Euroargodev *Cheat Sheet*

argopy
1.4.0

Join the community at 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.to_dataset()
fetcher.data
fetcher.index
```

Select 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 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, https or s3 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 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 / Computation per profile

```
ds.argo.teos10(['SA', 'CT', 'CNDC'])
ds.argo.reduce_profile(fct, params=['PRES', 'TEMP'])
```

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_QC'])
```

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

Argo-BGC specifics API

Set up argopy to work with BGC

Select the BGC dataset with the option **ds='bgc'**

⚠ as of version 1.4.0, only **synthetic** BGC data are supported with the **erddap** data source ⚠

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)
```

Additional variables

Complete your dataset with additional variables

```
ds.argo.canyon_med.predict()
ds.argo.canyon_b.predict()
ds.argo.content.predict()
ds.argo.optic.Zeu()#+Zpd,Z_iPAR_threshold,DCM
```

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 API

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()#or 'bgc-s'
```

Use the specific `query.params` to look for profiles with 1 or more parameter:

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

Use the specific `query.parameter_data_mode` to look for profiles in specified data modes:

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

Argo file stores API

Argo Index API

```
from argopy import ArgoIndex
ArgoIndex().convention_supported

idx = ArgoIndex(index_file='core')
idx = ArgoIndex(host='ftp', index_file='bgc-s')

idx.N_RECORDS
idx.to_dataframe(index=True)
```

```
box = [-60, -55, 40, 45, '2007-08', '2007-09']
idx.query.date(box)
idx.query.lon_lat(box)
idx.query.box(box)
idx.read_domain()
```

```
idx.query.wmo([1901393, 6902755])
idx.query.cyc(12)
idx.query.wmo_cyc(1901393, 12)
idx.read_wmo()
idx.read_dac_wmo()
idx.records_per_wmo()
```

```
idx.query.profiler_type(845)
idx.query.profiler_label('NINJA')
```

```
idx.query.compose({'box': BOX, 'wmo': WMOs})
idx.query.compose({'box': BOX, 'params': 'DOXY'})
idx.query.compose({'box': BOX,
                   'params': (['DOXY', 'DOXY2'],
                              {'logical': 'and'})
                  })
idx.query.compose({'params': 'DOXY',
                   'profiler_label': 'ARVOR'})
```

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

Argo Float API

```
from argopy import ArgoFloat
af = ArgoFloat(6902746)
af = ArgoFloat(6902746, aux=True)
af = ArgoFloat(WMO, host='/home/ref-argo/gdac')
af = ArgoFloat(WMO, host='https')
af = ArgoFloat(WMO, host='ftp')
af = ArgoFloat(WMO, host='s3')
```

```
af.ls_dataset()
af.open_dataset('prof')
af.open_dataset('Sprof', netCDF4=True)
af.open_dataset('prof', lazy=True)
```

Argo GDAC file system API

A **gdacfs** instance will provide most of the required methods to work with any file on a GDAC, without the burden of handling access protocols and paths construction.

Paths are relative to the GDAC root folder (which is natively the case in Argo files index):

```
from argopy import gdacfs
fs = gdacfs()
fs = gdacfs('http')
fs = gdacfs('ftp')
fs = gdacfs('s3')
fs = gdacfs('/home/ref-argo/gdac')
```

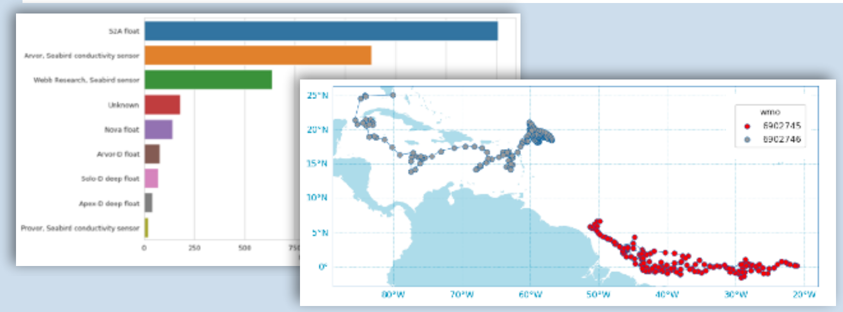
```
fs.glob("dac/aoml/13857/*_meta.nc")
fs.info("dac/aoml/13857/13857_meta.nc")

ds = fs.open_dataset("dac/coriolis/6903091/profiles/
R6903091_001.nc")

with fs.open("ar_index_this_week_meta.txt", "r") as f:
    data = f.readlines()
```

Data visualisation

API

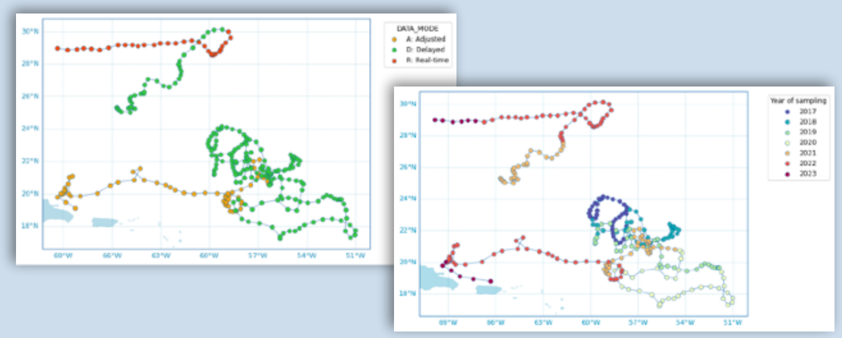
From argopy objects

```
# DataFetcher
from argopy import DataFetcher
f = DataFetcher().region([-75, -45, 20, 30,
                          0, 100, '2015-01', '2020-01'])
f.plot()
f.plot('trajectory')
f.plot('dac')
f.plot('profiler')

# ArgoFloat
from argopy import ArgoFloat
af = ArgoFloat(6902091)
af.plot.trajectory()
af.plot.map('TEMP', pres=450)
af.plot.scatter('DOXY', ds='Sprof')

# ArgoIndex
from argopy import ArgoIndex
idx = ArgoIndex()
idx.plot.trajectory()
idx.plot.bar(by='dac')
```



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()
```

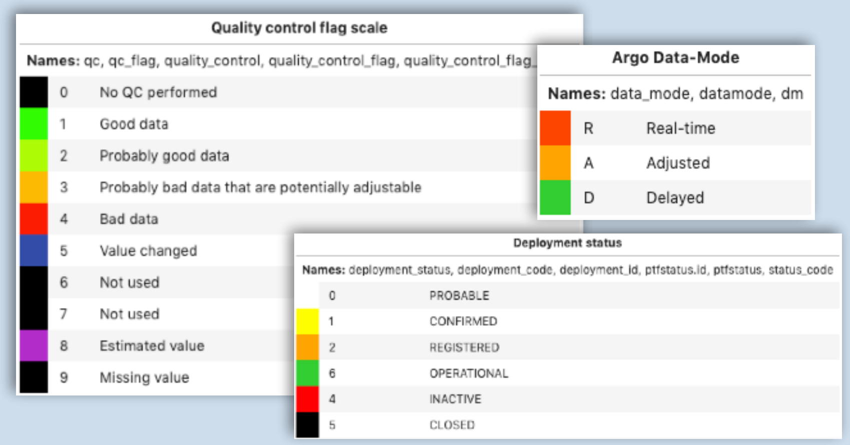
or direct access

```
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')
```



Quality control flag scale

Names: qc, qc_flag, quality_control, quality_control_flag, quality_control_flag_

⬛	0	No QC performed
🟩	1	Good data
🟩	2	Probably good data
🟧	3	Probably bad data that are potentially adjustable
🟥	4	Bad data
🟦	5	Value changed
⬛	6	Not used
⬛	7	Not used
🟪	8	Estimated value
⬛	9	Missing value

Argo Data-Mode

Names: data_mode, datamode, dm

🟥	R	Real-time
🟧	A	Adjusted
🟩	D	Delayed

Deployment status

Names: deployment_status, deployment_code, deployment_id, ptfstatus.id, ptfstatus, status_code

	0	PROBABLE
🟨	1	CONFIRMED
🟧	2	REGISTERED
🟩	6	OPERATIONAL
🟥	4	INACTIVE
⬛	5	CLOSED

Data quality control

API

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(box=[-65, -55,
                                      10,  20,
                                       0, 5000])
     ref_ctd = fetcher.to_xarray()
```

Argo meta & related data

API

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')
```

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')
ArgoDOI('95141').download()
```

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')
```

Float configuration parameters

Read configuration parameters for missions, cycles or launch-time.

```
from argopy import ArgoFloat
af = ArgoFloat(6903091)

af.config['CONFIG_CycleTime_hours']
af.config['CycleTime_hours', 1:3] # by missions
af.config.for_cycles('CycleTime_hours', [11, 12])
af.config.n_params
af.config.parameters
af.config.n_missions
af.config.missions
af.config.cycles
af.config.to_dataframe()

af.launchconfig['CTDDepthZone1SlicesThickness']
af.launchconfig.n_params
af.launchconfig.parameters
af.launchconfig.to_dataframe()
```

Argopy Cheatsheet

API documentation based on argopy release 1.4.0 (Jan. 2026)

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