                           be changed should be passed over in a single yml
                           file.
  -C, --coords             Update the metadata of the coordinates of the xcube
                           dataset.
  -I, --in-place           Edit the cube in place. Ignores output path.
  --help                   Show this message and exit.
```

Examples

The global attributes of the demo xcube dataset demo cube-1-250-250.zarr in the examples folder do not contain the creators name not an url. Furthermore the long name of the variable 'conc_chl' is 'Chlorophylll concentration', with too many l's. This can be fixed by using xcube edit. A yml-file defining the key words to be changed with the new content has to be created. The demo yml is saved in the examples folder.

Edit the metadata of the existing xcube dataset `cube-1-250-250-edited.zarr`:

```
$ xcube edit /examples/serve/demo/cube-1-250-250.zarr -M examples/edit/edit_metadata_
↪cube-1-250-250.yml -o cube-1-250-250-edited.zarr
```

The global attributes below, which are related to the xcube dataset coodrinates cannot be manually edited.

- `geospatial_lon_min`
- `geospatial_lon_max`
- `geospatial_lon_units`
- `geospatial_lon_resolution`
- `geospatial_lat_min`
- `geospatial_lat_max`
- `geospatial_lat_units`
- `geospatial_lat_resolution`
- `time_coverage_start`
- `time_coverage_end`

If you wish to update these attributes, you can use the commandline parameter `-C`:

> $ xcube edit /examples/serve/demo/cube-1-250-250.zarr -C -o cube-1-250-250-edited.zarr

The `-C` will update the coordinate attributes based on information derived directly from the cube.

Python API

The related Python API function is `xcube.core.edit.edit_metadata()`.

4.6.3 `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.

Options:
  -o, --output OUTPUT               Output path. If omitted, "INPUT.levels" will
                                    be used.
  -L, --link                        Link the INPUT instead of converting it to a
                                    level zero dataset. Use with care, as the
```

(continues on next page)

(continued from previous page)

```
                                  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.
  --help                          Show this message and exit.
```

Example

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

Python API

The related Python API function are

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

4.6.4 `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, 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
```

(continues on next page)

(continued from previous page)

```
                              input>" will be replaced by the input's filename
                              without extension (such as ".zarr"). Defaults to
                              "{input}-optimized.zarr".
  -I, --in-place              Optimize cube in place. Ignores output path.
  -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()`*.

4.6.5 `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] CUBE

  Delete empty chunks. Deletes all data files associated with empty (NaN-
  only) chunks in given CUBE, which must have ZARR format.

Options:
  --dry-run  Just read and process input, but don't produce any outputs.
  --help     Show this message and exit.
```

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.
  -o, --output OUTPUT             Output path. Defaults to 'out.zarr'.
  -f, --format [zarr|netcdf4|mem]
                                  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
                                  downsampling methods are 'count', 'first',
                                  'last', 'min', 'max', 'sum', 'prod', 'mean',
                                  'median', 'std', 'var', the upsampling
                                  methods are 'asfreq', 'ffill', 'bfill',
                                  '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
                                  'H', 'D', 'W', 'M', 'Q', 'Y'. Defaults to
                                  '1D'.
  -O, --offset TEXT               Offset used to adjust the resampled time
                                  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
                                  one of 'zero', 'slinear', 'quadratic',
                                  'cubic', 'linear', 'nearest', 'previous',
                                  '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
```

(continues on next page)

(continued from previous page)

```
                                  scipy.interpolate.interp1d(). Defaults to
                                  '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.
  --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:

```
$ xcube resample --vars conc_chl,conc_tsm -F 3D -M mean -M std -M count examples/
↪serve/demo/cube.nc
```

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 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".
  -D, --dim_name DIM-NAME     Name of the new dimension into variables.
                              Defaults to "var".
  -o, --output OUTPUT         Output path. If omitted, 'INPUT-vars2dim.INPUT-
                              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 publication

4.7.1 `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] [CUBE]...

  Serve data cubes via web service.

  Serves data cubes by a RESTful API and a OGC WMTS 1.0 RESTful and KVP
  interface. The RESTful API documentation can be found at
  https://app.swaggerhub.com/apis/bcdev/xcube-server.

Options:
  -A, --address ADDRESS  Service address. Defaults to 'localhost'.
  -P, --port PORT        Port number where the service will listen on.
                         Defaults to 8080.
  --prefix PREFIX        Service URL prefix. May contain template patterns
                         such as "${version}" or "${name}". For example
                         "${name}/api/${version}".
  -u, --update PERIOD    Service will update after given seconds of
                         inactivity. Zero or a negative value will disable
                         update checks. Defaults to 2.0.
  -S, --styles STYLES    Color mapping styles for variables. Used only, if one
                         or more CUBE arguments are provided and CONFIG is not
                         given. Comma-separated list with elements of the form
                         <var>=(<vmin>,<vmax>) or
                         <var>=(<vmin>,<vmax>,"<cmap>")
  -c, --config CONFIG    Use datasets configuration file CONFIG. Cannot be
                         used if CUBES are provided.
  --tilecache SIZE       In-memory tile cache size in bytes. Unit suffixes
                         'K', 'M', 'G' may be used. Defaults to '512M'. The
                         special value 'OFF' disables tile caching.
  --tilemode MODE        Tile computation mode. This is an internal option
                         used to switch between different tile computation
                         implementations. Defaults to 0.
  -s, --show             Run viewer app. Requires setting the environment
                         variable XCUBE_VIEWER_PATH to a valid xcube-viewer
                         deployment or build directory. Refer to
```

(continues on next page)

(continued from previous page)

```
                        https://github.com/dcs4cop/xcube-viewer for more
                        information.
  -v, --verbose         Delegate logging to the console (stderr).
  --traceperf           Print performance diagnostics (stdout).
  --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 at least the dimensions and shape (`time`, `lat`, `lon`), in exactly this order;
- that have 1D-coordinate variables corresponding to the dimensions;
- that have their spatial grid defined in the WGS84 (`EPSG:4326`) coordinate reference system.

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 in publicly accessible, AWS S3 compatible object storage (OBS).

As an example, here is the configuration of the demo server.

To increase imaging performance, xcube datasets can be converted to multi-resolution pyramids using the cli/xcube_level tool. In the configuration, the format must be set to `'level'`. Leveled xcube datasets are configured this way:

```
Datasets:

  - Identifier: my_multi_level_dataset
    Title: "My Multi-Level Dataset"
    FileSystem: local
    Path: my_multi_level_dataset.level
    Format: level

  - ...
```

To increase time-series extraction performance, xcube datasets my be rechunked with larger chunk size in the `time` dimension using the cli/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: local
    Path: my_dataset.zarr
    TimeSeriesDataset: my_dataset_opt_for_ts

  - Identifier: my_dataset_opt_for_ts
    Title: "My Dataset optimized for Time-Series"
    FileSystem: local
    Path: my_ts_opt_dataset.zarr
```

(continues on next page)

(continued from previous page)

```
    Format: zarr
    Hidden: True

  - ...
```

Example

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

```
xcube Server: WMTS, catalogue, data access, tile, feature, time-series services for
↪xarray-enabled data cubes, version 0.2.0
[I 190924 17:08:54 service:228] configuration file
↪'D:\\Projects\\xcube\\examples\\serve\\demo\\config.yml' successfully loaded
[I 190924 17:08:54 service:158] service running, listening on localhost:8080, try
↪http://localhost:8080/datasets
[I 190924 17:08:54 service:159] press CTRL+C to stop service
```

Web API

The xcube server has a dedicated Web API Documentation on SwaggerHub. It also lets you explore the API of existing xcube-servers.

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 Cube I/O

5.2 Cube generation

`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', drop_bounds=False, variables=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]
```

Parameters

- **`title`** – A title. Defaults to 'Test Cube'.
- **`width`** – Horizontal number of grid cells. Defaults to 360.
- **`height`** – Vertical number of grid cells. Defaults to 180.
- **`x_name`** – Name of the x coordinate variable. Defaults to 'lon'.
- **`y_name`** – Name of the y coordinate variable. Defaults to 'lat'.
- **`x_dtype`** – Data type of x coordinates. Defaults to 'float64'.
- **`y_dtype`** – Data type of y coordinates. Defaults to 'float64'.
- **`x_units`** – Units of the x coordinates. Defaults to 'degrees_east'.
- **`y_units`** – Units of the y coordinates. Defaults to 'degrees_north'.
- **`x_start`** – Minimum x value. Defaults to -180.
- **`y_start`** – Minimum y value. Defaults to -90.
- **`x_res`** – Spatial resolution in x-direction. Defaults to 1.0.

- **y_res** – Spatial resolution in y-direction. Defaults to 1.0.
- **inverse_y** – Whether to create an inverse y axis. Defaults to False.
- **time_name** – Name of the time coordinate variable. Defaults to 'time'.
- **time_periods** – Number of time steps. Defaults to 5.
- **time_freq** – Duration of each time step. Defaults to '1D'.
- **time_start** – First time value. Defaults to '2010-01-01T00:00:00'.
- **time_dtype** – Numpy data type for time coordinates. Defaults to 'datetime64[s]'.
- **time_units** – Units for time coordinates. Defaults to 'seconds since 1970-01-01T00:00:00'.
- **time_calendar** – Calender for time coordinates. Defaults to 'proleptic_gregorian'.
- **drop_bounds** – If True, coordinate bounds variables are not created. Defaults to False.
- **variables** – Dictionary of data variables to be added. None by default.

Returns A cube instance

5.3 Cube computation

5.4 Cube data extraction

5.5 Cube manipulation

xcube.core.unchunk.**unchunk_dataset**(*dataset_path: str*, *var_names: Sequence[str] = None*, *coords_only: bool = False*)

Unchunk dataset variables in-place.

Parameters

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

xcube.core.optimize.**optimize_dataset**(*input_path: str*, *output_path: str = None*, *in_place: bool = False*, *unchunk_coords: bool = False*, *exception_type: 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 set, 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.

Parameters

- **input_path** – Path to input dataset with ZARR format.

- **output_path** – 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 extentsion.
- **in_place** – Whether to modify the dataset in place. If False, a copy is made and *output_path* must be given.
- **unchunk_coords** – Whether to also consolidate coordinate chunk files.
- **exception_type** – Type of exception to be used on value errors.

5.6 Cube subsetting

xcube.core.select.**select_vars**(*dataset: xarray.Dataset, var_names: Collection[str] = None*) → xarray.Dataset

Select data variable from given *dataset* and create new dataset.

Parameters

- **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.

5.7 Cube masking

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

A set of mask variables derived from a variable *flag_var* with CF attributes "flag_masks" and "flag_meanings".

Each mask is represented by an *xarray.DataArray* and 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_masks" and "flag_meanings" are expected to exists and be valid.

classmethod get_mask_sets(*dataset: xarray.Dataset*) → Dict[str, xcube.core.maskset.MaskSet]

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

Parameters dataset – The dataset

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

5.8 Rasterisation of Features

5.9 Cube metadata

5.10 Cube verification

5.11 Multi-resolution pyramids

5.12 Utilities

class xcube.core.store.**CubeStore**(*dims: Sequence[str], shape: Sequence[int], chunks: Sequence[int], attrs: Dict[str, Any] = None, get_chunk: Callable[[CubeStore, str, Tuple[int, ...]], bytes] = None, trace_store_calls: bool = False*)

A Zarr Store that generates data cubes by allowing data variables to fetch or compute their chunks by a user-defined function *get_chunk*. Implements the standard Python `MutableMapping` interface.

This is how the *get_chunk* function is called::

```
data = get_chunk(cube_store, var_name, chunk_indexes)
```

where `cube_store` is this store, `var_name` is the name of the variable for which data is fetched, and `chunk_indexes` is a tuple of zero-based, integer chunk indexes. The result must be a Python *bytes* object.

Parameters

- **dims** – Dimension names of all data variables, e.g. ('time', 'lat', 'lon').
- **shape** – Shape of all data variables according to *dims*, e.g. (512, 720, 1480).
- **chunks** – Chunk sizes of all data variables according to *dims*, e.g. (128, 180, 180).
- **attrs** – Global dataset attributes.
- **get_chunk** – Default chunk fetching/computing function.
- **trace_store_calls** – Whether to print calls into the `MutableMapping` interface.

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

class xcube.core.schema.**CubeSchema**(*shape: Sequence[int], coords: Mapping[str, numpy.array], 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.

Parameters

- **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

Number of dimensions.

property dims
Tuple of dimension names.

property x_name
Name of the spatial x coordinate variable.

property y_name
Name of the spatial y coordinate variable.

property time_name
Name of the time coordinate variable.

property x_var
Spatial x coordinate variable.

property y_var
Spatial y coordinate variable.

property time_var
Time coordinate variable.

property x_dim
Name of the spatial x dimension.

property y_dim
Name of the spatial y dimension.

property time_dim
Name of the time dimension.

property shape
Tuple of dimension sizes.

property chunks
Tuple of dimension chunk sizes.

property coords
Dictionary of coordinate variables.

classmethod new(*cube: xarray.Dataset*) → xcube.core.schema.CubeSchema
Create a cube schema from given *cube*.

5.13 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.

Parameters

- **point** – extension point identifier
- **name** – extension name

Returns True, if extension exists

get_extension(*point: str*, *name: str*) → Optional[xcube.util.extension.Extension]
Get registered extension for given *point* and *name*.

Parameters

- **point** – extension point identifier
- **name** – extension name

Returns the extension or None, if no such exists

get_component (*point: str*, *name: str*) → Any

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

Parameters

- **point** – extension point identifier
- **name** – extension name

Returns extension component

find_extensions (*point: str, predicate: Callable[[Extension], bool] = None*) → List[xcube.util.extension.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** – extension point identifier
- **predicate** – optional filter function

Returns list of matching extensions

find_components (*point: str, predicate: Callable[[Extension], bool] = None*) → 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** – 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*) → xcube.util.extension.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.

Parameters

- **point** – extension point identifier
- **name** – extension name
- **component** – 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** – extension point identifier
- **name** – extension name

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
Whether this is a lazy extension that uses a loader.

property component
Extension component.

property point
Extension point identifier.

property name
Extension name.

property metadata
Extension metadata.

xcube.util.extension.**import_component** (*spec: str, transform: Callable[[Any, Extension], Any] = None, call: bool = False, call_args: Sequence[Any] = None, call_kwargs: Mapping[str, Any] = None*) → Callable[[xcube.util.extension.Extension], Any]
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** – 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** – 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

xcube.util.plugin.**get_extension_registry**()
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

SIX

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;
- Get places (GeoJSON features including vector data) that can be associated with xcube datasets.

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 in the in the xcube Web API reference.

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. When you open the page a message "cannot reach server" will appear. This is normal as the demo is configured to run with an xcube server started locally on default port 8080, see *Web API and Server*. Hence, you can either run an xcube server instance locally then reload the viewer page, or configure the viewer with an an existing xcube server. 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 are two demo servers that you may add for testing:

- DCS4COP Demo Server (`https://xcube2.dcs4cop.eu/dcs4cop-dev/api/0.1.0.dev6/`) providing ocean color variables in the North Sea area for the Data Cube Service for Copernicus (DCS4COP) EU project;
- ESDL Server (`https://xcube.earthsystemdatalab.net`) providing global essential climate variables (ECVs) variables for the ESA Earth System Data Lab.

7.2 Functionality

Coming soon…

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

XCUBE DATASET SPECIFICATION

This document provides a technical specification of the protocol and format for *xcube datasets*, data cubes in the xcube sense.

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.

8.1 Document Status

This is the latest version, which is still in development.

Version: 1.0, draft

Updated: 31.05.2018

8.2 Motivation

For many users of Earth observation data, multivariate coregistration, extraction, comparison, and analysis of different data sources is difficult, while data is provided in various formats and at different spatio-temporal resolutions.

8.3 High-level requirements

xcube datasets

- SHALL be time series of gridded, geo-spatial, geo-physical variables.
- SHALL use a common, equidistant, global or regional geo-spatial grid.
- SHALL shall be easy to read, write, process, generate.
- SHALL conform to the requirements of analysis ready data (ARD).
- SHALL be compatible with existing tools and APIs.
- SHALL conform to standards or common practices and follow a common data model.
- SHALL be formatted as self-contained datasets.
- SHALL be “cloud ready”, in the sense that subsets of the data can be accessed by individual URIs.

ARD links:

- http://ceos.org/ard/

- https://landsat.usgs.gov/ard
- https://medium.com/planet-stories/analysis-ready-data-defined-5694f6f48815

8.4 xcube Dataset Schemas

8.4.1 Basic Schema

- Attributes metadata convention
  - SHALL be CF >= 1.7
  - SHOULD adhere to Attribute Convention for Data Discovery
- Dimensions:
  - SHALL be at least `time`, `bnds`, and MAY be any others.
  - SHALL all be greater than zero, but `bnds` must always be two.
- Temporal coordinate variables:
  - SHALL provide time coordinates for given time index.
  - MAY be non-equidistant or equidistant.
  - `time[time]` SHALL provide observation or average time of *cell centers*.
  - `time_bnds[time, bnds]` SHALL provide observation or integration time of *cell boundaries*.
  - Attributes:
    * Temporal coordinate variables MUST have `units`, `standard_name`, and any others.
    * `standard_name` MUST be `"time"`, `units` MUST have format `"<deltatime> since <datetime>"`, where `datetime` must have ISO-format. `calendar` may be given, if not, `"gregorian"` is assumed.
- Spatial coordinate variables
  - SHALL provide spatial coordinates for given spatial index.
  - SHALL be equidistant in either angular or metric units
- Cube variables:
  - SHALL provide *cube cells* with the dimensions as index.
  - SHALL have shape
    * `[time, ..., lat, lon]` (see WGS84 schema) or
    * `[time, ..., y, x]` (see Generic schema)
  - MAY have extra dimensions, e.g. `layer` (of the atmosphere), `band` (of a spectrum).
  - SHALL specify the `units` metadata attribute.
  - SHOULD specify metadata attributes that are used to identify missing values, namely `_FillValue` and / or `valid_min`, `valid_max`, see notes in CF conventions on these attributes.
  - MAY specify metadata attributes that can be used to visualise the data:
    * `color_bar_name`: Name of a predefined colour mapping. The colour bar is applied between a minimum and a maximum value.

    * `color_value_min`, `color_value_max`: Minimum and maximum value for applying the colour bar. If not provided, minimum and maximum default to `valid_min`, `valid_max`. If neither are provided, minimum and maximum default to `0` and `1`.

8.4.2 WGS84 Schema (extends Basic)

- Dimensions:
  - SHALL be at least `time`, `lat`, `lon`, `bnds`, and MAY be any others.
- Spatial coordinate variables:
  - SHALL use WGS84 (EPSG:4326) CRS.
  - SHALL have `lat[lat]` that provides observation or average latitude of *cell centers* with attributes: `standard_name="latitude"` `units="degrees_north"`.
  - SHALL have `lon[lon]` that provides observation or average longitude of *cell centers* with attributes: `standard_name="longitude"` and `units="degrees_east"`.
  - SHOULD HAVE `lat_bnds[lat, bnds]`, `lon_bnds[lon, bnds]`: provide geodetic observation or integration coordinates of *cell boundaries*.
- Cube variables:
  - SHALL have shape `[time, ..., lat, lon]`.

8.4.3 Generic Schema (extends Basic)

- Dimensions: `time`, `y`, `x`, `bnds`, and any others.
  - SHALL be at least `time`, `y`, `x`, `bnds`, and MAY be any others.
- Spatial coordinate variables:
  - Any spatial grid and CRS.
  - `y[y]`, `x[x]`: provide spatial observation or average coordinates of *cell centers*.
    * Attributes: `standard_name`, `units`, other units describe the CRS / projections, see CF.
  - `y_bnds[y, bnds]`, `x_bnds[x, bnds]`: provide spatial observation or integration coordinates of *cell boundaries*.
  - MAY have `lat[y,x]`: latitude of *cell centers*.
    * Attributes: `standard_name="latitude"`, `units="degrees_north"`.
  - `lon[y,x]`: longitude of *cell centers*.
    * Attributes: `standard_name="longitude"`, `units="degrees_east"`.
- Cube variables:
  - MUST have shape `[time, ..., y, x]`.

8.5 xcube EO Processing Levels

This section provides an attempt to characterize xcube datasets generated from Earth Observation (EO) data according to their processing levels as they are commonly used in EO data processing.

8.5.1 Level-1C and Level-2C

- Generated from Level-1A, -1B, -2A, -2B EO data.
- Spatially resampled to common grid
  - Typically resampled at original resolution.
  - May be down-sampled: aggregation/integration.
  - May be upsampled: interpolation.
- No temporal aggregation/integration.
- Temporally non-equidistant.

8.5.2 Level-3

- Generated from Level-2C or -3 by temporal aggregation.
- No spatial processing.
- Temporally equidistant.
- Temporally integrated/aggregated.

CHAPTER

NINE

XCUBE DEVELOPER GUIDE

Version 0.2, draft

IMPORTANT NOTE: Any changes to this doc must be reviewed by dev-team through pull requests.

9.1 Preface

> *Gedacht ist nicht gesagt.Gesagt ist nicht gehört.Gehört ist nicht verstanden.Verstanden ist nicht einverstanden.Einverstanden ist nicht umgesetzt.Umgesetzt ist nicht beibehalten.*

by Konrad Lorenz (translation is left to the reader)

9.2 Table of Contents

9.3 Versioning

We adhere to PEP-440. Therefore, the xcube software version uses the format `<major>.<minor>.<micro>` for released versions and `<major>.<minor>.<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`.

9.4 Coding Style

We try adhering to PEP-8.

9.5 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.

9.5.1 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 PEP8 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:

```
@click.option('--output', '-o', metavar='OUTPUT',
              help='Output directory. If omitted, "INPUT.levels" will be used.')
```

Output format:

```
@click.option('--format', '-f', metavar='FORMAT', type=click.Choice(['zarr', 'netcdf
↪']),
              help="Format of the output. If not given, guessed from OUTPUT.")
```

Output parameters:

```
@click.option('--param', '-p', metavar='PARAM', multiple=True,
              help="Parameter specific for the output format. Multiple allowed.")
```

Variable names:

```
@click.option('--variable',--var', metavar='VARIABLE', multiple=True,
              help="Name of a variable. Multiple allowed.")
```

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.

9.5.2 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 PEP8 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.

9.5.3 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 PEP8 conventions;
5. is reflected in CLI and API, if desired;
6. is reflected in CHANGES.md.

9.5.4 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.*`

9.6 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. Create development branch named "–" or "—fix" (see below).
6. 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. Develop, test, and push to the remote branch as desired.
    3. In your last commit, utilize checklists above. (You can include the line "closes #" in your commit message to auto-close the issue once the PR is merged.)
7. Create PR if build servers succeed on your branch. If not, fix issue first.For the PR assign the team for review, agree who is to merge. Also reviewers should have checklist in mind.
8. Merge PR after all reviewers are accepted your change. Otherwise go back.
9. Remove issue label "in progress".
10. Delete the development branch.
11. 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 #" 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.

9.7 Branches and Releases

9.7.1 Target Branches

- The `master` branch contains latest developments, including new features and fixes. It is used to generate `<major>.<minor>.0` releases. That is, either `<major>` or `<minor>` is increased.
- The `<major>.<minor>.x` branch is the maintenance branch for a former release tagged `v<major>.<minor>.0`. It is used to generate maintenance `<major>.<minor>.<fix>` releases. That is, only `<fix>` is increased. Most changes to `<major>.<minor>.x` branch must obviously be merged into `master` branch too.

The software version string on all active branches is always `<major>.<minor>.<micro>.dev<n>`. Only for a release, we remove the `.dev<n>` suffix.

9.7.2 Development Branches

Development branches that target the `<major>.<minor>.x` branch should indicate that by using the suffix `-fix`, e.g. `coolguy-7633-div_by_zero_in_mean-fix`. After a pull request, the development branch will first be merged into the `<major>.<minor>.x` branch then into `master`.

9.8 Release Process

9.8.1 xcube

- Check issues in progress, close any open issues that have been fixed.
- In `xcube/version.py` remove the `.dev` suffix from version name.
- 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.
- Goto xcube/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 `CHANGELOG.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)
- In `xcube/version.py` increase version number and append a `.dev0` suffix to version name so that it is still PEP-440 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).
- Activate new doc version on ReadTheDocs.

Go through the same procedure for all xcube plugin packages dependent on this version of xcube.

TODO: Describe deployment to xcube conda package after release 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.

9.8.2 xcube Viewer

- Cd into viewer project directory (`.../xcube-viewer/.`).
- Remove the `-dev` suffix from `version` property in `package.json`.
- Remove the `-dev` suffix from `VIEWER_VERSION` constant in `src/config.ts`.
- Make sure `CHANGELOG.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
- 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 `CHANGELOG.md` into field “Describe this release”.
  – Press “Publish release” button.
- Deploy build contents to any relevant web content providers.
- After the release on GitHub, if the branch was `master`, create a new maintenance branch (see above).
- In `package.json` and `VIEWER_VERSION` constant in `src/config.ts` append `-dev.0` suffix . to version name so it is SemVer compatible.
- In `CHANGELOG.md` add a new version headline and attach `(in development)` to it.
- Push changes to either master or a new maintenance branch (see above).

CHAPTER

TEN

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.

10.1 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 *plugin_development*.

10.2 Available Plugins

10.2.1 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.

10.2.2 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.

10.3 Plugin Development

10.3.1 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.

10.3.2 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`::

```
from xcube.constants import EXTENSION_POINT_CLI_COMMANDS
from xcube.util import extension


def init_plugin(ext_registry: extension.ExtensionRegistry):
    """xcube SentinelHub extensions"""
    ext_registry.add_extension(loader=extension.import_component('xcube_sh.cli:cli'),
                               point=EXTENSION_POINT_CLI_COMMANDS,
                               name='sh_cli')
```

10.3.3 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.

10.3.4 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`.

10.3.5 Dataset I/O Extensions

More coming soon…

The extension point identifier is defined by the constant `xcube.constants.EXTENSION_POINT_DATASET_IOS`.

10.3.6 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 ELEVEN

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

S

T

U

X

Y