

# Euroargodev Cheat Sheet



Join the community at [github.com/euroargodev/argopy](https://github.com/euroargodev/argopy)

## Fetching Argo data

API

Import the data fetcher, select an access point (region, float or profile) and trigger data or index download:

### A basic example

```
from argopy import DataFetcher

fetcher = DataFetcher().region([-75, -45, 20, 30,
                                0, 100,
                                '2011-01',
                                '2011-06'])




fetcher = DataFetcher().float([6902746, 6902755])
fetcher = DataFetcher().profile(6902746, [1,12])

fetcher.to_xarray()
fetcher.to_dataframe()
fetcher.data
fetcher.index
```

## Select an user mode

API

**argopy** provides 3 user modes with different level of data post-processing:

-  **expert** mode: return all the Argo data, without any post-processing,
-  **standard** mode: simplifies the dataset, remove most of its jargon and return a priori good data,
-  **research** mode: simplifies the dataset to its heart, preserving only data of the highest quality for research studies.

*Details of the processing chain for each Argo parameters can be found in the documentation.*

### Select with global option setter:

```
argopy.set_options(mode='expert')
```

### Select in a temporary context:

```
with argopy.set_options(mode='expert'):
    DataFetcher().profile(6902746, 34)
```

### Select with fetcher options:

```
DataFetcher(mode='research').region([-75, -45,
                                         20, 30,
                                         0, 100])
```

## Select a data provider

API

**argopy** allows users to fetch Argo data from several sources:

- the **Ifremer erddap**. Updated daily, this database holds the complete dataset and is efficient for large requests
- a **GDAC server**. This could be the ftp or the https servers.
- your **local data** copy of the GDAC. Useful to work offline.
- the **Argovis** server. Updated daily, provides access to QC=1 data only

### Select with global option setter:

```
argopy.set_options(src='gdac',
                   gdac='https://...')
```

### Select in a temporary context:

```
with argopy.set_options(src='argovis'):
    DataFetcher().profile(6902746, 34)
```




### Select with fetcher options:

```
DataFetcher(src='erddap')
```

## Select a dataset

API

**argopy** provides 2 data sources for physical and biogeochemical parameters:

- The “**phy**” dataset provides data from floats that measure temperature, salinity, pressure, without limitation in depth. This dataset returns data from the  core &  deep missions.
- The “**bgc**” dataset provides data from floats that measure temperature, salinity, pressure and oxygen, pH, nitrate, chlorophyll, backscatter and irradiance, without limitation in depth. This dataset returns data from the  BGC mission.

### Select with global option setter:

```
argopy.set_options(ds='bgc')
```

### Select in a temporary context:

```
with argopy.set_options(ds='bgc'):
    DataFetcher().profile(6904241, 12)
```

### Select with fetcher options:

```
DataFetcher(ds='phy').float(6902746)
```

## Data manipulation

API

Use methods from the **argo** xarray accessor

### Transformation

#### # Points vs profiles

```
ds.argo.point2profile()
ds.argo.profile2point()
```

#### # Interpolation (pressure levels)

```
std = [0,100,200,500] # in db
ds.argo.interp_std_levels(std)
```

#### # Group-by pressure bins

```
b = np.arange(0., 2000., 250.0) # in db
ds.argo.groupby_pressure_bins(bins=b,
                              select='deep')
ds.argo.groupby_pressure_bins(bins=b,
                              select='random')
```

### Additional variables

Complete your dataset with additional variables using the TEOS-10

```
ds.argo.teos10(['SA', 'CT', 'CNDC'])
```

### Filters and Transformers

Filter measurements according to **QC flags** values:

```
ds.argo.filter_qc(QC_list=[1,2],
                  QC_fields='all')
ds.argo.filter_qc(QC_list=1,
                  QC_fields='PSAL')
```

Filter/transform a dataset according to **DATA\_MODE** parameter:

```
ds.argo.datamode.merge()
ds.argo.datamode.filter(dm=['D'], params='all')
```

Filter and transform variables according to the **OWC** salinity calibration software requirements:


```
ds.argo.filter_scalib_pres(force='default')
```

Filter and transform variables according to **research mode** requirements:




```
ds.argo.filter_researchmode()
```

### Snippet tags legend

Dataset:

 : core,  : deep,  : BGC

User mode:

 : expert,  : standard,  : research

Data selection:

 : region,  : float,  : profile

# Euroargodev Cheat Sheet



Join the community at [github.com/euroargodev/argopy](https://github.com/euroargodev/argopy)

## Argo meta data

API

### Index of profiles

```
from argopy import ArgoIndex
ArgoIndex(index_file='core') # 'bgc-s', 'bgc-b', 'aux'
ArgoIndex().N_RECORDS
ArgoIndex().convention_supported
```

```
box = [-60, -55, 40, 45, '2007-08', '2007-09']
ArgoIndex().search_tim(box)
ArgoIndex().search_lat_lon(box)
ArgoIndex().search_lat_lon_tim(box)
```

```
ArgoIndex().search_wmo([1901393, 6902755])
ArgoIndex().search_cyc(1)
ArgoIndex().search_wmo_cyc(1901393, 12)
ArgoIndex().read_wmo()
ArgoIndex().read_dac_wmo()
ArgoIndex().records_per_wmo()
```

```
ArgoIndex().N_MATCH
ArgoIndex().to_dataframe()
ArgoIndex().to_indexfile('myindex.csv')
```

### Reference tables

Based on NERC Vocabulary Server (NVS)  
Managed by the Argo Vocabulary Task Team (AVTT)

```
from argopy import ArgoNVSReferenceTables
ArgoNVSReferenceTables().tbl_name('R01')
ArgoNVSReferenceTables().tbl('R01')
ArgoNVSReferenceTables().all_tbl_name
ArgoNVSReferenceTables().all_tbl
ArgoNVSReferenceTables().search('sensor')
```

### ADMT Documentation

Access and discover all ADMT documentation

```
from argopy import ArgoDocs
ArgoDocs().list
ArgoDocs(35385)
ArgoDocs(35385).open_pdf(page=12)
ArgoDocs().search('CDOM')
```

### Deployment plan

Based on Ocean-OPS API, retrieve past and future plans

```
from argopy import OceanOPSDeployments
OceanOPSDeployments().to_dataframe()
OceanOPSDeployments([-90, 0,
                      0, 90]).to_dataframe()
OceanOPSDeployments().plot_status()
```

### GDAC snapshot with DOI

Access and discover all Argo GDAC snapshot DOI

```
from argopy import ArgoDOI
ArgoDOI() # last snapshot information
ArgoDOI().search('2020-02')
ArgoDOI().search('2020-02', network='BGC')
ArgoDOI('95141')
ArgoDOI(hashtag='95141')
```

## Argo-BGC specifics

API

### Set up argopy to work with BGC

Select the BGC dataset with the keyword **bgc** in several scopes, but note that

⚠ as of version 1.0.0,  
only **synthetic** BGC data are supported ⚠

#### Select with global option setter:

```
argopy.set_options(ds='bgc')
```

#### Select in a temporary context:

```
with argopy.set_options(ds='bgc'):
    DataFetcher().float(6904241)
```

#### Select with fetcher options:

```
DataFetcher(ds='bgc').profile(6902746, 12)
```

### Data fetcher with BGC

Select BGC parameters to be returned with the **params** argument:

```
# All parameters found in the access point will be returned:
DataFetcher(params='all') # (default if not specified)
```

```
# Only the DOXY variable will be returned:
DataFetcher(params='DOXY')
```

```
# Only DOXY and BBP700 will be returned:
DataFetcher(params=['DOXY', 'BBP700'])
```

Use the **measured** argument to force parameter(s) to have no NaNs:

```
# All parameters are allowed to have NaNs (not constrained):
DataFetcher(measured=None) # (default if not specified)
```

```
# All parameters won't have any NaNs (fully constrained):
DataFetcher(measured='all')
```

```
# Only DOXY won't have NaNs (partial constrain):
DataFetcher(measured='DOXY')
```

```
# Only DOXY and BBP700 won't have NaNs (partial constrain):
DataFetcher(measured=['DOXY', 'BBP700'])
```

### BGC profiles index

The **argopy** index store (see Argo meta data help) supports the **Bio**, **Synthetic** and **Auxiliary** Profile directory files.

```
from argopy import ArgoIndex
idx = ArgoIndex(index_file='bgc-b').load()
```

Use the specific **search\_params** to look for profiles with 1 or more parameter:

```
idx.search_params('DOXY')
idx.search_params(['DOXY', 'CDOM'])
idx.search_params(['DOXY', 'CDOM'], logical='or')
```

Use the specific **search\_parameter\_data\_mode** to look for profiles in specified data modes:

```
idx.search_parameter_data_mode({'BBP700': 'D'})
idx.search_parameter_data_mode({'DOXY': ['R', 'A']})
idx.search_parameter_data_mode({'BBP700': 'D',
                                'DOXY': 'D'},
                                logical='and')
```

From a Data or Index fetcher

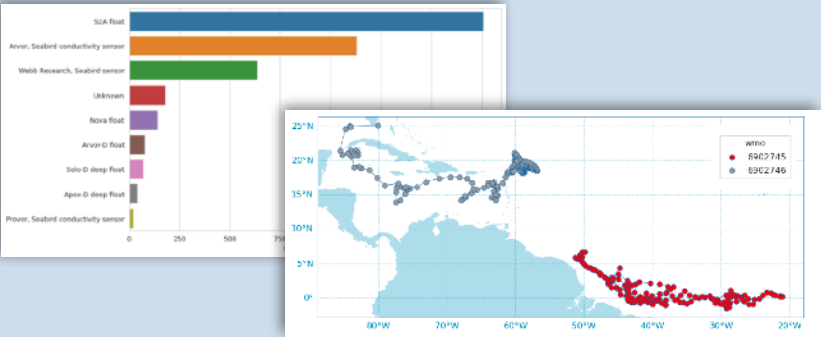
```
from argopy import DataFetcher
fetcher = DataFetcher()
fetcher.region([-75, -45, 20, 30, 0, 100,
                '2015-01', '2020-01']).load()
```

Trajectories

```
fetcher.plot()
fetcher.plot('trajectory')
```

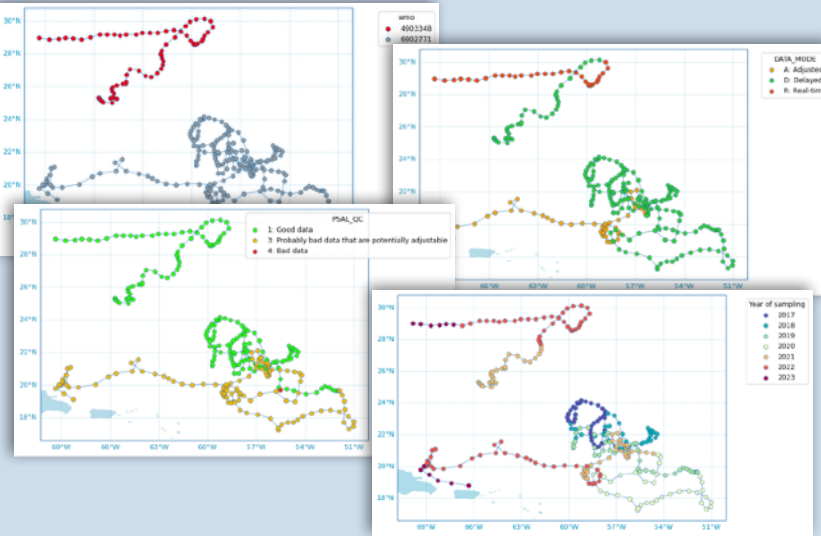
Histograms on properties

```
fetcher.plot('dac')
fetcher.plot('profiler')
```



Scatter maps from Datasets

```
from argopy.plot import scatter_map
scatter_map(ds)
scatter_map(ds, hue='DATA_MODE')
scatter_map(ds.isel(N_LEVELS=0), hue='PSAL_QC')
ds['year'] = ds['TIME.year'] # Add a variable
scatter_map(ds.isel(N_LEVELS=0),
            hue='year',
            cmap='Spectral_r',
            legend_title='Year of sampling')
```



Dashboards

For a collection of floats or profiles, get an easy and direct access to Euro-Argo, BGC, Ocean-Ops, Coriolis and Argovis dashboards

From a fetcher

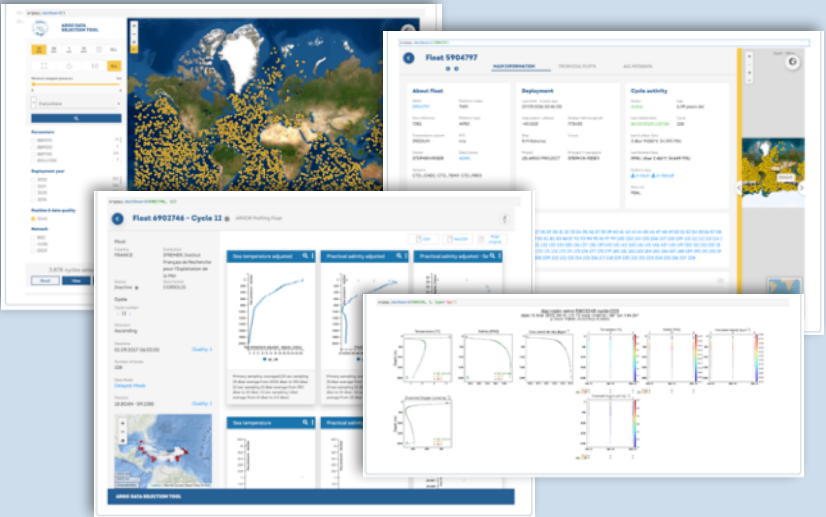
```
DataFetcher().float(6902746).dashboard()
```

or direct access

```
from argopy import dashboard
dashboard()
dashboard(6902746)
```

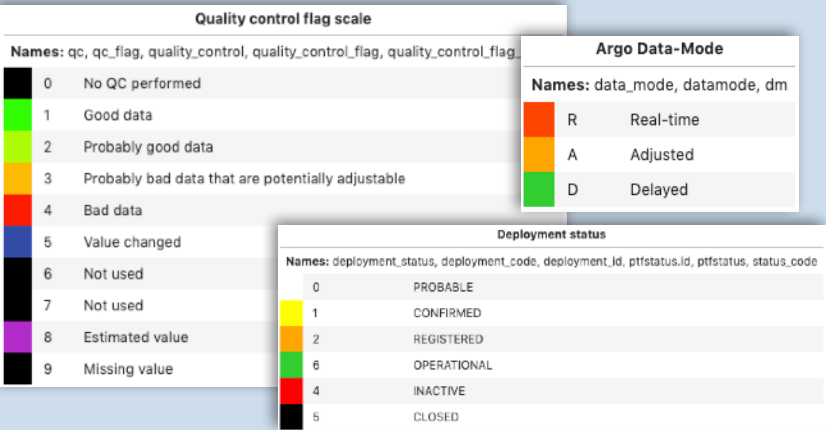
```
dashboard(6902746, 12)
dashboard(5903248, 3, type='bgc')
```

By default, this will insert the dashboard in a notebook cell, but it can also return the url to open in your browser.



Argo color palettes

```
from argopy.plot import ArgoColors
ArgoColors('data_mode')
ArgoColors('qc_flag')
ArgoColors('deployment_status')
```



Topography

Download a regional subset of the GEBCO 15'' topography

```
from argopy import TopoFetcher
ds = TopoFetcher([-65, -55, 10, 20],
                 cache=True).to_xarray()
```

CLS Altimetry tests

Easily checkout CLS altimetry test figures for one or more floats

```
from argopy import DataFetcher
fetcher.float([6902745,
               6902746])
fetcher.plot('qc_altimetry')
```

Data sources for OWC

Prepare Matlab data source files for the OWC analysis.

```
from argopy import DataFetcher
ds = DataFetcher(mode='expert')
    .float(6902766)
    .load().data
ds.argo.create_float_source('output_folder')
```

Reference data for core

Using the Ifremer erddap, [argopy](#) provides access to the core reference dataset from past Argo profiles as well as from ship-based CTD

Argo reference profiles

```
fetcher = DataFetcher(src='erddap', ds='ref')
fetcher.region([-65, -55, 10, 20,
                 0, 5000]).load()
```

ds = fetcher.data

Ship-based reference CTD profiles

```
from argopy import CTDRefDataFetcher
with argopy.set_options(user='jane_doe',
                        password='****'):
    fetcher = CTDRefDataFetcher([-65, -55,
                                10, 20,
                                0, 5000])

ref_ctd = fetcher.to_xarray()
```



Argopy Cheatsheet  
Copyright © 2024 Argopy Development Team  
Released under a EUPL-1.2 International License  
API documentation based on [argopy](#) release 1.0.0 (Oct. 2024)

Citation: Maze, G., & Balem, K. (2020). [argopy](#): A Python library for Argo ocean data analysis. *Journal of Open Source Software*, 5(53) [//doi.org/10.21105/joss.02425](https://doi.org/10.21105/joss.02425)