cmapdata

Release 0.1.0

Diana Haring

Sep 22, 2023

GETTING STARTED

cmapdata is a collection of scripts organized into dataset collection (collect), dataset processing (process) and dataset ingestion (ingest). These docs should provide a primer on data flow into CMAP's databases along with suggested improvements.

CHAPTER

ONE

INSTALLATION AND SETUP

The github repository **cmapdata** is on the Simons CMAP github organization page. To clone the repository navigate to/create the directory location where you want the cmapdata repository to live, enter in terminal:

```
git clone git@github.com:simonscmap/cmapdata.git
```

The DB repository that houses the SQL table creation scripts can be retrieved with:

```
git clone git@github.com:simonscmap/DB.git
```

1.1 Documentation

These docs were built using Sphinx and written in re-structured text. With sphinx installed (via pip), you can build the docs by running the command:

```
make html
```

in the directory **docs/**

This will produce a build in **docs/**

To view these locally, you can open up the index.html with chrome/firefox etc.

These docs are hosted on readthedocs.org. You can set up an account with your github and add a webhook so that the Documentation builds whenever you push to github.

Note: The auto generated API reference does not appear on readthedocs and will only appear locally.

CHAPTER

TWO

DATABASE DESIGN AND TABLE STRUCTURE

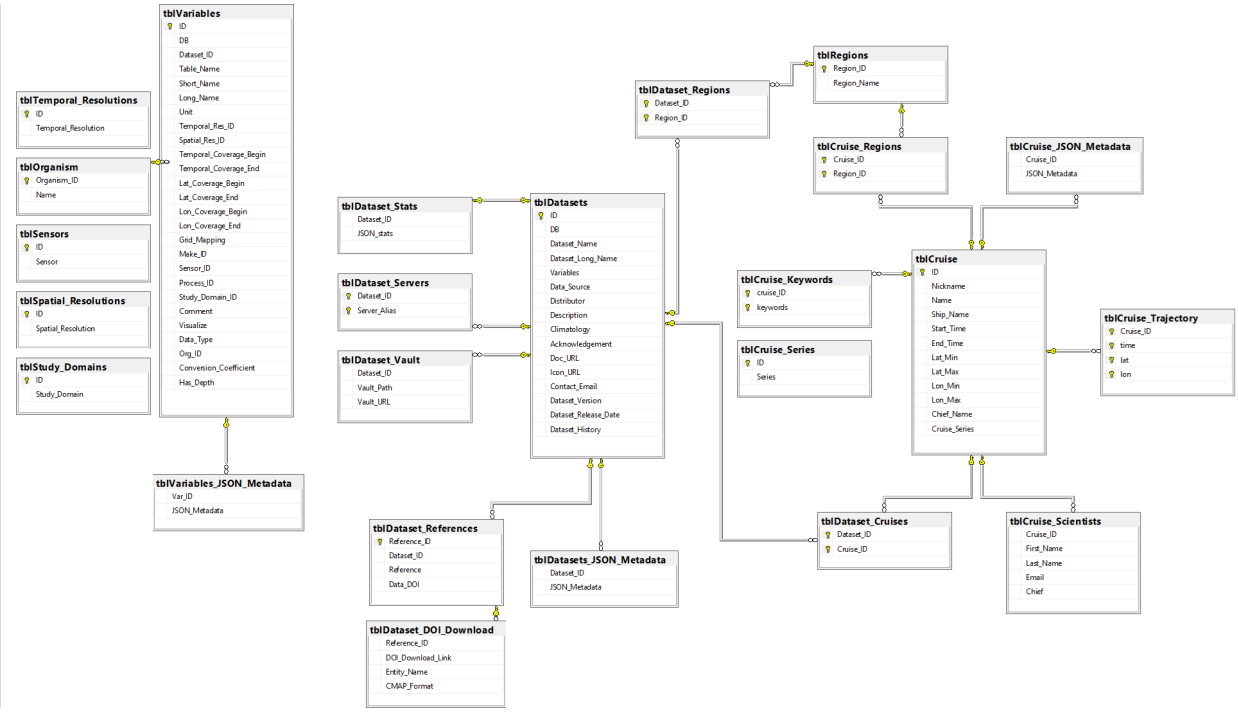


Fig. 1: CMAP DB metadata specific table database diagram

Simons CMAP currently has three servers that contain near replicates of the SQL Server database. The names of these three are: Rainier, Mariana and Rossby. Rainier was the first dedicated server to host the database and currently serves as the main 'source of truth'.

In addition to the three SQL Servers there is a a Spark SQL Warehouse cluster with Apache Hive ANSI SQL:2003 interface. Its alias is: Cluster. This is used for large datasets (i.e. satellite data and Darwin) or large, continuously updated datasets (i.e. Argo data).

Warning: Rainier is currently the production database and 'source of truth'. If you want to test features, use Mariana or Rossby.

Data tables along with metadata tables are stored in the same schema. Common queries live on the database as stored procedures. Data tables are independent of one another. The dataset table name is stored as a column in the metadata table *tblVariables*. This key links the data tables to the rest of the metadata.

Metadata tables are listed below with a brief description of each:

2.1 Variable Level Metadata

tblVariables links the data tables to the metadata through the column [Table_Name]. Columns with an _ID suffix are linked to other metadata tables. For example, Dataset level information in tblDatasets in linked through [Dataset_ID]. Each data variable in a dataset has a row in this table, containing the following columns:

- ID
- DB
- Dataset_ID
- Table_Name
- Short_Name
- Long_Name
- Unit
- Temporal_Res_ID
- Spatial_Res_ID
- Temporal_Coverage_Begin
- Temporal_Coverage_End
- Lat_Coverage_Begin
- Lat_Coverage_End
- Lon_Coverage_Begin
- Lon_Coverage_End
- Grid_Mapping
- Make_ID
- Sensor_ID
- Process_ID
- Study_Domain_ID
- Comment
- Visualize
- Data_Type
- Org_ID
- Conversion_Coefficient
- Has_Depth

tblOrganism contains the organism name and the Organism_ID connects to Org_ID in tblVariables for a variable describing organism abundance. Additional tables related to the organism identification project are: **tblOrgTaxon**, **tblOrgTrophic_Level**, **tblOrgSubtrophics**, **tblOrgTrophics**, **tblOrgSize_Image_Bigelow**, **tblOrgFunctional_Group_WORMS**, **tblOrgParaphyletic_Group_WORMS**, **tblOrgUnicellularity_WORMS**. Details on the project can be found in Jira Epic 8 (https://simonscmap.atlassian.net/browse/CMAP-8)

tblKeywords contains user submitted keywords used in the searching of a variable. tblKeywords contains an ID column, where each value which corresponds to a unique variable entry in tblVariables.

- var_ID
- keywords

tblTemporal_Resolution, **tblSpatial_Resolution**, **tblMake**, **tblSensor**, **tblProcess_Stages**, and **tblStudy_Domains** are all variable level tables that contain links between the ID's in tblVariables and their respective tables.

tblVariables_JSON_Metadata contains additional variable metadata that is unstructured to allow users to include any information that does not fall within the information in tblVariables.

For details on the unstructured metadata project see Jira the following tickets: (https://simonscmap.atlassian.net/browse/CMAP-563, https://simonscmap.atlassian.net/browse/CMAP-572). Each unstructured metadata object includes a value array and a description array. Values and descriptions are always arrays, even if empty or single values. Also, these arrays must always have identical lengths, even if descriptions are empty strings. Descriptions are meant to be human readable, short descriptions akin to alt-text for an image online. A single variable may have multiple entries in tblVariables_JSON_Metadata. An example of a variable-level unstructured metadata is:

```
{"cruise_names":{"values":["PS71"],"descriptions":["Operators Cruise Name"]},"meta_links
↪":{"values":["https://www.bodc.ac.uk/data/documents/nodb/285421/"],"descriptions":[
↪"BODC documentation link"]}}
```

Note: As of September 2023 the only dataset with unstructured metadata is Geotraces Seawater IDP2021v2. Argo Core and Argo BGC are both good candidates for including unstructured metadata.

2.2 Dataset Level Metadata

tblDatasets contains dataset level information and has links to tblVariables as well as links to cruise and region level information.

- ID
- DB
- Dataset_Name
- Dataset_Long_Name
- Variables
- Data_Source
- Distributor
- Description
- Climatology
- Acknowledgement
- Doc_URL
- Icon_URL
- Contact_Email
- Dataset_Version

- Dataset_Release_Date
- Dataset_History

tblDataset_References holds references associated with the dataset, typically a DOI, paper citation, or website. References that are true DOIs with data frozen in time are linked by Reference_ID to **tblDataset_DOI_Download**. This table is used for automating the download of DOI data (DOI_Download_Link) and includes a flag for whether the DOI download is the CMAP template used for submission via the validator (CMAP_Format)

tblDataset_Vault contains the relative path to the dataset leaf directory as well as a public link to the dataset leaf directory (read-only permission). In the future this may be joined to the catalog.

tblDataset_Servers holds the alias names of each server the dataset can be found on. This was implemented to allow for replication across some but not all servers, and free up space on Rainier.

Cruise Metadata The metadata for cruises in CMAP is separated into multiple tables. A core **tblCruise** links together metadata tables for cruise trajectories, cruise keywords, cruise region links and cruise dataset links. **tblDataset_Cruises** is the linking table to connect cruise_IDs with dataset_IDs.

Region Metadata The region tables in CMAP share a similar schema to the cruise tables layout. **tblRegions** contains the ocean regions (this can be expanded). **tblDataset_Regions** is similar to **tblDataset_Cruises** because it acts as the linking table between the region and dataset tables. **tblCruise_Regions** is a linking table between cruise IDs and region IDs.

tblDatasets_JSON_Metadata contains additional dataset metadata that is unstructured to allow users to include any information that does not fall within the information in tblDatasets.

For details on the unstructured metadata project see Jira the following tickets: (https://simonscmap.atlassian.net/browse/CMAP-563, https://simonscmap.atlassian.net/browse/CMAP-572). As with the variable-level unstructured metadata, ach unstructured metadata object for datasets includes a value array and a description array. Values and descriptions are always arrays, even if empty or single values. Also, these arrays must always have identical lengths, even if descriptions are empty strings. Descriptions are meant to be human readable, short descriptions akin to alt-text for an image online. A dataset may have multiple entries in tblDatasets_JSON_Metadata. An example of a dataset-level unstructured metadata is:

```
{"publication_link":{"values":["https://www.geotraces.org/geotraces-publications-
↪database/"],"descriptions":["Link to database of GEOTRACES publications"]}}
```

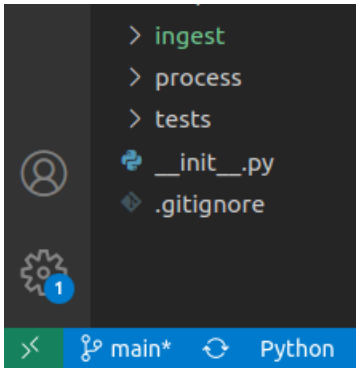
Note: As of September 2023 the only dataset with unstructured metadata is Geotraces Seawater IDP2021v2. Argo Core and Argo BGC are both good candidates for including unstructured metadata.

CHAPTER

THREE

COMPUTE RESOURCES AND DATA STORAGE

The two main computers used in the ingestion pipeline are a Dell XPS 15 laptop and a newer Exxact workstation. For memory intensive and multi-core data processing, the workstation is a useful resource. It could either be used directly from the lab or ssh'ed into to run processing jobs. Using VS Code's **Remote-SSH** extension, you can connect and modify files over ssh without using command line editors. To start a connection, click on the bottom left green icon. The ip address is for the workstation is 128.208.238.117



3.1 Data Flow

The web validator stores submitted datasets to Dropbox (**Dropbox/Apps/<dataset_short_name>/<dataset_short_name_timestamp.xls**). After submission the CMAP data team runs the dataset through the QC API. The outputs from the QC API are saved in Dropbox (**Dropbox/Apps/<dataset_short_name>/iterations/1/propose**). When changes are approved by the submitter, a copy of the finalized dataset is added to the accept folder within the iteration folder structure, as well as to the final folder where ingestion will pull from (**Dropbox/Apps/<dataset_short_name>/final**). Only one file should be saved in the final folder for ingestion.

Ingesting a dataset submitted through the validator pulls from the final folder and creates a folder based on the table name in the **vault/** directory.

3.2 Data Storage

Both the web application and the data ingestion pipeline share storage over dropbox. With an unlimited account, we can use dropbox to store all our pre-DB data. In addition to dropbox, the **vault/** also is synced on the workstation under: **~/data/CMAP Data Submission Dropbox/Simons CMAP/vault/**

For details on the vault structure, see Jira ticket 329 (https://simonscmap.atlassian.net/browse/CMAP-329)

```
├── assimilation
├── r2r_cruise
├── model
├── observation
    ├── in-situ
    |   ├── cruise
    |   |   |── {table_name}
    |   |       |── code
    |   |       |── doc
    |   |       |── metadata
    |   |       |── nrt
    |   |       |── raw
    |   |       |── rep
    |   |       |── stats
    |   ├── drifter
    |   ├── float
    |   ├── mixed
    |   └── station
    └── remote
        └── satellite
```

Dropbox's CLI tools are installed on the workstation. Using the selective sync feature of dropbox, the **/vault** stored on disk can be synced with the cloud. By reading/writing to disk, IO speeds for data processing should be improved.

If dropbox has stopped syncing, you can start the CLI by typing in terminal:

```
dropbox start
dropbox status
```

3.3 Workstation Repositories

Scripts for new SQL tables and indicies are written to the DB repository found here: **~/Documents/CMAP/DB/**

Python scripts for collection, ingestion, and processing are written to the cmapdata repository found here: **~/Documents/CMAP/cmapdata/**. The dataingest branch contains the most recent updates.

The vault directory that syncs with Dropbox is found here: **/data/CMAPDataSubmissionDropbox/SimonsCMAP/vault/** Note there are spaces in the directories "CMAP Data Submission" and "Simons CMAP"

Thumbnails for the catalog page are saved here: **/data/CMAPDataSubmissionDropbox/SimonsCMAP/static/mission_icons**

3.4 Synology NAS and Drobo Storage

Before storing data on Dropbox, two non-cloud storage methods were tried. Both the Drobo and Synology NAS are desktop size hard disk storage. Each contains ~40-50TB of disk space. There are limitations to each of these. The Drobo requires a connection through usb-c/thunderbolt. The Synology NAS can be accessed over the internet, ie (Network Attached Storage). They read/write speed for both is quite slow compared to the disks on the workstation. Perhaps one or both could be used as another backup?

CHAPTER

FOUR

PITFALLS

- Mariana has had both hardware and software issues in the past. Keep an eye on ingestion speeds – if they begin to decrease over time this could be a sign of an upcoming failure.
- kdswap0 issue

CHAPTER

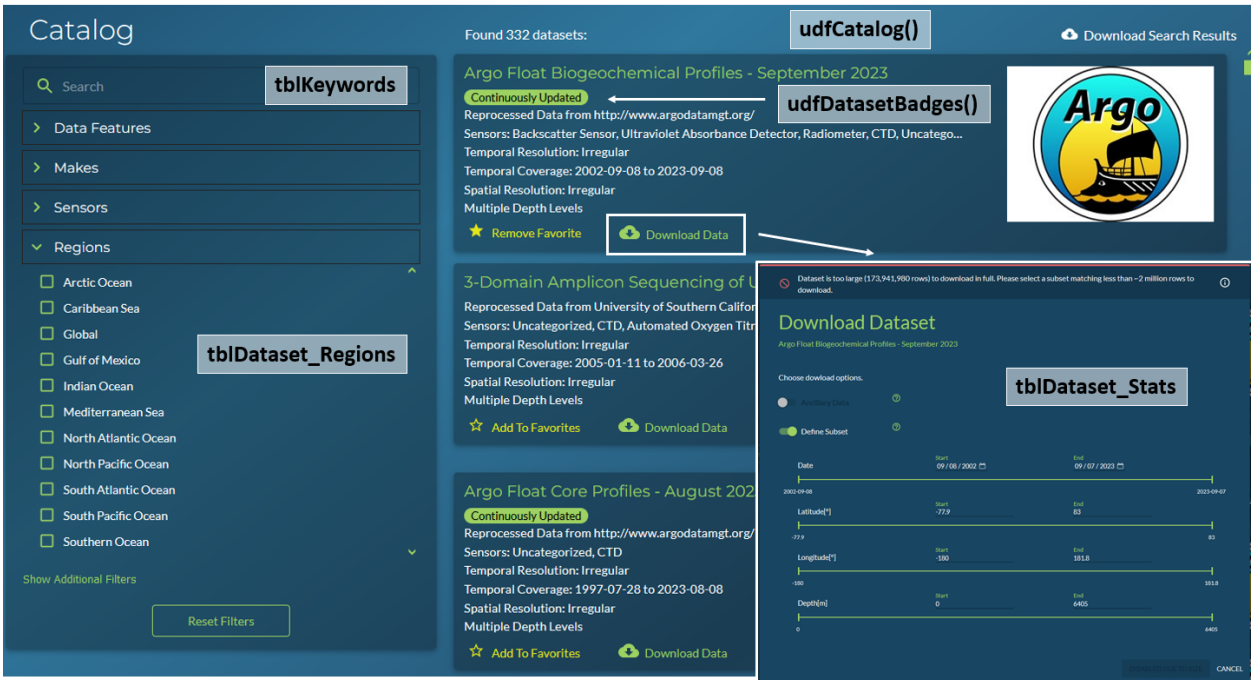
FIVE

CMAP WEBSITE

Below are a handful of screenshots of the CMAP website with names of the core tables in the database that populate each section. The examples given are based on lessons learned when debugging various issues after updates to the database.

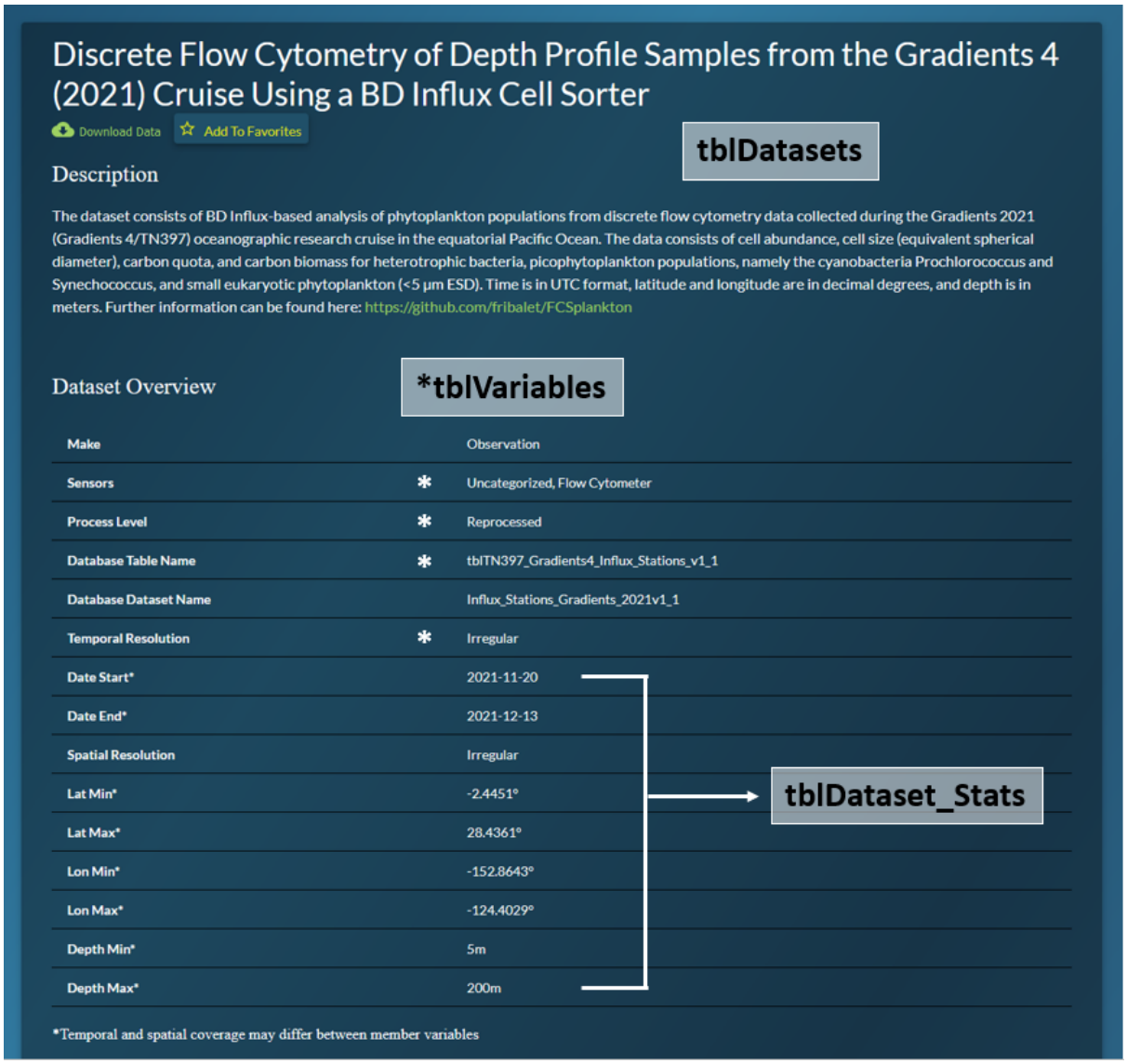
The functionality on the catalog page relies on multiple core tables.

- The search box relies on keywords associated with a dataset.
- The dataset cards are populated by the udfCatalog() SQL function
- The regions filter pulls from tblDataset_Regions. While all datasets that have gone through the QCI API checks include regions as keywords, the region filter does not reference keywords.
- The download data dialog populates the subset values from tblDataset_Stats.

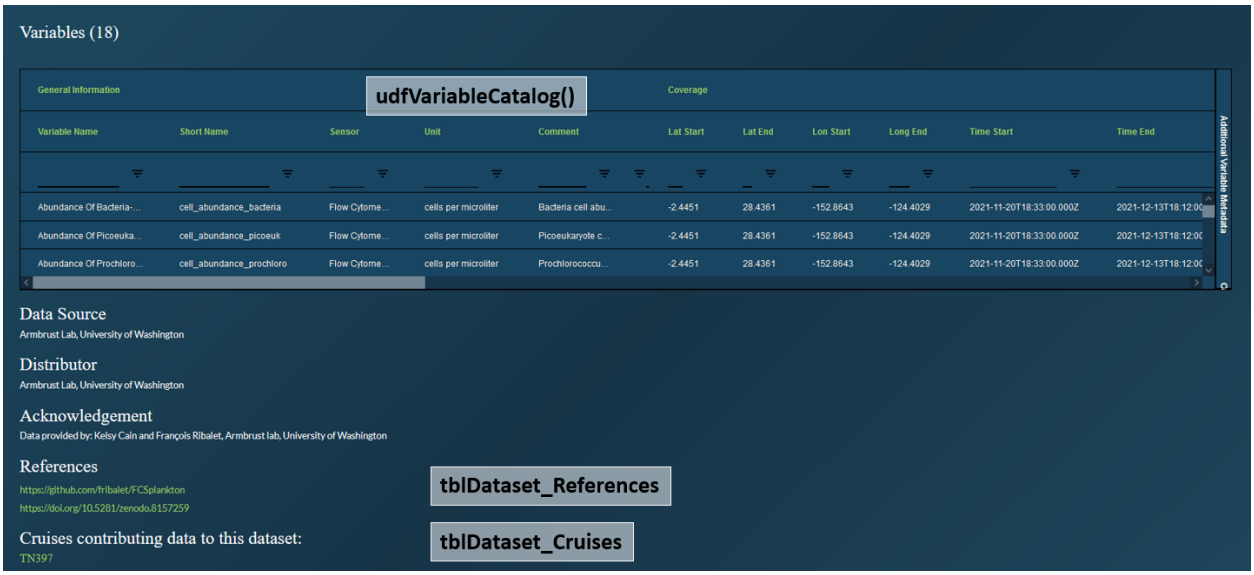


The top of the dataset page uses a stored procedure to join data from three core SQL tables:

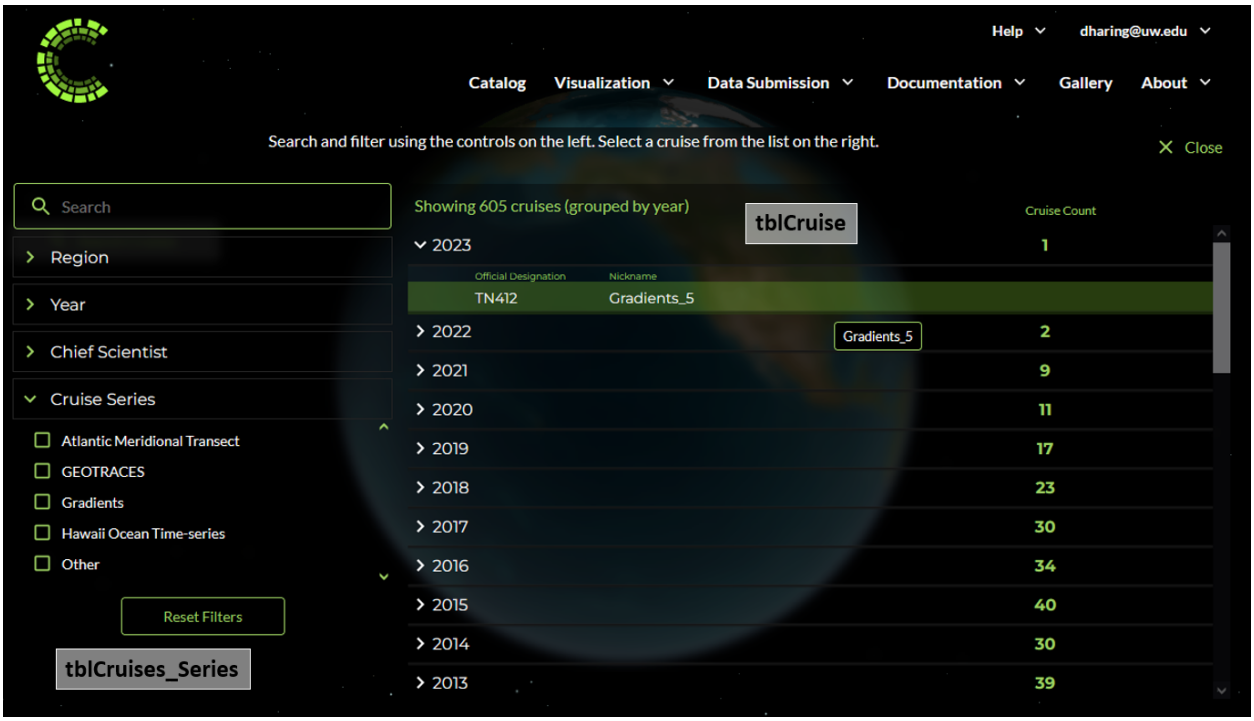
The bottom of the dataset page uses udfVariableCatalog() which is optimized to extract the JSON data from tblDataset_Stats more efficiently. The entries diplayed in the references section show all references submitted in the validator template, along with an additional DOI where applicable. The linked cruises at the bottom will display if there is a match on Name or Nickname in tblCruise. If the cruise is not yet in CMAP, the cruise name included in the


Discrete Flow Cytometry of Depth Profile Samples from the Gradients 4 (2021) Cruise Using a BD Influx Cell Sorter
Download Data
Add To Favorites
tblDatasets
Description
The dataset consists of BD Influx-based analysis of phytoplankton populations from discrete flow cytometry data collected during the Gradients 2021 (Gradients 4/TN397) oceanographic research cruise in the equatorial Pacific Ocean. The data consists of cell abundance, cell size (equivalent spherical diameter), carbon quota, and carbon biomass for heterotrophic bacteria, picophytoplankton populations, namely the cyanobacteria Prochlorococcus and Synechococcus, and small eukaryotic phytoplankton (<5 µm ESD). Time is in UTC format, latitude and longitude are in decimal degrees, and depth is in meters. Further information can be found here: https://github.com/fribalet/FCSplankton
Dataset Overview
*tblVariables
Make Observation
Sensors * Uncategorized, Flow Cytometer
Process Level * Reprocessed
Database Table Name * tblTN397_Gradients4_Influx_Stations_v1_1
Database Dataset Name Influx_Stations_Gradients_2021v1_1
Temporal Resolution * Irregular
Date Start* 2021-11-20
Date End* 2021-12-13
Spatial Resolution Irregular
Lat Min* -2.4451°
Lat Max* 28.4361°
Lon Min* -152.8643°
Lon Max* -124.4029°
Depth Min* 5m
Depth Max* 200m
tblDataset_Stats
*Temporal and spatial coverage may differ between member variables

dataset_meta_data tab of the validator excel template will not be displayed here. All datasets run through the QC API will include all cruise names and nicknames listed in the template as keywords.

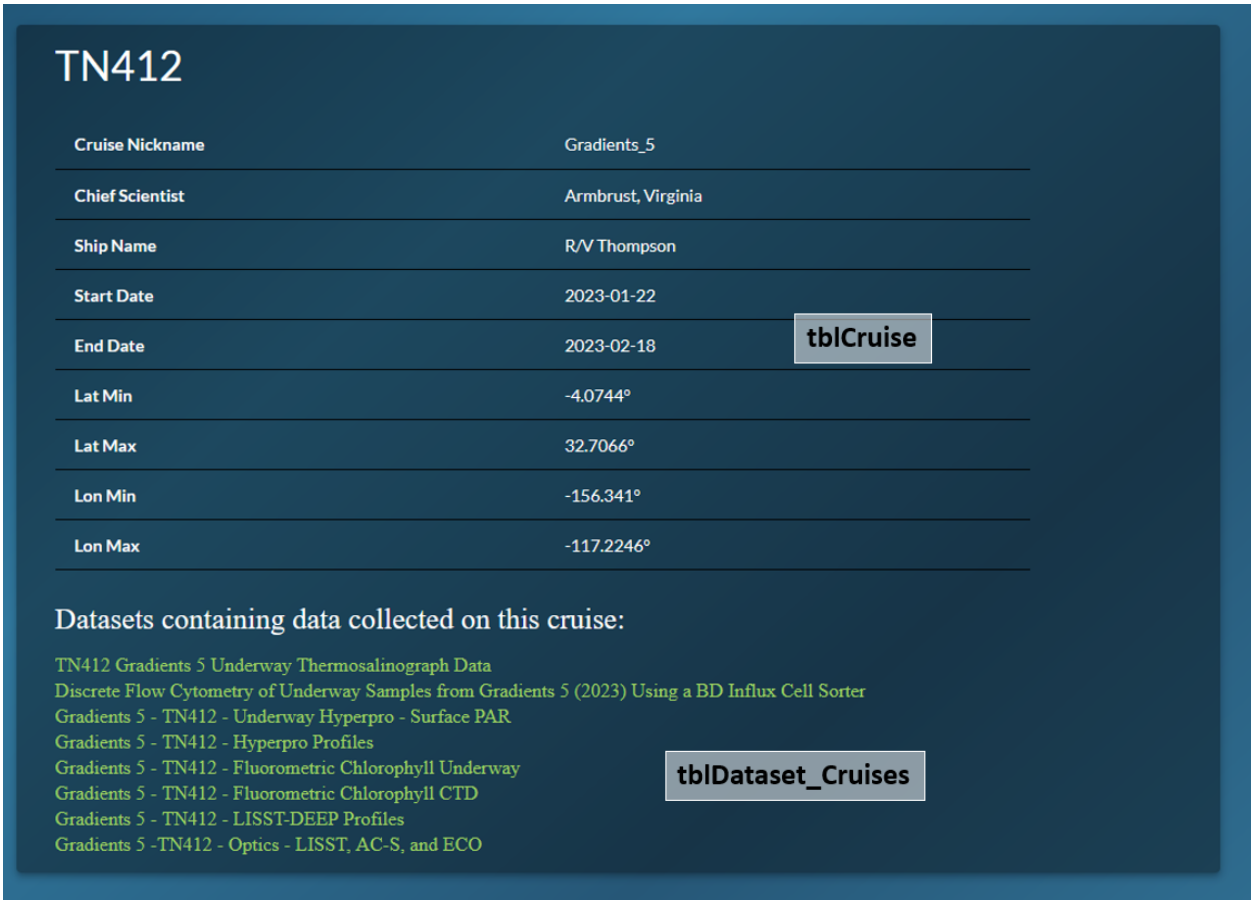


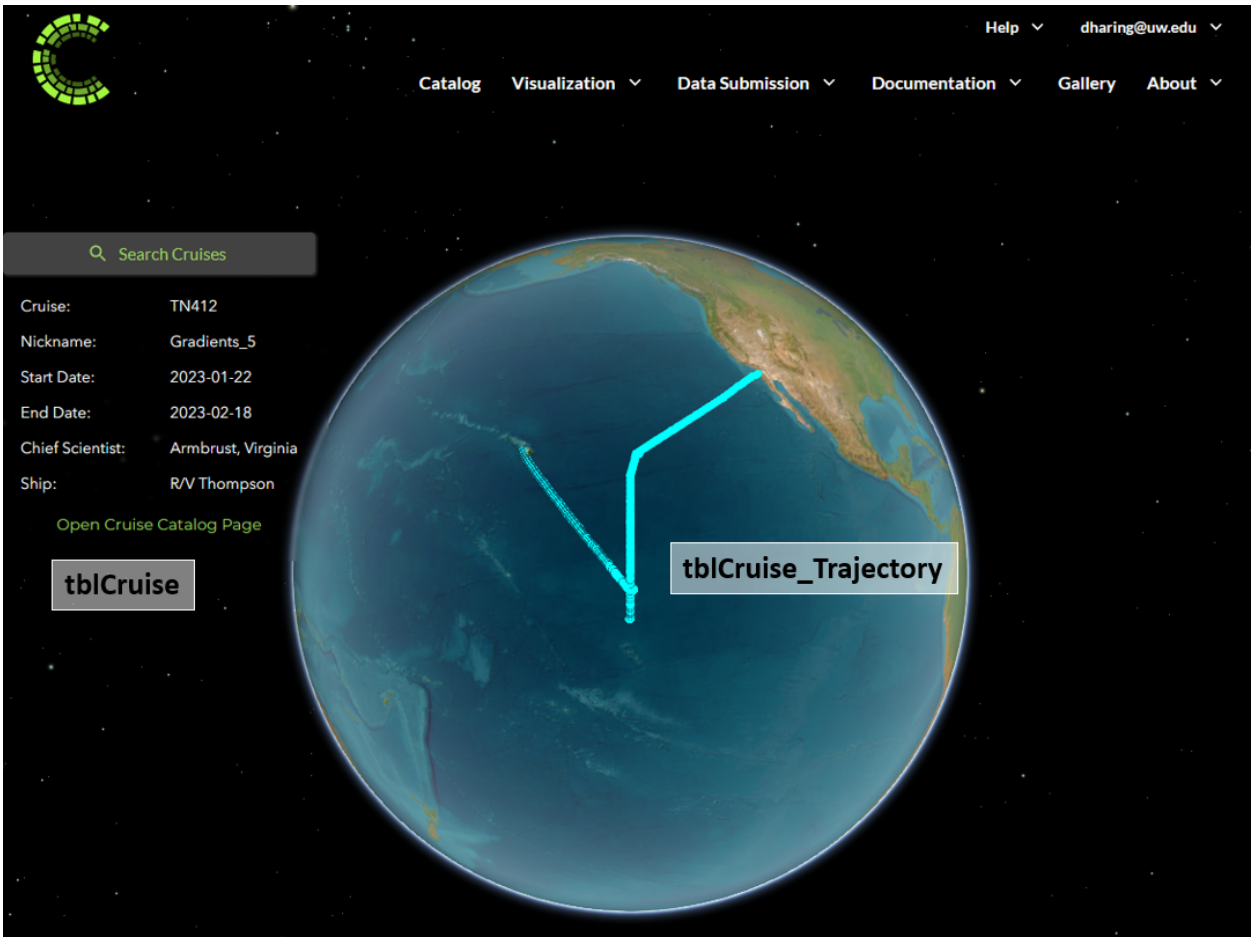
The cruise page groups cruises by year. If a cruise is added to tblCruise but does not include a min/max time value, the cruise name will be grouped in a NULL category at the bottom of listed years.



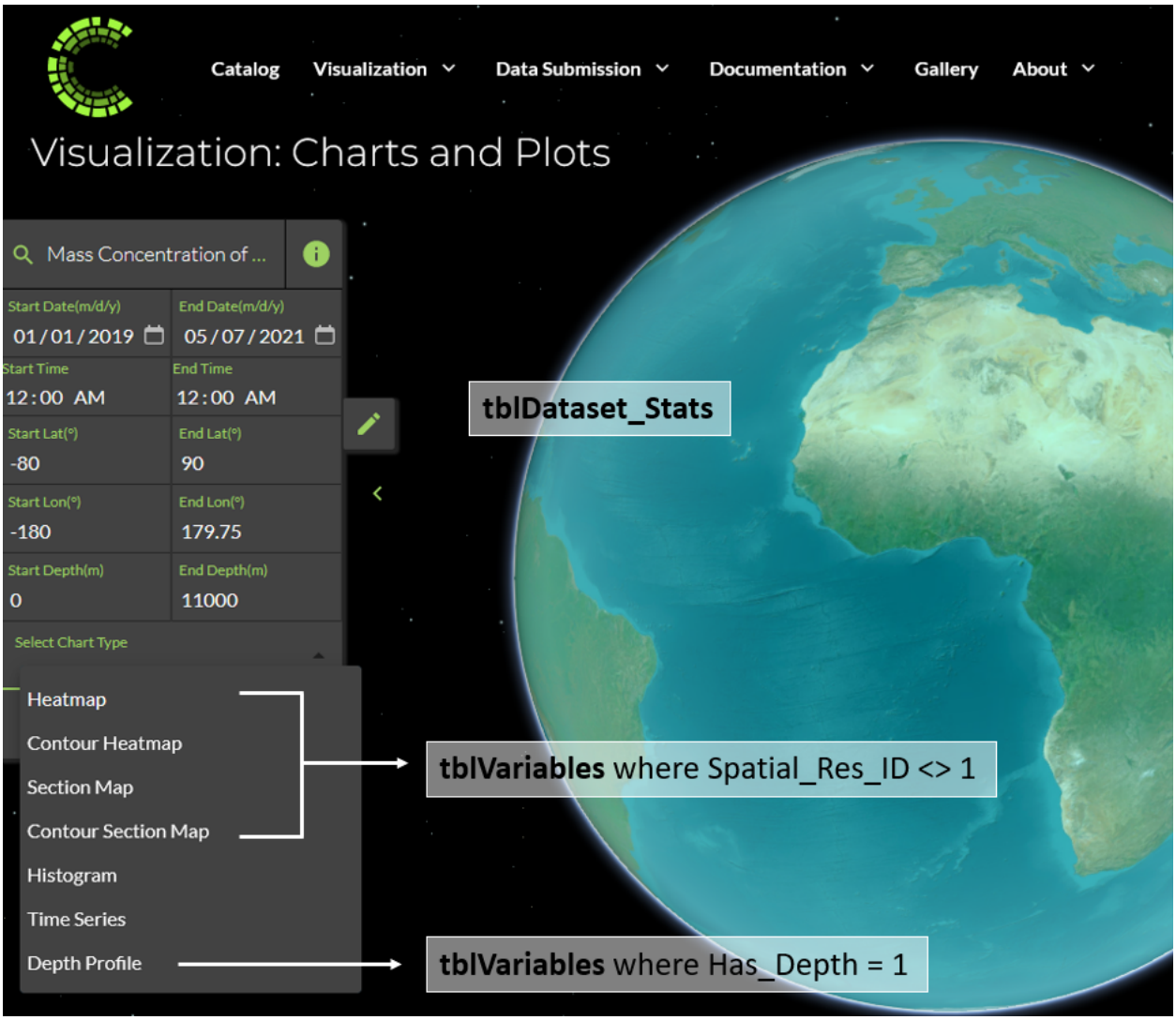
Only cruises that are associated with a dataset are displayed in the cruise search list. The cruise page will link you back to each dataset page it is associated with.

If a cruise is added but does not have an associated trajectory, selecting the cruise will zoom you in on your current view of the globe. The location the globe zooms you to is based on the data in tblCruise_Trajectory, not based on the min/max of lat/lon in tblCruise.


TN412
Cruise Nickname
Gradients_5
Chief Scientist
Armbrust, Virginia
Ship Name
R/V Thompson
Start Date
2023-01-22
End Date
2023-02-18
tblCruise
Lat Min
-4.0744°
Lat Max
32.7066°
Lon Min
-156.341°
Lon Max
-117.2246°
Datasets containing data collected on this cruise:
TN412 Gradients 5 Underway Thermosalinograph Data
Discrete Flow Cytometry of Underway Samples from Gradients 5 (2023) Using a BD Influx Cell Sorter
Gradients 5 - TN412 - Underway Hyperpro - Surface PAR
Gradients 5 - TN412 - Hyperpro Profiles
Gradients 5 - TN412 - Fluorometric Chlorophyll Underway
Gradients 5 - TN412 - Fluorometric Chlorophyll CTD
Gradients 5 - TN412 - LISST-DEEP Profiles
Gradients 5 -TN412 - Optics - LISST, AC-S, and ECO
tblDataset_Cruises


Help
dharing@uw.edu
Catalog
Visualization
Data Submission
Documentation
Gallery
About
Search Cruises
Cruise: TN412
Nickname: Gradients_5
Start Date: 2023-01-22
End Date: 2023-02-18
Chief Scientist: Armbrust, Virginia
Ship: R/V Thompson
Open Cruise Catalog Page
tblCruise
tblCruise_Trajectory

The viz page will show different charting options depending on the variable selected. Only gridded datasets (typically model or satellite) should be given a defined spatial resolution. If a variable is assigned a spatial resolution other than "Irregular" (ID = 1), it cannot have any missing lat/lon values that would leave holes in the global coverage. The depth profile chart option will only be visible if the Has_Depth = 1 in tblVariables.

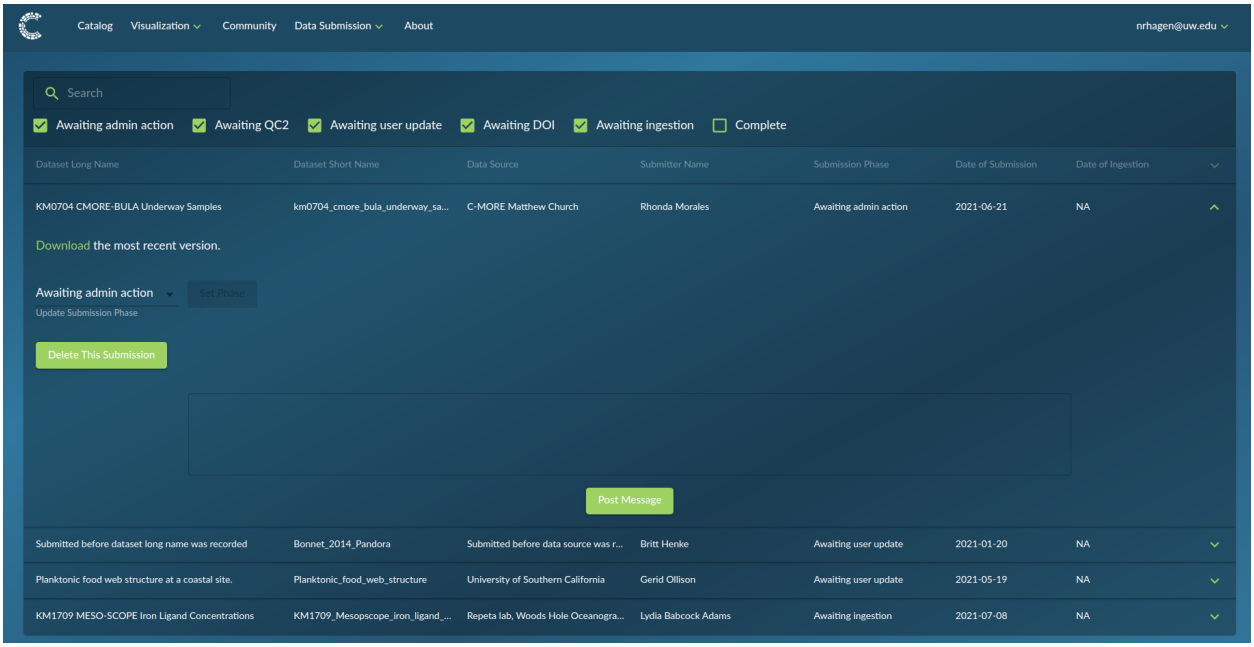


CHAPTER

SIX

WEB VALIDATOR

Collaborator datasets are submitted through our online web validator. This system checks the integrity of the data, dataset metadata and variable metadata sheets, looking for missing information and invalid data types. The status of the dataset is available to the curation team as well as the submitter on the data submission dashboard (https://simonscmap.com/datasubmission/admindashboard)

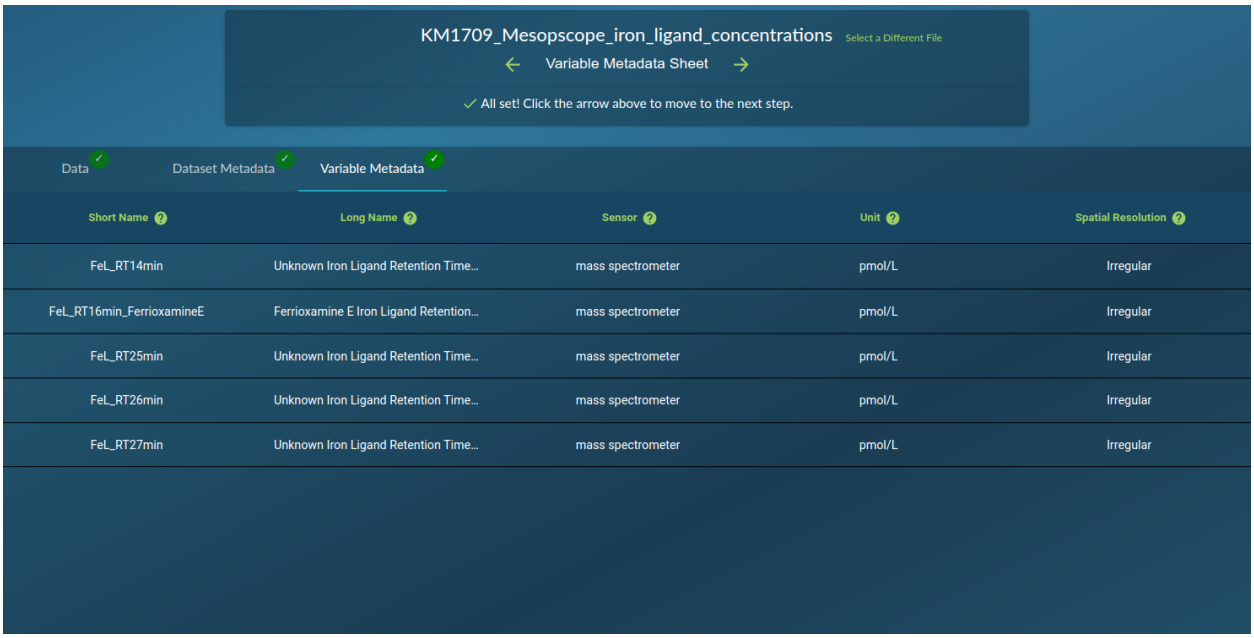


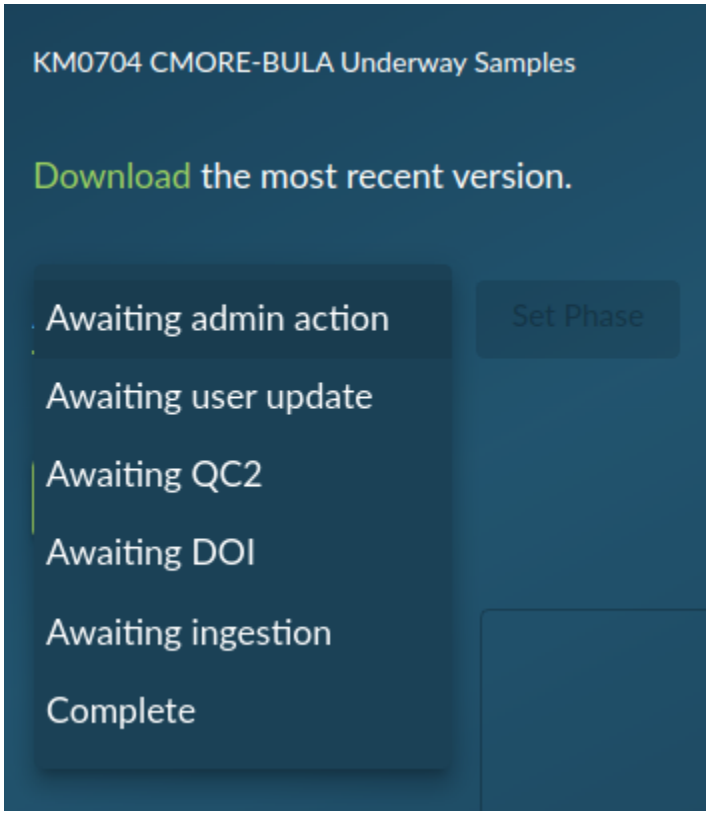
Once a collaborator has successfully submitted a dataset through the validator, the process will be handed over to the data curation team for additional QA/QC checks. Data and metadata should be checked to make sure it conforms to the data submission guide. At this point it is also a good idea to do some sanity checks on the data itself. A common pitfall from data submitters is to mix up the sign of longitude, placing the dataset in the wrong ocean. When the first check is complete, the dataset should have secondary independent QA/QC check.

Any edits of suggestions should be sent back to the submitter through the web validator. Additionally, any changes in the 'phase' should be updated through the dropdown menu in the dashboard.

Once all the dataset changes are complete, the submitter should register a DOI for their dataset and send it over via the validator.

Note: Datasets submitted through the web validator are currently stored in dropbox: 'CMAP Data Submission Dropbox/Simons CMAP/Apps/Simons CMAP Web Data Submission/{dataset_name}/{dataset_name_datestring.xlsx}. Validator folders are currently based on dataset short name. This means it is possible a submitted dataset will be


KM1709_Mesopscope_iron_ligand_concentrations
Select a Different File
Variable Metadata Sheet
All set! Click the arrow above to move to the next step.
Data
Dataset Metadata
Variable Metadata
Short Name
Long Name
Sensor
Unit
Spatial Resolution
FeL_RT14min
Unknown Iron Ligand Retention Time...
mass spectrometer
pmol/L
Irregular
FeL_RT16min_FerrioxamineE
Ferrioxamine E Iron Ligand Retention...
mass spectrometer
pmol/L
Irregular
FeL_RT25min
Unknown Iron Ligand Retention Time...
mass spectrometer
pmol/L
Irregular
FeL_RT26min
Unknown Iron Ligand Retention Time...
mass spectrometer
pmol/L
Irregular
FeL_RT27min
Unknown Iron Ligand Retention Time...
mass spectrometer
pmol/L
Irregular


KM0704 CMORE-BULA Underway Samples
Download the most recent version.
Awaiting admin action
Awaiting user update
Awaiting QC2
Awaiting DOI
Awaiting ingestion
Complete
Set Phase

saved to a folder that already exists, if a user reuses a dataset short name. A fix for this should be implemented in the future by leveraging the unique ID in tblData_Submissions.

CHAPTER

SEVEN

WORKFLOW

Warning: Rainier is currently the production database and 'source of truth'. If you want to test features, use Mariana or Rossby.

The process for ingesting datasets into CMAP differs based on a few factors. The three main categories are *User Submitted Datasets*, *Outside 'Small' Datasets* and *Outside 'Large' Datasets*. User submitted datasets that pass through the web validator must be <150MB. *Outside 'Small' Datasets* are datasets collected that are collected from an outside source that can generally fit in memory. An example would be an AMT or HOT dataset. *Outside 'Large' Datasets* are datasets collected from an outside source that have multiple data files and cannot fit into memory. Examples are satellite data, model data or large insitu collections such as ARGO or GOSHIP.

7.1 User Submitted Datasets

User submitted datasets are submitted through the web validator. Once the QA/QC checks are completed and a DOI is received, the dataset can be ingested into CMAP. Details on the QC process can be found here: https://simonscmap.atlassian.net/browse/CMAP-621. An additional step of adding org_id and conversion_coefficient columns to the variable metadata sheet in the submitted template is done only for variables describing organism abundance.

Using general.py, you can pass command line arguments to specify which server you wish to add the dataset to as well as including a DOI.

Where we have:

```
python general.py {table_name} {branch} {filename} {-d} {DOI link} {-l} {DOI download␣
↪link} {-f} {DOI file name} {-S} {server}
```

- {**table_name**}: Table name for the dataset. Must start with prefix "tbl". Ex. tblFalkor_2018
- {**branch**}: Branch where dataset should be placed in Vault. Ex's: cruise, float, station, satellite, model, assimilation
- {**filename**}: Base file name in vault/staging/combined/. Ex.: 'global_diazotroph_nifH.xlsx'
- {**-d**}: Optional flag for including DOI with dataset in tblReferences. DOI link string follows flag arg.
- {**DOI link**}: String for full web address of CMAP specific DOI. Ex. "https://doi.org/10.5281/zenodo.8306724"
- {**-l**}: Optional flag for including the DOI download link in tblDataset_DOI_Download. DOI dowload link string follows flag.
- {**DOI download link**}: String for DOI download link of CMAP specific DOI. Ex. "https://zenodo.org/record/8306724/files/Gradients5_TN412_LISST_DEEP_Profiles.xlsx?download=1"

- {**-f**}: Optional flag for DOI file name. DOI file name string follows flag.
- {**DOI file name**}: String for filename of CMAP specific DOI. Ex. "Gradients5_TN412_LISST_DEEP_Profiles.xlsx"
- {**-t**}: Optional flag for denoting if DOI is a web validator template. Default value is 1.
- {**DOI in CMAP template**}: Boolean if DOI is a web validator template.
- {**-S**}: Required flag for specifying server choice. Server name string follows flag.
- {**server**}: Valid server name string. Ex. "Rainier", "Mariana" or "Rossby"

An example string would be:

```
python general.py tblTN412_Gradients5_LISST_DEEP_Profiles cruise 'Gradients5_TN412_LISST_
↪DEEP_Profiles.xlsx' -S 'Rossby' -d 'https://doi.org/10.5281/zenodo.8306724' -l 'https:/
↪/zenodo.org/record/8306724/files/Gradients5_TN412_LISST_DEEP_Profiles.xlsx?download=1'␣
↪-f 'Gradients5_TN412_LISST_DEEP_Profiles.xlsx'
```

general.py contains wrapper functions that will split the excel sheet into pandas dataframes, transfer the data to vault/, build a suggested SQL table, insert data, split dataset_meta_data and vars_meta_data into SQL queries and insert into SQL metadata tables, build summary statistics, match provided cruises to cruises in the database, classify the dataset into ocean regions and create maps and icons for the web catalog.

Certain functions are only run when the server name is Rainier (creating icon map, data server alias assignment, and data ingestion tests). A suggested order for server ingestion is starting with Rossby (the fastest server), then ingesting to Mariana, and finally on Rainier. As DOIs are requirements for user submitted datasets, a function to test the data in the DOI matches the data in Rainier also runs automatically.

7.2 Outside 'Small' Datasets

These datasets usually need quite a bit of data munging to make them match the CMAP data format. Additionally, metadata needs to be collected and created. To keep a record of data transformations, any processing scripts should be placed in **/process/../process_datasetname.py**. Additionally, any relevant collection information should be placed in **/collect/../collect_datasetname.py**. A text file containing a link to the process and collect scripts in GitHub should be saved in the vault to **{dataset table name}/code/**

With the addition of the QC API, it is suggested to submit the final, cleaned dataset to the validator. Once QC is complete and the /final folder is populated with the finalized template, ingestion can be done as if it was a user submitted dataset as described above.

7.3 Outside 'Large' Datasets

These datasets are usually composed of multiple data files (generally in netcdf or hdf5). Some features of the ingestion pipeline only work for data that can fit into memory. Because of this, special care is needed to ingest these large datasets. All raw data should be saved in the vault /raw folder for the dataset. Depending on the source, data is downloaded using curl/wget/ftp etc. Any collection scripts should be stored in **/collect/../{collect_datasetname.py}. Once data has been transfered, the next step is any data processing. This should be recorded in **/process/../process_datasetname.py**. A text file containing a link to the process and collect scripts in GitHub should be saved in the vault to **{dataset table name}/code/**

In this data processing script, data should be read from the vault /raw folder, cleaned, sorted and inserted into the database(s).

Note: You will need to create a SQL table and add it to the databases prior to ingestion. Any SQL table creation script should be recorded in DB/ (repository is on Simons CMAP github). Adding indexes once the ingestion has completed will likely speed up ingestion.

After the data has been inserted and the indices successfully created, metadata will need to be created and added to the databases. A standard excel template should be used for the dataset and vars metadata sheets. Submit a template to the validator with a dummy data sheet that holds all variables, but only needs one row of data to make it through the validator. This allows the data curation team to run the QC API checks and create the /final folder needed for ingesting the metadata.

There are additional arguments you can use for large datasets:

- {**-a**}: Optional flag for specifying server name where data is located
- {**data_server**}: Valid server name string. Ex. "Rainier", "Mariana", "Rossby", or "Cluster"
- {**-i**}: Optional flag for specifying icon name instead of creating a map thumbnail of the data
- {**icon_filename**}: Filename for icon in Github instead of creating a map thumbnail of data. Ex: argo_small.jpg
- {**-p**}: Optional flag for defining process level
- {**process_level**}: Default value is "rep". Change to "nrt" for near-real-time datasets
- {**-F**}: Optional flag for specifying a dataset has a valid depth column. Default value is 0
- {**-N**}: Optional flag for specifying a 'dataless' ingestion or a metadata only ingestion

The {-a} flag can be used if the data is not present on all on-prem servers (Rainier, Rossby, and Mariana), and has to be used if the data is only on the cluster. It can also help speed up the calculation of stats when ingesting metadata to Mariana or Rainier, if you use Rossby as the data_server. Rossby is the fastest on-prem server.

The {-i} flag is used if you want to display a logo instead of creating a map of the data for the thumbnail on the catalog page. The icon_filename needs to include the file extension, and should reference a logo or icon already saved in /static/mission_icons

The {-F} flag is needed when adding metadata that doesn't include the full dataset in the excel template. When ingesting a template with data in it, the ingestion code checks for the presence of a depth field automatically. The depth flag is needed for the viz page to know which chart types to display. If a large dataset has a "depth" field (which should only be named as such if there are no rows with missing depth values), include **-F 1** in your ingestion command.

7.4 Metadata Updates

There are times when a dataset is already ingested, but updates to the metadata are needed. The **-U** argument will delete all metadata present for the dataset, but will retain the data table. The **-F** depth flag will need to be included if the dataset has depth. All flags related to a DOI will need to be included if the DOI link is not in the dataset_references column of the dataset_meta_data tab.

CHAPTER

EIGHT

TABLE CREATION AND INDEXING

8.1 Space-Time Index

All datasets in CMAP share a common data format that includes columns with space a time information (time, lat, lon, depth <opt.>). With these, space-time indexes can be created on data tables. Creating clustered indicies on this ST index will speed up query performance on large datasets. Only include depth in the index if the dataset has a depth component.

8.2 Climatology

To quickly calculate climatology for large datasets, specific climatology indicies can be created. Examples of these can be found in the tblSSS_NRT_cl1 or others.

8.3 File Groups

In each server, data in the database is split into File Groups. The available space can be queried with a SQL Server stored procedure. You can execute a stored procedure either in SSMS, Azure Data Studio or a pyodbc query.

```
EXEC uspFileGroup_Volume()
```

CHAPTER

NINE

DATA VALIDATION

There are multiple data validation tests built into the ingestion code. Additional functions can be added to the scripts below, depending on when you would like the checks to run.

9.1 Pre-Ingestion Tests

There are multiple touch points for a use submitted dataset to pass through data and formatting checks. The first is a successful submission through the validator. Following that, the CMAP data curation team uses the QC API (https://cmapdatavalidation.com/docs#tag/Pre-Ingestion-Checks) to run additional checks and fill in keywords. Prior to the existence of the QC API, pre-ingestion checks were written into the ingestion code that are now duplicative. Specifically, checks for outliers in data values are done during submission to the validator, in the viz output of the QC API, and when the data is read into memory for ingestion.

ingest/data_checks.py contains various functions for checking data, either before or during ingestion.

check_df_on_trajectory was a test written before the QC API checked if the data submitted fell within the space and time bounds of the associate cruise trajectory in CMAP. It uses the CMAP_Sandbox database on Beast, which will likely be retired.

check_df_values checks that lat and lon are within range, depth is not below zero, and checks all numeric variables for min / max < or > 5 times standard deviation of dataset. Returns 0 if all checks pass, returns 1 or more for each variable with values out of expected range. This is called in **general.py** with each ingestion.

check_df_nulls checks a dataframe against an existing SQL table for nulls where a SQL column is defined as not null. Returns 0 if all checks pass, returns 1 or more for each dataframe variable with nulls where SQL column is not null.

check_df_constraint checks a dataframe again an existing SQL table's unique indicies. Returns 0 if all checks pass, returns 1 if duplicates are in the dataframe in columns that should contain unique values.

check_df_dtypes checks a dataframe against an existing SQL table. Returns 0 if all checks pass, returns 1 or more for each column with different data types.

check_df_ingest runs the last three tests and returns 0 if all checks passed.

check_metadata_for_organism checks variable_metadata_df for variable names or units that could be associated with organisms. Checks variable short and long names for any name found in tblOrganism. Returns True if no potential organism variables are present. This is called in **general.py** with each ingestion.

validate_organism_ingest checks for the presence of org_id and conversion_coefficient in the vars_meta_data sheet. If those columns are present but blank, if checks that they should be blank. Checks that both columns are filled if one is filled for a variable. Checks that the conversion_coefficient is correct based on unit (this last check could use additional logic). Returns True if columns are not present or no issues are found. This is called in **general.py** with each ingestion.

9.2 Post-Ingestion Tests

All post-ingestion tests are run when the ingestion server is Rainier. As Rainier is the production database for the CMAP website, testing for a successful ingestion on Rossby first is suggested. With Rainier as the final server for ingestion, the following tests are run in **post_ingest.py**:

checkServerAlias checks the table is present on each server listed for a dataset in tblDataset_Servers. If less than three servers are listed in tblDataset_Server, it checks each on prem server if the table is present.

checkRegion checks that at least one region is associated with a dataset in tblDataset_Regions. If the dataset only lives on the cluster it will either assign all regions associated with Argo datasets, or allow you to enter your own. If no regions are associated with a dataset and it is in an on-prem server, regions will be assigned automatically based on a distinct list of lat and lon.

checkHasDepth checks that the Has_Depth flag in tblVariables is accurate based on the presence of a depth column in the data table.

compareDOI downloads a CMAP template from a DOI link and checks the data against what is in SQL. Checks numeric columns with math.isclose() as the number of significant digits can change on import. Deletes downloaded template after checks.

pycmapChecks calls various pycmap functions. Skips tests on stats if the dataset is larger than 2 million rows (included to stop SELECT * FROM running on the cluster, specifically for Argo). Note: due to a cache of the dataset IDs on the api layer, it's possible the pycmap tests will fail if the cache has not reset after ingestion.

fullIngestPostChecks runs all checks listed above, with the optional argument to check the DOI data. Also runs **api_checks.postIngestAPIChecks()** on every 10th dataset (see DB API Endpoints below for details on this). This is called in **general.py** with each ingestion to Rainier.

9.3 DB API Endpoints

Information on the DB API endpoints can be found here: https://cmapdatavalidation.com/docs#tag/Post-Ingestion-Checks

You can test out each endpoint here: https://cmapdatavalidation.com/try#/

Running the DB API checks on each ingestion should not be necessary. There is both a monetary cost for each API check and an unneccesary load on the servers to justify running these checks on each ingestion. Currently the logic to run these checks is on every 10 datasets. Each new dataset, or metadata update, will result in a new Dataset ID. These IDs are not backfilled, so when metadata is deleted, the old Dataset ID will not be reused.

The **fullIngestPostChecks** function will run **api_checks.postIngestAPIChecks() ** when a Dataset ID is divisible by ten.

```
def postIngestAPIChecks(server = 'Rossby'):
## Runs DB endpoint checks. Default server is Rossby
    db_name = 'Opedia'
    strandedTables()
    strandedVariables(server, db_name) ## Checks all on prem servers
    numericLeadingVariables(server, db_name)
    duplicateVarLongName(server, db_name)
    duplicateDatasetLongName()
    datasetsWithBlankSpaces(server, db_name)
    varsWithBlankSpace(server, db_name)
```

The default server is Rossby as it's the fastest. Only strandedVariables() checks all on prem servers.

CHAPTER

TEN

CONTINUOUS INGESTION

There are currently 13 datasets processed and ingested continuously. For details on the project, see Jira ticket 688 (https://simonscmap.atlassian.net/browse/CMAP-688)

All near real time (NRT) datasets are only ingested to the cluster. Note that dataset replication can be done across any of our servers. See Jira ticket 582 for details on the distributed dataset project (https://simonscmap.atlassian.net/browse/CMAP-582). The following datasets are downloaded, processed, and ingested utilizing run_cont_ingestion.py. This can run via the terminal:

```
cd ~/Documents/CMAP/cmapdata
python run_cont_ingestion.py
```

Table 1: Datasets Collected and Processed Daily

Dataset	Target Servers	New Data Available
Sattelite SST	Rainier, Mariana, Rossby, Cluster	Daily
Satellite SSS	Rainier, Mariana, Rossby, Cluster	Daily, ~2 week lag
Satellite CHL	Mariana, Rossby, Cluster	~Weekly, ~2 month lag
Satellite CHL NRT	Cluster	~Weekly
Satellite POC	Mariana, Rossby, Cluster	~Weekly, ~2 month lag
Satellite POC NRT	Cluster	~Weekly
Satellite AOD	Mariana, Rossby, Cluster	Monthly
Satellite Altimetry NRT (Signal)	Cluster	Daily
Satellite Altimetry (Signal)	Mariana, Rossby, Cluster	~4x a year
Satellite PAR (Daily)	Cluster	Daily, ~1 month lag
Satellite PAR NRT (Daily)	Cluster	Daily

Each dataset has a collect and process script.

10.1 Collection Scripts

Collection scripts can be found in cmapdata/collect/model and in cmapdata/collect/sat within the dataingest branch of the GitHub repository (https://github.com/simonscmap/cmapdata). Each dataset (with the exception of Argo) adds an entry per file downloaded to tblProcess_Queue.

tblProcess_Queue contains information for each file downloaded for continuous ingestion. It holds the original naming convention for each file, the file's relative path in the vault, the table name it will be ingested to, the date and time it was downloaded and processed, and a column to take note of download errors.

- ID
- Original_Name
- Path
- Table_Name
- Downloaded
- Processed
- Error_Str

Each collection script does the following:

1. Retries download for any file previously attempted that includes and error message
2. Get the date of the last succesful download
3. Attempts to download the next available date range. Start and end dates are specific to each dataset's temporal resolution and new data availability (see New Data Available column in table above)
4. Entries for each date a download is attempted for are writted to tblProcess_Queue, with successful downloads denoted by a NULL Error_Str

Dataset-specific logic:

1. tblPisces_Forecast_cl1 updates the data on a rolling schedule. Each week new data is available, and ~2 weeks of previous data is overwritten by the data producer. GetCMEMS_NRT_MERCATOR_PISCES_continuous.py will re-download these files, and if successfully downloaded, will delete the existing data for that day from the cluster. Simply creating new parquet files and pushing to S3 does not update the data in the cluster, so it is necessary to delete the data first.
2. The NRT datasets should not overlap with the matching REP datasets. For example, when new data is successfully ingested for tblModis_PAR_cl1, data for that date is deleted from tblModis_PAR_NRT.
3. A successful download for one day of tblWind_NRT_hourly data results in 24 files. An additional check is in place when finding the last successful date downloaded by including there need to be 24 files for that date. If it is less than 24, it will retry downloading that date.
4. Data for tblAltimetry_REP_Signal is updated by the data provider sporadically throughout the year. New data is checked weekly. Date range for downloads is based on latest files in the FTP server. See /collect/sat/CMEMS/GetCMEMS_REP_ALT_SIGNAL_continuous.py for details.

If a file download is attempted but fails because no data is available yet, the Original_Name will be the date where data wasn't available (ex. "2023_08_09") and the Error_Str will be populated. Each collection script has a retryError function that queries dbo.tblProcess_Queue for any entries where Error_Str is not null. If a file is successfully downloaded on a retry, tblProcess_Queue will be updated with the original file name and date of successful download.

10.2 Process Scripts

Each process script does the following:

1. Pulls newly downloaded files from tblProcess_Queue where Error_Str is NULL.
2. Does a schema check on the new downloaded file against the oldest NetCDF in the vault
3. Processes file and adds climatology fields (year, month, week, dayofyear)
4. Does a schema check on the processed file against the oldest parquet file in the vault
5. Ingests to on-prem servers (see Target Servers column in table above)
6. Copies parquet file to S3 bucket for ingestion to the cluster
7. Updates Processed column in tblProcess_Queue
8. Adds new entry to tblIngestion_Queue

tblIngestion_Queue contains information for each file processed for continuous ingestion. It holds the file's relative path in the vault, the table name it will be ingested to, the date and time it was moved to S3 (Staged), and the date and time it was added to the cluster (Started and Ingested).

- ID
- Path
- Table_Name
- Staged
- Started
- Ingested

Once all new files have been processed from tblProcess_Queue and added to tblIngestion_Queue, trigger the ingestion API. The URL is saved in ingest/credentials.py as S3_ingest. It is best to only trigger the ingestion API once, which is why the snippet below is run after files for all datasets have been processed. See Jira ticket 688 for additional details: (https://simonscmap.atlassian.net/browse/CMAP-688)

```
requests.get(cr.S3_ingest)
```

After all files have successfully ingested to the cluster (Ingested will be filled with the date and time it was completed), each dataset will need updates to tblDataset_Stats. In run_cont_ingestion.py, updateCIStats(tbl) formats the min and max times to ensure the download subsetting and viz page works properly. In short, time must be included, along with the '.000Z' suffix.

10.3 Troubleshooting

Occasionally datasets will have days missing, resulting in a date being retried on each new run of run_cont_ingestion.py. In some cases, data will never be provided for these dates. This information can be found in the documentation provided by each data provider. For example, the SMAP instrument used for SSS data experienced downtime between Aug 6 - Sept 23 2022 (see Missing Data section: https://remss.com/missions/smap/salinity/). That date range was deleted from tblProcess_Queue to prevent those dates from being rechecked each run.

If there are known issues of data already ingested that the data producer has fixed, the entry for the impacted dates should be deleted from tblProcess_Queue and tblIngestion_Queue and redownloaded. Data should be delete from impacted on-prem servers and the cluster as applicable before reingestion.

Each dataset's processing script has checks for changes in schema. Some data providers will change the dataset name when a new version is processed, but not all. If a processing script finds a schema change for a dataset that has the same name / ID / version number, a new dataset should be made in CMAP with a suffix denoting a new change log iteration. For example, tblModis_CHL_cl1 is a reprocessed version of tblModis_CHL.

10.4 Batch Ingestion

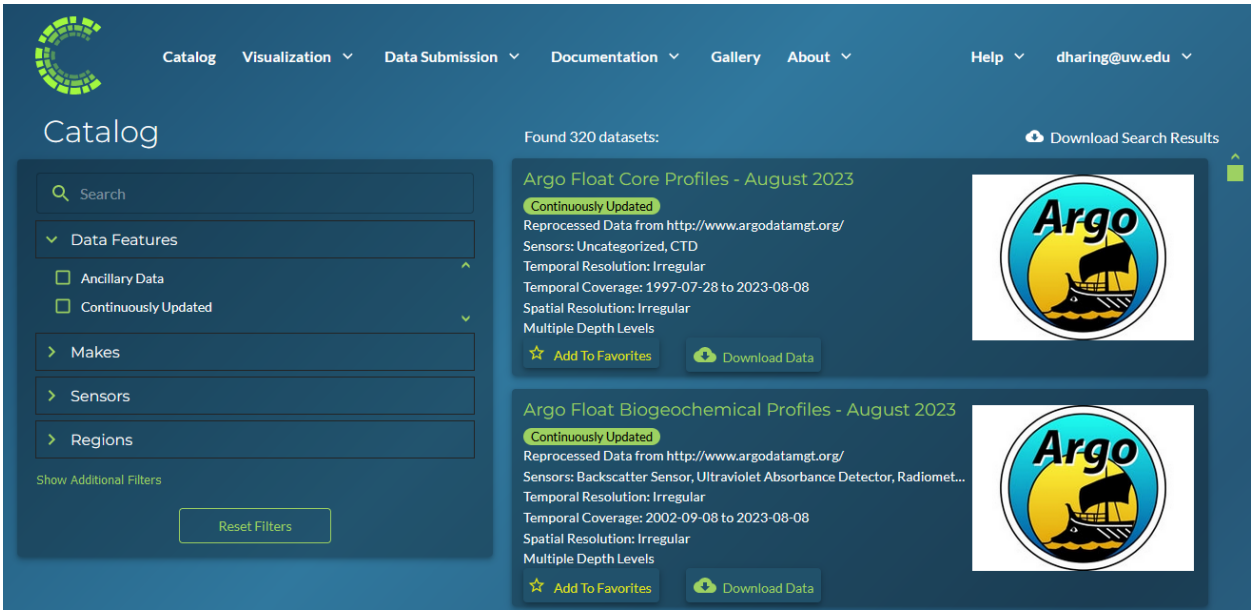
The following datasets are to be ingested monthly. Due to the nature of updates done by the data provider, each month of Argo is a new dataset. These datasets will be ingested via batch ingestion instead of appending to existing tables like the datasets described above. See the outside large dataset walkthrough for details on Argo processing.

Table 2: Datasets Collected and Processed Monthly

Dataset	Target Servers	New Data Available
Argo REP Core	Cluster	Monthly
Argo REP BGC	Cluster	Monthly

10.5 Continuous Ingestion Badge on Website

The CMAP Catalog page includes a filter for Continuously Updated datasets and displays badges for each applicable dataset.



The badges and filter call the uspDatasetBadges stored procedure, which in turn calls the udfDatasetBadges() function. As Argo datasets are a batch ingestion, they are not included in tblProcess_Queue. In order to have the badges display for Argo datasets, a union was done for any Argo REP table, regardless of month.

```
select distinct table_name, ci = 1 from tblProcess_Queue
      union all
      select distinct table_name, ci = 1  from tblvariables where Table_Name like
↪'tblArgo%_REP_%'
```

10.6 Sea Surface Salinity Walkthrough

There are two version of Sea Surface Salinity (SSS) data. For details on the differences see Jira ticket 754 (https://simonscmap.atlassian.net/browse/CMAP-754). Continuous ingestion downloads the REMSS SMAP data. The previous version of SSS data in CMAP was collected from V4.0 provided by REMSS. The updates in the V5.0 release recalculated historic data (details can be found here: https://remss.com/missions/smap/salinity/), which meant we could no longer append new data to the existing table.

Due to the release of V5.0, a new table was made with a "change log" suffix of 1. New SSS data is currently ingested into **tblSSS_NRT_cl1**. The tblSSS_NRT table can be retired and removed from the databases after users have been notified via a news update on the homepage. A typical wait time has been one month after publishing a news story before a dataset can be removed.

10.6.1 Download SSS Data

The data is downloaded using wget, calling the data.remss URL with the year and day of year of the data:

```
file_url = f'https://data.remss.com/smap/SSS/V05.0/FINAL/L3/8day_running/{yr}/RSS_smap_
↪SSS_L3_8day_running_{yr}_{dayn_str}_FNL_V05.0.nc'
```

As with each continuously ingested dataset, there is a function to retry any previous dates that resulted in an error. Errors are generated by either a successful, but empty download, or a failed download attempt.

The first function run in the **GetREMSS_SSS_cl1_continuous.py** script is:

```
def retryError(tbl):
    qry = f"SELECT Original_Name from dbo.tblProcess_Queue WHERE Table_Name = '{tbl}'␣
↪AND Error_Str IS NOT NULL"
    df_err = DB.dbRead(qry, 'Rainier')
    dt_list = df_err['Original_Name'].to_list()
    if len(dt_list)>0:
        dt_list = [dt.datetime.strptime(x.strip(), '%Y_%m_%d').date() for x in dt_list]
        for date in dt_list:
            get_SSS(date, True)
```

This function checks tblProcess_Queue for any previous errors and runs a retry on the download.

The get_SSS function formats the date into the necessary day of year format and attempts a download via wget. If the download is a retry, the entry for that date in tblProcess_Queue will be updated with the original file name and the date of the successful download. If the download is not a retry, a new entry will be added to tblProcess_Queue. If any failure occurs, the error_str will be populated and the original file name will be populated with the date of the data that failed to download (ie "2023_08_26"). This string is converted to a date for future retries.

```
def get_SSS(date, retry=False):
    yr = date.year
    dayn = format(date, "%j")
    dayn_str = dayn.zfill(3)
    file_url = f'https://data.remss.com/smap/SSS/V05.0/FINAL/L3/8day_running/{yr}/RSS_
↪smap_SSS_L3_8day_running_{yr}_{dayn_str}_FNL_V05.0.nc'
    save_path = f'{vs.satellite + tbl}/raw/RSS_smap_SSS_L3_8day_running_{yr}_{dayn_str}_
↪FNL_V05.0.nc'
    wget_str = f'wget --no-check-certificate "{file_url}" -O "{save_path}"'
    try:
        os.system(wget_str)
```

(continues on next page)

(continued from previous page)

```
        Error_Date = date.strftime('%Y_%m_%d')
        Original_Name = f'RSS_smap_SSS_L3_8day_running_{yr}_{dayn_str}_FNL_V05.0.nc'
            ## Remove empty downloads
        if os.path.getsize(save_path) == 0:
            print(f'empty download for {Error_Date}')
            if not retry:
                metadata.tblProcess_Queue_Download_Insert(Error_Date, tbl, 'Opedia',
↪'Rainier','Download Error')
                os.remove(save_path)
        else:
            if retry:
                metadata.tblProcess_Queue_Download_Error_Update(Error_Date, Original_
↪Name,  tbl, 'Opedia', 'Rainier')
                print(f"Successful retry for {Error_Date}")
            else:
                metadata.tblProcess_Queue_Download_Insert(Original_Name, tbl, 'Opedia',
↪'Rainier')
    except:
        print("No file found for date: " + Error_Date )
        metadata.tblProcess_Queue_Download_Insert(Error_Date, tbl, 'Opedia', 'Rainier',
↪'No data')
```

After the retry function is run, the last date that was successfully downloaded is retrieved by checking tblIngestion_Queue, tblProcess_Queue, and max date from the cluster.

```
def getMaxDate(tbl):
  ## Check tblIngestion_Queue for downloaded but not ingested
  qry = f"SELECT Path from dbo.tblIngestion_Queue WHERE Table_Name = '{tbl}' AND␣
↪Ingested IS NULL"
  df_ing = DB.dbRead(qry, 'Rainier')
  if len(df_ing) == 0:
      qry = f"SELECT max(path) mx from dbo.tblIngestion_Queue WHERE Table_Name = '{tbl}'␣
↪AND Ingested IS NOT NULL"
      mx_path = DB.dbRead(qry,'Rainier')
      path_date = mx_path['mx'][0].split('.parquet')[0].rsplit(tbl+'_',1)[1]
      yr, mo, day = path_date.split('_')
      max_path_date = dt.date(int(yr),int(mo),int(day))
      qry = f"SELECT max(original_name) mx from dbo.tblProcess_Queue WHERE Table_Name = '
↪{tbl}' AND Error_str IS NOT NULL"
      mx_name = DB.dbRead(qry,'Rainier')
      if mx_name['mx'][0] == None:
          max_name_date = dt.date(1900,1,1)
      else:
          yr, mo, day = mx_name['mx'][0].strip().split('_')
          max_name_date = dt.date(int(yr),int(mo),int(day))
      max_data_date = api.maxDateCluster(tbl)
      max_date = max([max_path_date,max_name_date,max_data_date])
  else:
      last_path = df_ing['Path'].max()
      path_date = last_path.split('.parquet')[0].rsplit(tbl+'_',1)[1]
      yr, mo, day = path_date.split('_')
      max_date = dt.date(int(yr),int(mo),int(day))
```

(continues on next page)

(continued from previous page)

```
  return max_date
```

The date range to check data for is specific to each dataset depending on the temporal scale and typical delay in new data availability from the data producer. For SSS data from REMSS, there is a NetCDF file for each day (timedelta(days=1)), and new data is generally available on a two week delay.

Note: If new data has not been published by REMSS for a month or so, emailing their support account (support@remss.com) has been helpful to restart their processing job.

10.6.2 Process SSS Data

The first step of **process_REMSS_SSS_cl1_continuous.py** is to pull the list of all newly downloaded files. The tblProcess_Queue and tblIngestion_Queue tables only live on Rainier, so that server needs to be specified when retrieving the new files ready for processing:

```
qry = f"SELECT Original_Name from tblProcess_Queue WHERE Table_Name = '{tbl}' AND Path␣
↪IS NULL AND Error_Str IS NULL"
flist_imp = DB.dbRead(qry,'Rainier')
flist = flist_imp['Original_Name'].str.strip().to_list()
```

The schema of the newly downloaded NetCDF is compared against the oldest NetCDF in the vault. If any new columns are added or renamed, the processing will exit and the data will not be ingested. After the NetCDF has gone through all the processing steps, the schema of the finalized parquet file is checked against the oldest parquet file in the vault. Again, if there are any differences the processing will exit and the data will not be ingested. This logic is present in all continuously ingested (CI) datasets. Additional steps done for all CI datasets include: adding climatology columns, updating tblProcessQueue with processing datetime, saving parquet file to vault, pushing parquet from vault to S3 bucket, and adding a new entry to tblIngestion_Queue.

Processing logic specific to SSS includes: pulling time from NetCDF coordinate, extracting single variable from NetCDF (sss_smap), and mapping longitude from 0, 360 to -180, 180. Because SSS data is frequently accessed, it is ingested into all on-prem servers, as well as the cluster.

A single parquet file is ingested into on-prem servers simultaneously using Pool. The current BCP wrapper creates a temporary csv file with the table name and server name in it, to allow for multiple ingestions at once. The multiprocessing is not done on multiple files for the same dataset across servers as the current file naming convention could cause clashes if overwritten.

The list of original file names is looped through for processing:

```
for fil in tqdm(flist):
  x = xr.open_dataset(base_folder+fil)
  df_keys = list(x.keys())
  df_dims =  list(x.dims)
  if df_keys != test_keys or df_dims!= test_dims:
      print(f"Check columns in {fil}. New: {df.columns.to_list()}, Old: {list(x.keys())}
↪")
      sys.exit()
  x_time = x.time.values[0]
  x = x['sss_smap']
  df_raw = x.to_dataframe().reset_index()
  df_raw['time'] = x_time
```

(continues on next page)

(continued from previous page)

```
    x.close()
    df = dc.add_day_week_month_year_clim(df_raw)
    df = df[['time','lat','lon','sss_smap','year','month','week','dayofyear']]
    df = df.sort_values(["time", "lat","lon"], ascending = (True, True,True))
    df = dc.mapTo180180(df)
    if df.dtypes.to_dict() != test_dtype:
        print(f"Check data types in {fil}. New: {df.columns.to_list()}")
        sys.exit()
    fil_name = os.path.basename(fil)
    fil_date = df['time'][0].strftime("%Y_%m_%d")
    path = f"{nrt_folder.split('vault/')[1]}{tbl}_{fil_date}.parquet"
    df.to_parquet(f"{nrt_folder}{tbl}_{fil_date}.parquet", index=False)
    metadata.tblProcess_Queue_Process_Update(fil_name, path, tbl, 'Opedia', 'Rainier')
    s3_str = f"aws s3 cp {tbl}_{fil_date}.parquet s3://cmap-vault/observation/remote/
↪satellite/{tbl}/nrt/"
    os.system(s3_str)
    metadata.tblIngestion_Queue_Staged_Update(path, tbl, 'Opedia', 'Rainier')
    a = [df,df,df]
    b = [tbl,tbl,tbl]
    c = ['mariana','rossby','rainier']
    with Pool(processes=3) as pool:
        result = pool.starmap(DB.toSQLbcp_wrapper, zip(a,b,c))
        pool.close()
        pool.join()
```

CHAPTER

ELEVEN

USER SUBMITTED DATASET WALKTHROUGH

This example should walk you through the steps of ingesting a user submitted dataset into the database.

For this example, we are going to be using the dataset: **Falkor_2018 - 2018 SCOPE Falkor Cruise Water Column Data**

11.1 Removal of Previously Existing Dataset

This dataset was an early submitted dataset and has recently been revised to bring it up to line with the current CMAP data submission guidelines. The dataset was reviewed by CMAP data curators, which means the finalized updated dataset will be found in the /final folder: **Dropbox/Apps/<dataset_short_name>/final**

Because this dataset already exists in the database, we must first remove the old version.

To do this, we can use some of the functionality in cmapdata/ingest/metadata.py

By calling this function **deleteCatalogTables(tableName, db_name, server)**, we can remove any metadata and data tables from a given server.

> **Warning:** This function has drop privileges! Make sure you want to wipe the dataset metadata and table.

```
python metadata.py
```

```
>>> deleteCatalogTables('tblFalkor_2018','Rainier')
```

Continue this function for any other existing servers. ex. 'Mariana', 'Rossby'

If only the metadata needs updating, calling the function **deleteTableMetadata(tableName, db_name, server)** will remove all metadata associated with the dataset, but will not delete the data table.

11.2 Specifying the Ingestion Arguments

Using ingest/general.py, you can pass command line arguments to specify which server you wish to add the dataset to as well as including a DOI. Because the DOI is a CMAP template, the optional flag -t is not necessary to include as the default is true.

Navigate to the ingest/ submodule of cmapdata. From there, run the following in the terminal.

```
python general.py {table_name} {branch} {filename} {-d} {DOI link} {-l} {DOI download␣
↪link} {-f} {DOI file name} {-S} {server}
```

- {**table_name**}: Table name for the dataset. Must start with prefix "tbl". Ex. tblFalkor_2018
- {**branch**}: Branch where dataset should be placed in Vault. Ex's: cruise, float, station, satellite, model, assimilation
- {**filename**}: Base file name in vault/staging/combined/. Ex.: 'Falkor_2018.xlsx'
- {**-d**}: Optional flag for including DOI with dataset in tblReferences. DOI link string follows flag arg.
- {**DOI link**}: String for full web address of CMAP specific DOI. Ex. "https://doi.org/10.5281/zenodo.5208854"
- {**-l**}: Optional flag for including the DOI download link in tblDataset_DOI_Download. DOI dowload link string follows flag.
- {**DOI download link**}: String for DOI download link of CMAP specific DOI. Ex. "https://zenodo.org/record/5208854/files/tblFalkor_2018%20%282%29.xlsx?download=1"
- {**-f**}: Optional flag for DOI file name. DOI file name string follows flag.
- {**DOI file name**}: String for filename of CMAP specific DOI. Ex. "tblFalkor_2018 (2).xlsx"
- {**-t**}: Optional flag for denoting if DOI is a web validator template. Default value is 1.
- {**DOI in CMAP template**}: Boolean if DOI is a web validator template.
- {**-S**}: Required flag for specifying server choice. Server name string follows flag.
- {**server**}: Valid server name string. Ex. "Rainier", "Mariana" or "Rossby"

An example string for full ingest would be:

```
python general.py tblFalkor_2018 cruise 'Falkor_2018.xlsx' -d 'https://doi.org/10.5281/
↪zenodo.5208854' -l 'https://zenodo.org/record/5208854/files/tblFalkor_2018%20%282%29.
↪xlsx?download=1' -f 'tblFalkor_2018 (2).xlsx' -S 'Rainier'
```

There are two additional arguments if you are only updating the metadata, and not reingesting the data table.

- {**-U**}: Optional flag for specifying updates to metadata only
- {**-F**}: Optional boolean depth flag. Default value is 0. Set to 1 if the data has a depth column

An example string for metadata update would be:

```
python general.py tblFalkor_2018 cruise 'Falkor_2018.xlsx' -d 'https://doi.org/10.5281/
↪zenodo.5208854' -l 'https://zenodo.org/record/5208854/files/tblFalkor_2018%20%282%29.
↪xlsx?download=1' -f 'tblFalkor_2018 (2).xlsx' -S 'Rainier' -F 1 -U
```

Behind the scenes, the script is doing:

1. Parsing the user supplied arguments.
2. Creates **vault/** folder structure, syncs Dropbox, and transfers the files to **vault/**.
3. Splitting the data template into data, dataset_metadata and vars_metadata files. Saves metadata as parquet files in /metadata folder.
4. Importing into memory the data, dataset_metadata and vars_metadata sheets as pandas dataframes.
5. Checks for presence of variables describing organism abundance based on units, short name, and long name. If Org_ID is present in the variable metadata sheet, checks that Conversion_Coefficient aligns with units.
6. Checks values in data for outliers (+/- five times standard deviation) or invalid ranges for lat, lon, depth.
7. Creating a suggested SQL table and index based on the infered data.
8. Insert data into newly created table.

9. Insert metadata into various metadata tables, match cruises and classify ocean region(s).
10. Create summary stats and insert into tblDataset_Stats.
11. Add server alias to tblDataset_Servers.
12. Create dataset icon and push to github.

Once the ingestion completes without any errors, check the catalog to see if the table is visable. It is also advisable to try to plot one or more variables, download the full datasets, as well as a subset.

Note: See the future code changes section for ideas on improvements.

CHAPTER

TWELVE

OUTSIDE SMALL DATASET WALKTHROUGH

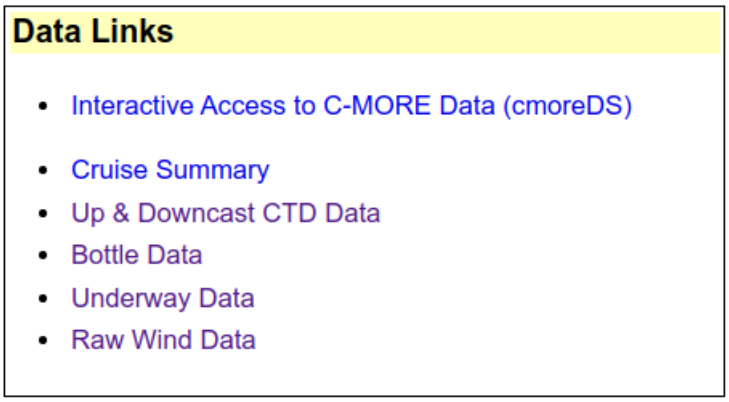
Datasets from outside sources do not follow CMAPs data guidelines and vary drastically in quality of metadata. Many dataset have variables that are not self explanatory and may have data flags or conventions that will take some work to understand. One of the best resources is the rest of the Armbrust lab, which has tons of oceanography domain knowledge and experience working with these datasets. Additionally, contacting the dataset owner/provider can also provide insight.

In this example, we are going to walkthrough collecting, processing and ingesting a small outside dataset into CMAP.

Recently, we wanted to add multiple datasets from the CMORE-BULA cruise that traveled from Hawaii south to Fiji. These data were available on the HOT (Hawaii Ocean Time-series) FTP site.

12.1 Collecting a small dataset from an FTP site using wget

From the UH CMORE-BULA site: https://hahana.soest.hawaii.edu/cmorebula/cmorebula.html there are links for data access. Each of these leads you to an FTP site.



The FTP site for the CMORE-BULA CTD dataset contains 46 individual CTD casts. Instead of downloading them all by hand, we can use some functionality from the wget command line tool.

To keep a record of dataset processing, it is a good idea to create a collect_{dataset_name}.py script in the collect/ submodule of cmapdata. For this example, we have created a directory with the tree structure:

```
├── insitu
    ├── cruise
        ├── misc_cruise
```

(continues on next page)

Index of /FTP/cmore/ctd/bula1

Name	Last modified	Size	Description
Parent Directory		-	
Readme.ctd	2008-12-24 10:43	2.0K	
bu1s1c1dn.ctd	2009-02-25 14:30	16K	
bu1s1c1up.ctd	2009-02-25 14:30	16K	
bu1s1c2dn.ctd	2009-02-25 14:30	82K	
bu1s1c2up.ctd	2009-02-25 14:30	81K	
bu1s1c3dn.ctd	2009-02-25 14:30	17K	
bu1s1c3up.ctd	2009-02-25 14:30	17K	
bu1s2c1dn.ctd	2009-02-25 14:30	16K	
bu1s2c1up.ctd	2009-02-25 14:30	17K	
bu1s2c2dn.ctd	2009-02-25 14:30	82K	
bu1s2c2up.ctd	2009-02-25 14:30	80K	
bu1s2c3dn.ctd	2009-02-25 14:30	17K	
bu1s2c3up.ctd	2009-02-25 14:30	17K	
bu1s3c1dn.ctd	2009-02-25 14:30	17K	
bu1s3c1up.ctd	2009-02-25 14:30	16K	
bu1s3c2dn.ctd	2009-02-25 14:30	81K	
bu1s3c2up.ctd	2009-02-25 14:30	80K	
bu1s3c3dn.ctd	2009-02-25 14:30	17K	
bu1s3c3up.ctd	2009-02-25 14:30	17K	

(continued from previous page)

```
            ├── KM0704_CMORE_BULA
            │   └── collect_KM0704_CMORE_BULA.py
```

For each dataset in this cruise (CTD, underway, underway sample, bottle & wind), we can collect any files using wget.

For CTD we can write a download function using wget and pass the FTP link for the CTD files. Using wget, we can specify the output directory.

```
odir = vs.cruise + "tblKM0704_CMORE_BULA/raw/"

def download_KM0704_data(outputdir, download_link):
    wget_str = f"""wget -P '{outputdir}' -np -R "'index.html*" robots=off -nH --cut-dirs␣
↪8 -r  {download_link}"""
    os.system(wget_str)

# #download ctd
download_KM0704_data(
    odir + "CTD/", "https://hahana.soest.hawaii.edu/FTP/cmore/ctd/bula1/"
)
```

For all the datasets in CMORE-BULA, this process would need to be repeated.

12.2 Processing a small dataset

Now that the CTD files have been collected, we can begin processing them. Once again, we are going to create a dataset/collection specific file for a record. In this example, we will call it process_KM0704_CMORE_BULA.py This should be placed in the **process/** submodule of cmapdata.

```
├── insitu
    ├── cruise
        ├── misc_cruise
            ├── KM0704_CMORE_BULA
            │   └── process_KM0704_CMORE_BULA.py
```

The full CTD processing steps can be found in this file, but in summary they are:

1. Use the glob library to create a list of all .ctd files collected previously.
2. Iterate thorough list
    - read csv into pandas dataframe
    - replace '-9' missing values with np.nan
    - extract station,cast,cast direction and num_observations from filename using string splitting.
    - create new columns of variable specific quality flags out of strange combined flag column.
    - drop unneeded columns
    - append data
3. concatenate cleaned data into Pandas Dataframe

Other datasets in the CMORE-BULA cruise required additional processing. Some required time and depth formatting. Others were missing spatio-temporal coordinates, which had to be collected using spatial or station/cast # joins to the other datasets.

Once all the data are processed, the can be exported to dropbox/../vault/observation/in-situ/cruise/tblKM0704_CMORE_BULA/raw where dataset and variable metadata sheets can be added. Once these are complete, the dataset should run through the validator. From here, follow the steps outlined in the **user submitted datasets walkthrough** section to ingest the data into the database.

CHAPTER

THIRTEEN

OUTSIDE LARGE DATASET WALKTHROUGH

Outside large datasets require similar collecting/processing methods to outside small datasets, however the ingestion strategy can differ. In this section, we will outline examples of collecting, processing and ingesting two large outside datasets.

13.1 Argo Float Walkthrough

The ARGO float array is a multi-country program to deploy profiling floats across the global ocean. These floats provide a 3D insitu ocean record. The CORE argo floats provide physical ocean parameters, while the BGC (Biogeochemical) specific floats provide Biogeochemical specific variables (nutrients, radiation etc.).

These Argo datasets are a part of the continuous ingestion project, but differ in process as each month will create a new table for each dataset.

13.1.1 Argo Data Collection

ARGO float data are distributed through two main DAAC's. Individual files can be accessed directly from FTP servers from each DAAC. Alternatively, a zipped file of all float records updated monthly can be found at: https://www.seanoe.org/data/00311/42182/. These are released on the 10th of every month.

To keep a record of the collection, we will create a collect_{dataset_name}.py file.

```
import vault_structure as vs
import os

def downloadArgo(newmonth, tar_url):
    """Download Argo tar file. Creates new vault tables based on newmonth stub
    Args:
        newmonth (string): Month and year of new data used as table suffix (ex. Sep2023)
        tar_url (string): URL pointing to tar download for newest data (ex. https://www.
↪seanoe.org/data/00311/42182/data/104707.tar.gz)
    """
    tbl_list = [f'tblArgoCore_REP_{newmonth}',f'tblArgoBGC_REP_{newmonth}']
    for tbl in tbl_list:
        vs.leafStruc(vs.float_dir+tbl)
    base_folder = f'{vs.float_dir}{tbl}/raw/'
    output_dir = base_folder.replace(" ", "\\ ")
    os.system(f"""wget --no-check-certificate {tar_url} -P {output_dir}""")
```

Argo float data and metadata from Global Data Assembly Center (Argo GDAC)

Dated	2021-07-15
Author (s)	Argo
Contributor (s)	Akazawa Fumihiko, Alraddadi Turki, Ananda Pascual, Andre Xavier, Arhan Michel, Atmadipoera Agus, Babin Marcel, Balan Sorin, Ballestero Daniel, Baringer Molly, Barre Nicolas, Beebeejaun M., Belbeoch Mathieu, Belchi Pedro Velez, Bellingham Clare, Bernard Vincent , Bittig Henry, Blain Stephane, Boebel Olaf, Boetius Antje, Boss Emmanuel, Bourles Bernard, Bower Amy, Boyer •••
DOI	10.17882 / 42182
Publisher	SEANOE
Keyword (s)	float, Argo, global ocean observing system, ocean circulation, in-situ, ocean pressure, sea water salinity, sea water temperature, multi-year, weather climate and seasonal observation, global-ocean
Abstract	Argo is a global array of 3,000 free-drifting profiling floats that measures the temperature and salinity of the upper 2000 m of the ocean. This allows, for the first time, continuous monitoring of the temperature, salinity, and velocity of the upper ocean, with all data being relayed and made publicly available within hours after collection. The array provides 100,000 temperature / salinity profiles and velocity measurements per year distributed over the global oceans at an average of 3-degree spacing. Some floats provide additional bio-geo parameters such as oxygen or chlorophyll. All data collected by Argo floats are publically available in near real-time via the Global Data Assembly Centers (GDACs) in Brest (France) and Monterey (California) after an automated quality control (QC), and in scientifically quality controlled form, delayed data mode,
Licence	CC BY
Use	A user of Argo data is expected to read and understand this manual and the documentation about the data contained in the "attributes" of the NetCDF data files, as these contain essential information about data quality and accuracy. A user should acknowledge use of Argo data in all publications and products where such data are used, preferably with the DOI and following standard sentence: "These data were collected and made freely available by the international Argo project and the national programs that contribute to it . "
User's manual	Argo documentation Argo user's manual
Data	ftp://ftp.ifremer.fr/ifremer/argo ftp://usgodae.org/pub/outgoing/argo ERDDAP

Data

Queue	Size	Format	Processing	Access	Key
2021-07-10	40 GB	NC, NetCDF	Quality controlled data	Open access	86141
2021-06-10	40 GB	NC, NetCDF	Quality controlled	Open access	85023

Click to download the data

DATA

Download metadata
TXT , RIS , XLS , RTF , BIBTEX

Equipments
Coriolis (In situ data for operational oceanography)

References
Johnson Gregory C., Hosoda Shigeki, Jayne Steven R., Oke Peter R., Riser Stephen C., Roemmich Dean, Suga Tohsio, Thierry Virginie, Wijffels Susan E., Xu Jianping Argo — Two Decades: Global Oceanography, Revolutionized. Annual Review of Marine Science IN

The raw data will be saved in **dropbox/../vault/observation/in-situ/float/tblArgoBGC_REP_{newmonth}/raw** This file will need to be unzipped, either using python or bash. The functions for doing so in Python are in process_ARGO_BGC_Sep2023.py

Once the data has been unzipped, there are four subfolders:

```
├── aux
├── dac
├── doc
└── geo
```

dac contains the data. Descriptions for the rest can be found in the argo data handbook (http://dx.doi.org/10.13155/29825).

The **dac** subfolder contains 11 daacs/distributors. Each of these contains zipped files.

To unzip and organize these by BGC and Core. The following scripts were run as part of **process_ARGO.py**

```
def unzip_and_organize_BGC():
    vs.makedir(argo_base_path + "BGC/")
    os.chdir(argo_base_path)
    for daac in tqdm(daac_list):
        os.system(
            f"""tar -xvf {daac}_bgc.tar.gz -C BGC/ --transform='s/.*\///' --wildcards --
↪no-anchored '*_Sprof*'"""
        )
```

A similar function is then run for the Core files.

13.1.2 Argo Data Processing

Once the data collection is complete, we can start processing each argo netcdf file. To keep a record, we will create a record in the **process/** submodule of cmapdata.

```
├── insitu
    ├── float
        ├── ARGO
            └── process_ARGO.py
```

Since BGC specific floats and Core floats contain different sets of variables, the processing has been split into two scripts.

Detailed processing steps for the argo core and bgc can be found in process_ARGO_BGC_Sep2023.py and process_ARGO_Core_Sep2023. The processing is done with Pool from the multiprocessing library. The rough processing logic is outlined below:

1. Use the glob library to create a list of all netcdf files in the BGC directory.
2. Iterate thorough list
    - import netcdf with xarray
    - decode binary xarray column data
    - export additional metadata cols for future unstructured metadata
    - drop unneeded metadata cols
    - checks no new columns are present this month

- convert xarray to dataframe and reset index
- add a depth specific column calculated from pressure and latitude using python seawater library
- rename Space-Time columns
- format datetime
- drop any duplicates create by netcdf multilevel index
- drop any invalid ST rows (rows missing time/lat/lon/depth)
- sort by time/lat/lon/depth
- add climatology columns
- reorder columns and add any missing columns
- replace any inf or nan string values with np.nan (will go to NULL in SQL server)
- strips any whitespace from string col values
- removes nan strings before setting data types
- checks there is data in dataframe before exporting parquet file to /rep folder

Because the data will only live on the cluster, the fastest way to calculate stats for such a large dataset is to aggregate the values from each processed parquet file. Once all NetCDF files have been processed and parquet files saved to /rep, the following steps are completed:

1. Read each parquet file into a pandas dataframe
2. Query the dataframe to remove space and time data flagged as "bad" (_QC = 4)
3. Calculate min/max for each field with describe()
4. Append min/max values for each file to a stats dataframe
5. Export stats dataframe to /stats directory to be used during dataless ingestion

Before passing off for ingestion to the cluster, run through each processed parquet file to ensure the schema matches across all files. Past errors have been caused by empty parquet files and empty columns in one profile that are string data types in other profiles. Reading a parquet file into a dataframe and checking for matches is not suffient as pandas can read data types differently than the cluster will. The most successful checks to date were completed using pyarrow and pyarrow.parquet.

> **Warning:** Any schema error in a single parquet file will cause the bulk ingestion to fail

The last step for all process scripts is to copy the GitHub URL for the script to the /code folder in the vault. The example below calls the metadata.export_script_to_vault function and saves a text file named "process" in the dataset's code folder in the vault.

```
metadata.export_script_to_vault(tbl,'float_dir',f'process/insitu/float/ARGO/process_Argo_
↪BGC_{date_string}.py','process.txt')
```

Bulk Ingestion to the Cluster

Due to the size of the Argo datasets, and the monthly creation of a new dataset, both Argo Core and Argo BGC only live on the cluster. After all parquet files are created and checked for matching schemas, a bulk ingestion will be done to create the new tables on the cluster.

Creating and Ingesting Metadata

Once the bulk ingest is complete on the cluster, the metadata can be added. All dataset ingestion using general.py (see cruise ingestion for differences) pulls metadata from a folder named "final" within the validator folders in DropBox. For large datasets, you will still need to submit a template to the validator. In order to pass the validator tests you will need to include a minimum of one row of data in the data sheet. The values can all be placeholders, but must contain some value.

If no new variables have been added, the data curation team does not need to re-run the QC API. Use the last month's metadata for Argo and update the **dataset_meta_data** tab with new values for dataset_short_name, dataset_long_name, dataset_version, dataset_release_date, and dataset_references. In the **vars_meta_data** tab, replace old references of dataset names in the variable keywords to current month. These keywords are usually assigned by the QC API.

After submitting through the validator, create a folder named final in **dropbox../Apps/Simons CMAP Web Data Sunmission/ARGO_BGC_Sep2023** and copy the submitted template into /final for ingestion.

To ingest the metadata only, you can use ingest/general.py

Navigate to the ingest/ submodule of cmapdata. From there, run the following in the terminal. Because the DOI for the Argo datasets is already in the references column in the **dataset_meta_data** tab of the metadata template, you do not need to use the {-d} flag with ingestion.

```
python general.py {table_name} {branch} {filename} {-S} {server} {-a} {data_server} {-i}
↪{icon_filename} {-F} {-N}
```

- {**table_name**}: Table name for the dataset. Must start with prefix "tbl". Ex. tblArgoBGC_REP_Sep2023
- {**branch**}: Branch where dataset should be placed in Vault. Ex's: cruise, float, station, satellite, model, assimilation
- {**filename**}: Base file name in vault/staging/combined/. Ex.: 'global_diazotroph_nifH.xlsx'
- {**-S**}: Required flag for specifying server choice for metadata. Server name string follows flag.
- {**server**}: Valid server name string. Ex. "Rainier", "Mariana" or "Rossby"
- {**-a**}: Optional flag for specifying server name where data is located
- {**data_server**}: Valid server name string. Ex. "Rainier", "Mariana", "Rossby", or "Cluster"
- {**-i**}: Optional flag for specifying icon name instead of creating a map thumbnail of the data
- {**icon_filename**}: Filename for icon in Github instead of creating a map thumbnail of data. Ex: argo_small.jpg
- {**-F**}: Optional flag for specifying a dataset has a valid depth column. Default value is 0
- {**-N**}: Optional flag for specifying a 'dataless' ingestion or a metadata only ingestion.

An example string for the September 2023 BGC dataset is:

```
python general.py tblArgoBGC_REP_Sep2023 float 'ARGO_BGC_Sep2023.xlsx' -i 'argo_small.jpg
↪' -S 'Rossby' -N -a 'cluster' -F 1
```

Removing Old Argo Data

Once a new month of Argo data is accessible on the CMAP website, a previous month can be retired. The current plan is to keep three months of Argo data available to users. For example, with the addition of September data, the June data can be deleted from the vault. The data needs to be removed from the cluster **before** the parquet files are deleted from Dropbox. Metadata for the retired month can be removed from the CMAP catalog before the parquet files are deleted, if need be. This can be done using the following function in metadata.py:

CHAPTER

FOURTEEN

GEOTRACES SEAWATER WALKTHROUGH

The Geotraces Seawater dataset is unique for two reasons: 1. The data table contains >1,024 columns which is the max allowed by SQL Server 2. The data producers requested the inclusion of unstructured metatdata

14.1 Geotraces Overview

Geotraces IDP2021v2 contains 5 datasets: Seawater, Sensor, Precipitation, Aerosols, and Cryosphere. Only Seawater and Aerosols have updated data in v2, but the download link includes the v1 data for the remaining 3 datasets. Based on discussions with the Geotraces data team, there are no plans to update the remaining 3 datasets to v2. The second version includes additional cruises and fixes to multiple errors found in v1. The full list of fixes can be found here: https://www.bodc.ac.uk/geotraces/data/idp2021/documents/geotraces_idp2021v2_changes.pdf

14.1.1 Geotraces Data Collection

A zipped file of all five GEOTRACES datasets can be found on BODC: https://www.bodc.ac.uk/data/published_data_library/catalogue/10.5285/ff46f034-f47c-05f9-e053-6c86abc0dc7e/

The raw data will be saved in **dropbox/../vault/observation/in-situ/cruise/tblGeotraces_{dataset_name}/raw** These files will need to be unzipped, saving to each dataset's raw folder. Copies of the included PDFs are moved to the /metadata folder. The script to download the data and unzip the raw data into each of the 5 dataset folders in the vault is here: ../cmapdata/collect/insitu/cruise/GEOTRACES/collectGeotraces_v2.py

In addition to a NetCDF for each dataset, Geotraces also includes an infos folder. Within that folder is a static html file for each cruise and variable combination that metadata was submitted for. These are scraped to populate the variable-level unstructured metadata (UM) which is described in detail below.

14.1.2 Geotraces Seawater Processing

Detailed processing steps for the Seawater dataset can be found in process/insitu/cruise/GEOTRACES/process_Geotraces_Seawater_IDP2021v2.py. The rough processing logic is outlined below:

- import netcdf with xarray
- loop through netcdf metadata and export to Geotraces_Seawater_Vars.xlsx to build out the vars_metadata sheet for validator
- decode binary xarray column data
- change flag values from ASCII to int / string values based on Geotraces request
- rename depth field as it contains nulls

- build suggested SQL table based on netcdf data types
- convert xarray to dataframe and reset index
- add a depth specific column calculated from pressure and latitude using python seawater library
- rename Space-Time columns
- format datetime
- map longitude values from 0, 360 to -180, 180
- drop any invalid ST rows (rows missing time/lat/lon/depth)
- reorder columns and sort by time/lat/lon
- create two temp tables in SQL
- split dataframe in two, retaining the initial 17 metadata columns in both
- ingest split dataframes into two temp tables
- insert into final table with column set
- drop two temporary tables

Ingestion to the Database

Geotraces_Seawater_IDP2021v2 includes 1186 variables which exceeds the limit SQL server allows to be held within a single table. Due to the nature of how the data is organized, much of the data is sparse, as not every cruise collected data for each variable.

An example of the syntax to create a column set for sparse columns is shown below. Not all columns in the full create table script are show here.

```
CREATE TABLE [dbo].[tblGeotraces_Seawater_IDP2021v2](
    [time] [datetime] NOT NULL,
    [lat] [float] NOT NULL,
    [lon] [float] NOT NULL,
    [N_SAMPLES] [int] NULL,
    [N_STATIONS] [int] NULL,
    [cruise_id] [nvarchar](6) NULL,
    [station_id] [nvarchar](26) NULL,
    [station_type] [nvarchar](1) NULL,
    [Bot__Depth] [float] NULL,
    [Operator_s_Cruise_Name] [nvarchar](13) NULL,
    [Ship_Name] [nvarchar](20) NULL,
    [Period] [nvarchar](23) NULL,
    [Chief_Scientist] [nvarchar](31) NULL,
    [GEOTRACES_Scientist] [nvarchar](77) NULL,
    [Cruise_Aliases] [nvarchar](14) NULL,
    [Cruise_Information_Link] [nvarchar](75) NULL,
    [BODC_Cruise_Number] [float] NULL,
    [CSet] [xml] COLUMN_SET FOR ALL_SPARSE_COLUMNS  NULL,
    [CTDPRS_T_VALUE_SENSOR] [float] SPARSE  NULL,
    [CTDPRS_T_VALUE_SENSOR_qc] [nvarchar](1) SPARSE  NULL,
    [DEPTH_SENSOR] [float] SPARSE  NULL,
    [DEPTH_SENSOR_qc] [nvarchar](1) SPARSE  NULL,
```

(continues on next page)

(continued from previous page)

```
    ...
    ) ON [FG3]
```

In order to successfully ingest into the column set, the data needs to be unioned from the two temp tables described above into the full table. BCP directly into the table with the column set fails.

Add New Cruises

The first version of Geotraces included data for multiple cruises that were not in CMAP. See the cruise ingestion for details on the template specific for adding cruise metadata and trajectories. Most US-based cruises can be found on the following websites:

- R2R: https://www.rvdata.us/browse_vessels
- SAMOS: https://samos.coaps.fsu.edu/html/cruise_data_availability.php
- BODC: https://www.bodc.ac.uk/data/bodc_database/nodb/search/
- UNOLS: https://strs.unols.org/Public/Search/diu_ships.aspx

It is always preferred to use navigation or underway data for a cruise trajectory. In rare cases this data is not publicly available and sample locations can be used instead. Geotraces provides an API endpoint for sample locations that is "live (or close to) dynamically created data from Geotraces databases".

The collection script for the new cruises in v2 can be found here: cmap-data/collect/insitu/cruise/GEOTRACES/collectGeotraces_sample_locations_v2.py

The processing script for the the new cruises can be found here: cmap-data/process/insitu/cruise/GEOTRACES/process_Geotraces_trajectories_v2.py

The processing script creates the excel template needed for cruise trajectory ingestion. The metadata details for each cruise was pulled from IDP2021v2_Cruises.pdf, which is included in download provided by Geotraces. The final template is saved to the vault here: ../vault/r2r_cruise/{cruise_name}/raw/{cruise_name}_cruise_meta_nav_data.xlsx

An example ingestion string for a new cruise is:

```
python general.py "SAG25_cruise_meta_nav_data.xlsx" -C SAG25 -S "Rossby" -v True
```

The {-v} flag tells the ingestion script to look in the raw folder of the vault, instead of pulling from Apps validator folder.

Creating and Ingesting Metadata

The Geotraces NetCDFs contain metadata that can be used to build out the vars_meta_data sheet for validator submission. This includes variable short name (Geotraces requested we maintain their variable short names), long names, and flag values. Below is syntax used to loop through the variables and create an initial spreadsheet of provided metadata:

```
tbl = 'tblGeotraces_Seawater_IDP2021v2'
meta_folder = f'{vs.cruise}{tbl}/metadata/'
n = f'{vs.cruise}{tbl}/raw/GEOTRACES_IDP2021_Seawater_Discrete_Sample_Data_v2.nc'
x = xr.open_dataset(n)

d1 = {'var_name':[], 'std_name':[], 'long_name':[], 'dtype':[], 'units':[], 'comment':[],
↪ 'flag_val':[], 'flag_def':[]}
df_varnames = pd.DataFrame(data=d1)
```

(continues on next page)

(continued from previous page)

```
for varname, da in x.data_vars.items():
    dtype = da.data.dtype
    if 'flag_values' in da.attrs.keys():
        fl_val = da.attrs['flag_values'].tolist()
        fl_def = da.attrs['flag_meanings']
    else:
        fl_val = None
        fl_def = None
    if 'long_name' in da.attrs.keys():
        long_name = da.attrs['long_name']
    else:
        long_name = None
    if 'standard_name' in da.attrs.keys():
        std_name = da.attrs['standard_name']
    else:
        std_name = None
    if 'comment' in da.attrs.keys():
        comment = da.attrs['comment']
    else:
        comment = None
    if 'units' in da.attrs.keys():
        units = da.attrs['units']
    else:
        units = None

    d1 = {'var_name':[varname], 'std_name':[std_name], 'long_name':[long_name], 'dtype
↪':[dtype], 'units':[units], 'comment':[comment], 'flag_val':[fl_val], 'flag_def':[fl_
↪def]}
    temp_df = pd.DataFrame(data=d1)
    df_varnames = df_varnames.append(temp_df, ignore_index=True)

df_varnames.to_excel(meta_folder +'Geotraces_Seawater_Vars.xlsx', index=False)
```

With so many variable names, a significant amount of additional work is needed to clean up this initial spreadsheet. All variable long names should be title case. Specifically for Geotraces, there were many instances when the metadata in the NetCDF was truncated. Holes were filled in by referring to the relevant static HTML files included in the infos folder, later used to scrape for UM.

All dataset ingestion using general.py (see cruise ingestion for differences) pulls metadata from a folder named "final" within the validator folders in DropBox. For large datasets, you will still need to submit a template to the validator. In order to pass the validator tests you will need to include a minimum of one row of data in the data sheet. The values can all be placeholders, but must contain some value. After the data curation team run the QC API to add the necessary keywords, they will include the finalized template to Apps/Geotraces_Seawater_IDP2021v2/final.

To ingest the metadata only, you can use ingest/general.py

Navigate to the ingest/ submodule of cmapdata. From there, run the following in the terminal. Because the DOI for the Argo datasets is already in the references column in the **dataset_meta_data** tab of the metadata template, you do not need to use the {-d} flag with ingestion.

```
python general.py {table_name} {branch} {filename} {-S} {server} {-a} {data_server} {-i}
↪{icon_filename} {-F} {-N}
```

- {**table_name**}: Table name for the dataset. Must start with prefix "tbl". Ex. tblArgoBGC_REP_Sep2023
- {**branch**}: Branch where dataset should be placed in Vault. Ex's: cruise, float, station, satellite, model, assimilation
- {**filename**}: Base file name in vault/staging/combined/. Ex.: 'global_diazotroph_nifH.xlsx'
- {**-S**}: Required flag for specifying server choice for metadata. Server name string follows flag.
- {**server**}: Valid server name string. Ex. "Rainier", "Mariana" or "Rossby"
- {**-i**}: Optional flag for specifying icon name instead of creating a map thumbnail of the data
- {**icon_filename**}: Filename for icon in Github instead of creating a map thumbnail of data. Ex: argo_small.jpg
- {**-F**}: Optional flag for specifying a dataset has a valid depth column. Default value is 0
- {**-N**}: Optional flag for specifying a 'dataless' ingestion or a metadata only ingestion.

An example string for the September 2023 BGC dataset is:

```
python general.py tblGeotraces_Seawater_IDP2021v2 cruise 'Geotraces_Seawater_IDP2021v2.
↪xlsx' -i 'tblGeotraces_Sensor.jpg' -S 'Rossby' -N
```

Creating and Ingesting Unstructured Metadata

The unstructured metadata (UM) for v1 of Geotraces Seawater was provided by Jesse McNichol. As there were additional cruises and variables in v2, a new set of UM needed to be scraped from the static HTML files included in the Geotraces data download.

The file naming convention for the HTML files is {cruise_name}_{variable_short_name}.html

The files were scraped using BeautifulSoup. The cruise name and variable name were parsed from the file name. The Geotraces data team requested we include the BODC documentation links on methods for each variable and cruise. Additional links to cruise information are also included in the html files, but were not requested. These can be added to the future iteration of the cruise page if the new designs include UM for cruises.

For details on the unstructured metadata project see Jira the following tickets: (https://simonscmap.atlassian.net/browse/CMAP-563, https://simonscmap.atlassian.net/browse/CMAP-572). Each unstructured metadata object includes a value array and a description array. Values and descriptions are always arrays, even if empty or single values. Also, these arrays must always have identical lengths, even if descriptions are empty strings. Descriptions are meant to be human readable, short descriptions akin to alt-text for an image online. A single variable may have multiple entries in tblVariables_JSON_Metadata. An example of a variable-level unstructured metadata is:

```
{"cruise_names":{"values":["PS71"],"descriptions":["Operators Cruise Name"]},"meta_links
↪":{"values":["https://www.bodc.ac.uk/data/documents/nodb/285421/"],"descriptions":[
↪"BODC documentation link"]}}
```

The script for creating UM for Geotraces Seawater is here: ..cmapdata/process/insitu/cruise/GEOTRACES/scrape_Geotraces_Seawater_U

Only one entry was requested by the Geotraces data team for dataset level metadata:

```
{"publication_link":{"values":["https://www.geotraces.org/geotraces-publications-
↪database/"],"descriptions":["Link to database of GEOTRACES publications"]}}
```

The dataset-level UM is ingested in the scrape script using DB.toSQLpandas(). The variable-level UM is ingested using DB.toSQLbcp_wrapper(), though requires a final update to fix BCP including additional quotes, causing the JSON to no longer be valid:

```
qry = """UPDATE tblVariables_JSON_Metadata SET json_metadata =␣
↪replace(replace(replace(json_metadata,'"""','"'), '"{','{'), '}"','}')"""
DB.DB_modify(qry, server)
```

You can check for invalid JSON in tblVariables_JSON_Metadata and tblDatasets_JSON_Metadata with the following:

```
SELECT * FROM tblVariables_JSON_Metadata WHERE ISJSON(JSON_Metadata) = 0
```

CHAPTER

FIFTEEN

MESOSCALE EDDY DATA WALKTHROUGH

15.1 Mesoscale Eddy Version History

The first version of Mesoscale Eddy data in CMAP was v2.0 provided by AVISO. Upon the release of v3.2, the version in CMAP was no longer being updated by AVISO and recommended to discontinue the use of v2 (https://ostst.aviso.altimetry.fr/fileadmin/user_upload/OSTST2022/Presentations/SC32022-A_New_Global_Mesoscale_Eddy_Trajectory_Atlas_Derived_from_Altimetry___Presentation_and_Future_Evolutions.pdf).

A video descibing the differences in versions can be found here: https://www.youtube.com/watch?v=4Vs3ZJNMViw

When data providers reprocess historic data so that it no longer aligns with data already ingested in CMAP, a new dataset for CMAP needs to be created. In this case, since AVISO provided a new name to their dataset (v3.2), the change did not fall under CMAP's internal change log naming convention (see continuous ingestion section for more details). AVISO's documentation notes updates are done "multiple times a year", and they create a new DOI for each temporal extension. They don't change their naming convention with temporal extensions, so any updates will result in a new change log table name.

While v2.0 consisted of one dataset, v3.2 included two datasets (allsat/twosat) are different enough such that AVISO decided to create two releases. Within each of these two datasets there is data for three types of eddies (long, short, untracked) describing the lifetime of an eddy, each split into two NetCDF files (cyclonic, anticyclonic). The 12 provided NetCDF files were ingested into 6 datasets described below.

15.1.1 Mesoscale Eddy Data Collection

AVISO provides data download through their FTP service which requires a login.

In order to keep all raw data in their respective folders, the first step is to create the 6 new dataset folders in the vault:

```
from ingest import vault_structure as vs

tbl = 'tblMesoscale_Eddy_'
sat_list = ['twosat','allsat']
ln_list = ['untracked', 'short', 'long']
for sat in sat_list:
    for ln in ln_list:
        vs.leafStruc(vs.satellite+tbl+sat+'s_'+ln)
```

Each individual NetCDF is then downloaded into the corresponding dataset's /raw folder in the vault:

```
cyc_list = ['Cyclonic', 'Anticyclonic']
for sat in sat_list:
    for ln in ln_list:
```

(continues on next page)

(continued from previous page)

```
        base_folder = f'{vs.satellite}{tbl}{sat}s_{ln}/raw/'
        print(base_folder)
        os.chdir(base_folder)
        for cyc in cyc_list:
            url_base = f"ftp://dharing@uw.edu:NBxOn4@ftp-access.aviso.altimetry.fr/value-
↪added/eddy-trajectory/delayed-time/META3.1exp_DT_{sat}/META3.1exp_DT_{sat}_{cyc}_{ln}_
↪19930101_20200307.nc"
            urllib.request.urlretrieve(url_base, base_folder+f'META3.1exp_DT_{sat}_{cyc}_
↪{ln}_19930101_20200307.nc')
```

15.1.2 Mesoscale Eddy Processing

Each NetCDF files includes 50 samples per observation. The lat and lon of the centroid don't change per sample and most have the same values within an observation. These variables have small differences across samples in an observation: effective_contour_lat, effective_contour_longitude, speed_contour_latitude, speed_contour_longitude, uavg_profile. Due to duplication of lat/lon and small varations across a subset of variables, each dataset was subset for sample = 0.

The processing logic for each NetCDF is outlined below:

- import netcdf with xarray, selecting where NbSample = 0
- loop through netcdf metadata and export to AVISO_Eddy32_allsats_Vars.xlsx to build out the vars_metadata sheet for validator
- call SQL.full_SQL_suggestion_build() to create SQL tables
- loop through both NetCDFs per dataset
- rename lat and lon, drop obs field
- add column eddy_polarity based on Anticyclonic vs Cyclonic file
- add climatology fields
- map longitude values from 0, 360 to -180, 180
- ingest with DB.toSQLbcp_wrapper()

15.1.3 Creating and Ingesting Metadata

All dataset ingestion using general.py (see cruise ingestion for differences) pulls metadata from a folder named "final" within the validator folders in DropBox. For large datasets, you will still need to submit a template to the validator. In order to pass the validator tests you will need to include a minimum of one row of data in the data sheet. The values can all be placeholders, but must contain some value. After the data curation team run the QC API to add the necessary keywords, they will include the finalized template to Apps/Mesoscale_Eddy_*/final.

To ingest the metadata only, you can use ingest/general.py

Navigate to the ingest/ submodule of cmapdata. From there, run the following in the terminal. Because the DOI for the Mesoscale Eddy datasets is already in the references column in the **dataset_meta_data** tab of the metadata template, you do not need to use the {-d} flag with ingestion.

```
python general.py {table_name} {branch} {filename} {-S} {server} {-a} {data_server} {-i}
↪{icon_filename} {-F} {-N}
```

- {**table_name**}: Table name for the dataset. Must start with prefix "tbl". Ex. tblArgoBGC_REP_Sep2023

- {**branch**}: Branch where dataset should be placed in Vault. Ex's: cruise, float, station, satellite, model, assimilation
- {**filename**}: Base file name in vault/staging/combined/. Ex.: 'global_diazotroph_nifH.xlsx'
- {**-S**}: Required flag for specifying server choice for metadata. Server name string follows flag.
- {**server**}: Valid server name string. Ex. "Rainier", "Mariana" or "Rossby"
- {**-i**}: Optional flag for specifying icon name instead of creating a map thumbnail of the data
- {**icon_filename**}: Filename for icon in Github instead of creating a map thumbnail of data. Ex: argo_small.jpg
- {**-F**}: Optional flag for specifying a dataset has a valid depth column. Default value is 0
- {**-N**}: Optional flag for specifying a 'dataless' ingestion or a metadata only ingestion.
- {**-v**}: Optional flag denoting if metadata template is present in the raw folder of the vault
- {**in_vault**}: If True, pulls template from vault. Default is False, which pulls from /final folder in Apps folder created after submitting to the validator

These datasets were ingested before the QC API was written. The use of the vault flag for datasets should no longer be used as all metadata should go through the API, at minimum for the automatic addition of all the keywords.

An example string used for a Mesoscale Eddy dataset is:

```
python general.py tblMesoscale_Eddy_allsats_long satellite 'tblMesoscale_Eddy_allsats_
↪long.xlsx' -i 'chelton_aviso_eddy.png' -S 'Rossby' -v True -N
```

CHAPTER

SIXTEEN

INGESTING CRUISE METDATA AND TRAJECTORY

CMAP contains cruise trajectories and metadata stored in seperate tables (tblCruise and tblCruise_Trajectory). These allow us to visualize cruise tracks on map, colocalize datasets with cruise tracks and links datasets to specific cruises.

A cruise ingestion template should contain two sheets. One for cruise metadata and another for cruise trajectory.

16.1 Metadata Sheet

Nickname	Name	Ship_Name	Chief_Name	Cruise_Series
< Ship Nickname (ex. Gradients 3) >	< UNOLS Cruise Name (ex. KM1906) >	< Official Ship Name (ex. Kilo Moana) >	< Chief Scientist Name (ex. Ginger Armbrust) >	< opt. Cruise Series (ex. Gradients/HOT/etc.) >

The metadata sheet contains cruise metadata that will populate tblCruise. The ST bounds will be filled in with the ingestion process.

16.2 Trajectory Sheet

time	lat	lon
< Format %Y-%m-%dT%H:%M:%S, Time-Zone: UTC, example: 2014-02-28T14:25:55 >	< Format: Decimal (not military grid system), Unit: degree, Range: [-90, 90] >	< Format: Decimal (not military grid system), Unit: degree, Range: [-180, 180] >

The trajectory sheet contains ST information of the cruise. This should have enough points to give an accurate cruise trajectory, without having too high a sampling interval. A good target might be minute scale.

16.3 Ingesting Cruise Templates

Similar to how datasets are ingested into CMAP, we can use the functionallity in the **ingest** subpackage.

Completed crusie templates should start the ingestion process in '/CMAP Data Submission Dropbox/Simons CMAP/vault/r2r_cruise/{cruise_name}/{cruise_name_template.xlsx}'

Using ingest/general.py, you can pass command line arguments to specify a cruise ingestion as well as a server.

Navigate to the ingest/ submodule of cmapdata. From there, run the following in the terminal.

```
python general.py {filename} {-C} {cruise_name} {-S} {server}
```

- {**filename**}: Base file name in vault/staging/combined/. Ex.: 'TN278_cruise_meta_nav_data.xlsx'
- {**-C**}: Flag indicating for cruise ingestion. Follow with cruise_name.
- {**cruise_name**}: String for official (UNOLS) cruise name Ex. TN278
- {**-S**}: Required flag for specifying server choice. Server name string follows flag.
- {**server**}: Valid server name string. Ex. "Rainier", "Mariana" or "Rossby"
- {**-v**}: Optional flag denoting metadata template is present in the raw folder of the vault
- {**in_vault**}: If True, pulls template from vault. Default is False, which pulls from /final folder in Apps folder created after submitting to the validator

An example string would be:

```
python general.py 'TN278_cruise_meta_nav_data.xlsx' -C TN278 -S "Rainier" -v True
```

Behind the scenes, the script is doing:

1. parsing the user supplied arguments.
2. Splitting the data template into cruise_metadata and cruise_trajectory files.
3. Importing into memory the cruise_metadata and cruise_trajectory sheets as pandas dataframes.
4. Filling in the ST bounds for the cruise_metdata dataframe with min/max's from the trajectory dataframe.
5. Inserting the metadata dataframe into tblCruise.
6. Inserting the trajectory dataframe into tblCruise_Trajectory.
7. Using the trajectory dataframe to classify the cruise by ocean region(s).
8. Inserting the cruise_ID and region_ID's into tblCruise_Regions.

CHAPTER

SEVENTEEN

DB

The repository DB (https://github.com/simonscmap/DB) under the Simons CMAP github page stores all of the SQL Server table creation scripts.

This repo contains subdirectories for db creation, stored procedures (usp), user-defined functions(udf) and data and metadata tables.

The structure with examples is outlined below:

```
├── db
│   └── db.sql
├── tables
│   ├── api
│   │   └── tblFront_End_Errors.sql
│   ├── core
│   │   ├── cruise
│   │   │   └── tblCruise_Trajectory.sql
│   │   └── tblVariables_update.sql
│   ├── model
│   │   └── tblPisces_NRT.sql
│   ├── observation
│   │   └── tblWOA_Climatology.sql
│   └── satellite
│       └── tblWind_NRT.sql
├── udf
│   └── udfVariableMetaData.sql
└── usp
    └── uspWeekly.sql
```

17.1 Custom Table Creation

- FG's, columns, datatypes

17.2 Indexing Strategy

- **indices (time/lat/lon depth) climatology etc.**
  -how these relate to performance

CHAPTER

EIGHTEEN

COLLECT

This submodule contains various scripts for collecting outside data. Methods include FTP, curl, wget and others. Scripts are usually one-off and are a record of the method used to collect the data. They are organized hierarchically in a similar fashion to **/vault**.

```
├── assimilation
├── model
├── observation
    ├── in-situ
    |   ├── cruise
    |   |   |── cruise_name
    |   |       |── collect_{cruise_name}.py
    |   ├── drifter
    |   ├── float
    |   ├── mixed
    |   └── station
    └── remote
        └── satellite
```

18.1 collection strategies

The oceanography data in CMAP comes from multiple sources which vary in the amount of data processing required and available metadata. The first step of ingesting a dataset from an outside source into CMAP is collecting the data. This generally starts with a python collection script. This both servers to collect the data as well as leave a record.

18.2 FTP Servers

Some datasets, especially when there are multiple files, are available over FTP servers. To retrieve this data, you can either use some GUI FTP application such as FileZilla or a command line utility such as wget or curl. Examples of using wget are available in some of the collect.py scripts. Some FTP sites required registrations and username/passwords.

Index of /FTP/cmore/ctd/bula1

Name	Last modified	Size	Description
Parent Directory		-	
Readme.ctd	2008-12-24 10:43	2.0K	
bu1s1c1dn.ctd	2009-02-25 14:30	16K	
bu1s1c1up.ctd	2009-02-25 14:30	16K	
bu1s1c2dn.ctd	2009-02-25 14:30	82K	
bu1s1c2up.ctd	2009-02-25 14:30	81K	
bu1s1c3dn.ctd	2009-02-25 14:30	17K	
bu1s1c3up.ctd	2009-02-25 14:30	17K	
bu1s2c1dn.ctd	2009-02-25 14:30	16K	
bu1s2c1up.ctd	2009-02-25 14:30	17K	
bu1s2c2dn.ctd	2009-02-25 14:30	82K	
bu1s2c2up.ctd	2009-02-25 14:30	80K	
bu1s2c3dn.ctd	2009-02-25 14:30	17K	
bu1s2c3up.ctd	2009-02-25 14:30	17K	
bu1s3c1dn.ctd	2009-02-25 14:30	17K	
bu1s3c1up.ctd	2009-02-25 14:30	16K	
bu1s3c2dn.ctd	2009-02-25 14:30	81K	
bu1s3c2up.ctd	2009-02-25 14:30	80K	
bu1s3c3dn.ctd	2009-02-25 14:30	17K	
bu1s3c3up.ctd	2009-02-25 14:30	17K	
bu1s4c1dn.ctd	2009-02-25 14:30	16K	
bu1s4c1up.ctd	2009-02-25 14:30	16K	
bu1s4c2dn.ctd	2009-02-25 14:30	81K	
bu1s4c2up.ctd	2009-02-25 14:30	80K	
bu1s4c3dn.ctd	2009-02-25 14:30	17K	

18.3 Zipped File Links

Some data providers such as Pangea provide datasets and metadata as zipped files. While this is very convenient, it is a good idea to still create a collect_datasetname.py file with the zipped file link.

18.4 Webscrapping

Some of the cruise trajectory and metadata was initially collected from R2R (Rolling Deck to Repository). Generally, webscraping is only a last resort.

CHAPTER

NINETEEN

PROCESS

Process is similar to collect in some ways. It contains independent scripts for processing larger datasets into the CMAP database. It serves as a record for the data processing/cleaning steps. The organization roughly mirors **vault/**.

19.1 data flow

As files are processed, retain the raw, unprocessed data in **vault/../{dataset_name}/raw**. Smaller datasets can be exported to **vault/../{dataset_name}/raw** to create a template for submitting to the web validator. Larger datasets such as satellite, model or ARGO float data are too large to be run through the web validator. Metadata should still go through the validator, including dummy data in the data tab with at least 1 row of placeholder data. These datasets can be cleaned and ingested into the database in one process script. Once the data has been ingested, a cleaned version should be exported to **vault/** as a parquet file.

CHAPTER

TWENTY

INGEST

ingest/ is the submodule that contains ingestion and data processing functions, data vault structure logic and DB connection information. Ingesting a dataset using this submodule will be covered in data_ingestion/workflow.

ingest/ is broken into the multiple python scripts to separate ingestion logic. Descriptions and uses of each are described below.

20.1 api_checks.py

This script holds functions that leverage the DB API endpoint. See the data validation section for details.

20.2 common.py

This script holds many commonly used simple functions for data processing/cleaning. It is a good location for generalized use functions.

20.3 credentials.py

This is a simple file hidden with .gitignore. **Make sure this is not pushed to github!** It contains usernames, passwords, ip addresses, ports and connection strings. It is used primarilly by DB.py for database connections. Make sure this file is duplicated across machines.

20.4 cruise.py

This file contains some cruise metadata helper functions and database calls along with a smattering of old cruise trajectory/metadata webscrapping stuff for r2r (rolling deck to repository).

20.5 data_checks.py

This script holds functions used to prepare datasets for ingestion, as well as within the ingestion process. See the data validation section for details.

20.6 data.py

This file contains general cleaning and data processing functions specifically for the data sheet of the template.

20.7 DB.py

DB contains the SQL connection logic, table insert logic, bcp (MSSQL bulk copy program) wrapper functions etc.

20.8 general.py

general.py contains wrapper functions that take arguments through argparse to ingest datasets. It is the main script used in the collection script.

20.9 ingest_test.py

This is a legacy script fragment for a started post-ingestion test suite, with the aim of testing the success of the ingestion. Could be removed, rewritten or expanded upon.

20.10 mapping.py

Contains the functionallity for creating .png and .html interactive maps from input datasets. These are stored in /static and used in the web catalog to give a spatial representation of a dataset. html interactive maps were never included in the catalog, but could be.

20.11 metadata.py

This contains multiple functions to format the dataset_meta_data and vars_meta_data into custom SQL queries.

20.12 region_classification.py

This uses input dataset coordinates along with a geopackage of ocean regions to classify a dataset by spatial region.

20.13 SQL.py

Has functionality to suggest SQL tables and basic indices.

20.14 stats.py

Functions to build summary statistics from datasets. These results are used for data size estimations in the web app.

20.15 transfer.py

A few functions to move and split excel files from /staging to /vault

20.16 vault_structure.py

vault_structure contains the relative paths of vault as well as some directory creation structure.

```
├── assimilation
├── model
├── observation
    ├── in-situ
        ├── cruise
            ├── nrt
            ├── rep
            ├── metadata
            ├── doc
            ├── code
            ├── raw
        ├── drifter
        ├── float
        ├── mixed
        └── station
    └── remote
        └── satellite
├── r2r_cruise
```

CHAPTER

TWENTYONE

CODE CHANGES

- Improve test coverage
- Update all .format() to fstring formatting
- Metadata insert should all be captured in a SQL transaction, so that if a table insert fails, other tables and relations are rolled back.
- Build up cruise ingestion infrastructure
- Build gridded spatio-temporal dataset classifier
- Finish cmapsync and put on cronjob with report emailed

CHAPTER

TWENTYTWO

API REF COMMON.PY

CHAPTER

TWENTYTHREE

API REF CRUISE.PY

CHAPTER

TWENTYFOUR

API REF DATA.PY

CHAPTER

TWENTYFIVE

API REF DB.PY

CHAPTER

TWENTYSIX

API REF GENERAL.PY

CHAPTER

TWENTYSEVEN

API REF MAPPING.PY

CHAPTER

TWENTYEIGHT

API REF METADATA.PY

CHAPTER

TWENTYNINE

API/API_REGION_CLASSIFICATION.PY

CHAPTER

THIRTY

API REF SQL.PY

CHAPTER

THIRTYONE

API REF STATS.PY

CHAPTER

THIRTYTWO

API REF TRANSFER.PY

CHAPTER

THIRTYTHREE

API REF VAULT_STRUCTURE.PY