CalCOFI.io Docs

2024-10-30

Table of contents

# 1. Process

|  |
| --- |
| Figure 1.1: CalCOFI data workflow. |

The original raw **data**, most often in tabular format [e.g., comma-separated value (\*.csv)], gets **ingest**ed into the **database** by R [scripts](https://github.com/CalCOFI/scripts) that use functions and lookup data tables in the R package [**calcofi4r**](https://calcofi.github.io/calcofi4r/reference/index.html) where functions are organized into *Read*, *Analyze* and *Visualize* concepts. The application programming interface (**API**) provides a program-language-agnostic public interface for rendering subsets of data and custom visualizations given a set of documented input parameters for feeding interactive applications (**Apps**) using Shiny (or any other web application framework) and **reports** using Rmarkdown (or any other report templating framework). Finally, R scripts will **publish** metadata (as [Ecological Metadata Language](https://docs.ropensci.org/EML)) and data packages (e.g., in Darwin format) for discovery on a variety of data ***portals*** oriented around slicing the tabular or gridded data ([ERDDAP](https://coastwatch.pfeg.noaa.gov/erddap/information.html)), biogeographic analysis ([OBIS](https://obis.org)), long-term archive ([DataOne](https://www.dataone.org), [NCEI](https://www.ncei.noaa.gov)) or metadata discovery ([InPort](https://www.fisheries.noaa.gov/inport/)). The **database** will be spatially enabled by PostGIS for summarizing any and all data by ***Areas of Interest*** (AoIs), whether pre-defined (e.g., sanctuaries, MPAs, counties, etc.) or arbitrary new areas. ([Figure 1.1](#fig-sw-arch))

* ERDDAP: great for gridded or tabular data, but does not aggregate on the server or clip to a specific area of interest

# 2. Reports

## 2.1 Sanctuaries

* [Channel Islands WebCR](https://noaa-onms.github.io/cinms)  
  web-enabled Condition Report
  + [Forage Fish](https://noaa-onms.github.io/cinms/modals/forage-assemblage.html)  
    example of using calcofi4r functions that pull from the API
* [UCSB Student Capstone](https://shiny.calcofi.io/capstone)

# 3. Applications

* [CalCOFI Oceanography](https://shiny.calcofi.io/oceano)  
  oceanographic summarization by arbitrary area of interest and sampling period
* [UCSB Student Capstone](https://shiny.calcofi.io/capstone)

# 4. API

The raw interface to the Application Programming Interface (API) is available at:

* [api.calcofi.io](https://api.calcofi.io)

Here we will provide more guidance on how to use the API functions with documented input arguments, output results and examples of use.

## 4.1 /variables: get list of variables for timeseries

Get list of variables for use in /timeseries

## 4.2 /species\_groups: get species groups for larvae

Not yet working. Get list of species groups for use with variables larvae\_counts.count in /timeseries

## 4.3 /timeseries: get time series data

## 4.4 /cruises: get list of cruises

Get list of cruises with summary stats as CSV table for time (date\_beg)

## 4.5 /raster: get raster map of variable

Get raster of variable

## 4.6 /cruise\_lines: get station lines from cruises

Get station lines from cruises (with more than one cast)

## 4.7 /cruise\_line\_profile

Get profile at depths for given variable of casts along line of stations

# 5. Database

## 5.1 Database naming conventions

We’re circling the wagons to come up with the best conventions for naming. Here are some ideas:

* [Learn SQL: Naming Conventions](https://www.sqlshack.com/learn-sql-naming-conventions/)
* [Best Practices for Database Naming Conventions - Drygast.NET](https://drygast.net/blog/post/database_naming_conventions)

### 5.1.1 Name tables

* Table names are plural and use all lower case.

### 5.1.2 Name columns

* To name columns, use [**snake-case**](https://cran.r-project.org/web/packages/snakecase/vignettes/introducing-the-snakecase-package.html) (i.e., lower-case with underscores) so as to prevent the need to quote SQL statements. (TIP: Use [janitor::clean\_names()](https://sfirke.github.io/janitor/reference/clean_names.html) to convert a table.)
* Unique **identifiers** are suffixed with:
  + \*\_id for unique integer keys;
  + \*\_key for unique string keys;
  + \*\_seq for auto-incrementing sequence integer keys.
* Suffix with **units** where applicable (e.g., \*\_m for meters, \*\_km for kilometers, degc for degrees Celsius). See [units vignette](https://cran.r-project.org/web/packages/units/vignettes/measurement_units_in_R.html).
* Set geometry column to **geom** (used by [PostGIS](https://postgis.net) spatial extension). If the table has multiple geometry columns, use geom for the default geometry column and geom\_{type} for additional geometry columns (e.g., geom\_point, geom\_line, geom\_polygon).

## 5.2 Use Unicode for text

The [default character encoding for Postgresql](https://www.postgresql.org/docs/current/multibyte.html#MULTIBYTE-SETTING) is unicode (UTF8), which allows for international characters, accents and special characters. Improper encoding can royally mess up basic text.

Use Unicode (utf-8 in Python or UTF8 in Postgresql) encoding for all database text values to support international characters and documentation (i.e., tabs, etc for markdown conversion).

* In **Python**, use [**pandas**](https://pandas.pydata.org/docs/index.html) to read ([read\_csv()](https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html)) and write ([to\_csv()](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_csv.html)) with UTF-8 encoding (i.e., encoding='utf-8').:
* import pandas as pd  
  from sqlalchemy import create\_engine  
  engine = create\_engine('postgresql://user:password@localhost:5432/dbname')  
    
  # read from a csv file  
  df = pd.read\_csv('file.csv', encoding='utf-8')  
    
  # write to PostgreSQL  
  df.to\_sql('table\_name', engine, if\_exists='replace', index=False, method='multi', chunksize=1000, encoding='utf-8')  
    
  # read from PostgreSQL  
  df = pd.read\_sql('SELECT \* FROM table\_name', engine, encoding='utf-8')  
    
  # write to a csv file with UTF-8 encoding  
  df.to\_csv('file.csv', index=False, encoding='utf-8')
* In **R**, use [**readr**](https://readr.tidyverse.org/index.html) to read ([read\_csv()](https://readr.tidyverse.org/reference/read_delim.html)) and write ([write\_excel\_csv()](https://readr.tidyverse.org/reference/write_delim.html)) to force UTF-8 encoding.
* library(readr)  
  library(DBI)  
  library(RPostgres)  
    
  # connect to PostgreSQL  
  con <- dbConnect(RPostgres::Postgres(), dbname = "dbname", host = "localhost", port = 5432, user = "user", password = "password")  
    
  # read from a csv file  
  df <- read\_csv('file.csv', locale = locale(encoding = 'UTF-8')) # explicit  
  df <- read\_csv('file.csv') # implicit  
    
  # write to PostgreSQL  
  dbWriteTable(con, 'table\_name', df, overwrite = TRUE)  
    
  # read from PostgreSQL  
  df <- dbReadTable(con, 'table\_name')  
    
  # write to a csv file with UTF-8 encoding  
  write\_excel\_csv(df, 'file.csv', locale = locale(encoding = 'UTF-8')) # explicit  
  write\_excel\_csv(df, 'file.csv') # implicit

## 5.3 Describe tables and columns

* Use the COMMENT clause to add descriptions to tables and columns, either through the GUI [pgadmin.calcofi.io](https://pgadmin.calcofi.io/) (by right-clicking on the table or column and selecting Properties) or with SQL. For example:
* COMMENT ON TABLE public.aoi\_fed\_sanctuaries IS 'areas of interest (`aoi`) polygons for federal \*\*National Marine Sanctuaries\*\*; loaded by \_workflow\_ [load\_sanctuaries](https://calcofi.io/workflows/load\_sanctuaries.html)';
* Note the use of [**markdown**](https://www.markdownguide.org/cheat-sheet/) for including links and formatting (e.g., bold, code, italics), such that the above SQL will render like so:
* areas of interest (aoi) polygons for federal **National Marine Sanctuaries**; loaded by *workflow* [load\_sanctuaries](https://calcofi.io/workflows/load_sanctuaries.html)
* It is especially helpful to link to any ***workflows*** that are responsible for the ingesting or updating of the input data.
* These descriptions can be viewed in the CalCOFI **API** [api.calcofi.io](https://api.calcofi.io) as CSV tables (see code in [calcofi/api: plumber.R](https://github.com/CalCOFI/api/blob/8ad9d9ad62fd526d4b8da23357759f1ad196cb88/plumber.R#L916-L990)):
  + [api.calcofi.io/db\_tables](https://api.calcofi.io/db_tables)  
    fields:  
    - schema: (only “public” so far)
    - table\_type: “table”, “view”, or “materialized view” (none yet)
    - table: name of table
    - table\_description: description of table (possibly in markdown)
  + [api.calcofi.io/db\_columns](https://api.calcofi.io/db_columns)  
    fields:  
    - schema: (only “public” so far)
    - table\_type: “table”, “view”, or “materialized view” (none yet)
    - table: name of table
    - column: name of column
    - column\_type: data type of column
    - column\_description: description of column (possibly in markdown)
* Fetch and display these descriptions into an interactive table with [calcofi4r::**cc\_db\_catalog()**](https://calcofi.io/calcofi4r/reference/cc_db_catalog.html).

## 5.4 Relationships between tables

* See [calcofi/workflows: **clean\_db**](https://calcofi.io/workflows/clean_db.html)
* TODO: add calcofi/apps: db to show latest tables, columns and relationsips

## 5.5 Spatial Tips

* Use [ST\_Subdivide()](https://postgis.net/docs/ST_Subdivide.html) when running spatial joins on large polygons.

# 6. References

## 6.1 R packages

* API: plumber ([Schloerke and Allen 2024](#ref-R-plumber))
* docs: Quarto ([Allaire and Dervieux 2024](#ref-R-quarto))
* apps: Shiny ([Chang et al. 2024](#ref-R-shiny))

Allaire, JJ, and Christophe Dervieux. 2024. *Quarto: R Interface to Quarto Markdown Publishing System*. <https://github.com/quarto-dev/quarto-r>.

Chang, Winston, Joe Cheng, JJ Allaire, Carson Sievert, Barret Schloerke, Yihui Xie, Jeff Allen, Jonathan McPherson, Alan Dipert, and Barbara Borges. 2024. *Shiny: Web Application Framework for r*. <https://shiny.posit.co/>.

Schloerke, Barret, and Jeff Allen. 2024. *Plumber: An API Generator for r*. <https://www.rplumber.io>.