CalCOFI.io Docs

2025-04-23

Table of contents

1 Process

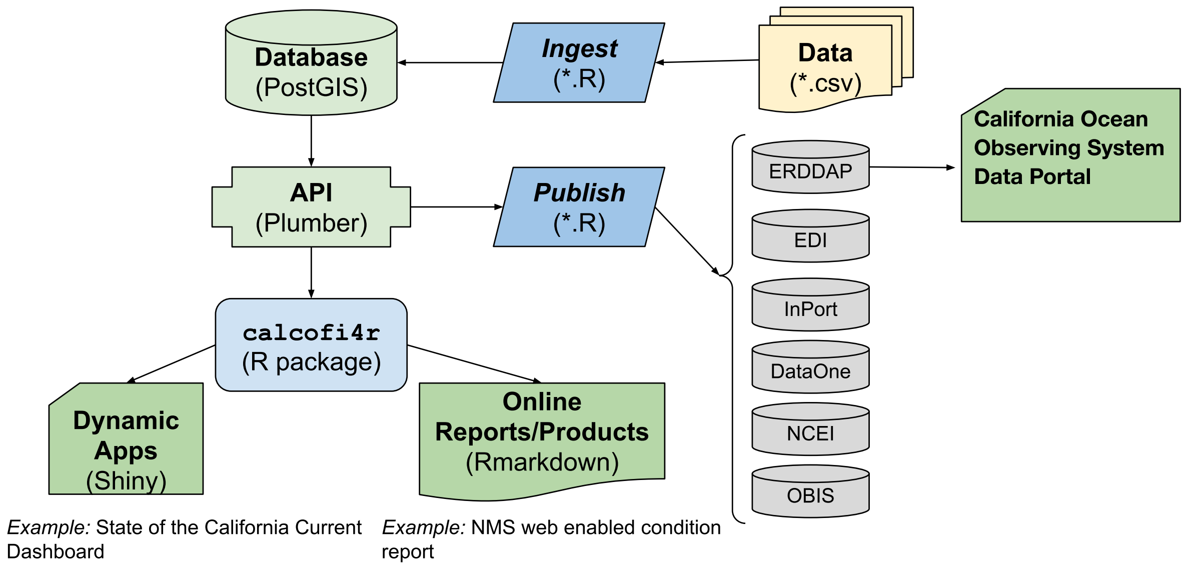


Figure 1.1: CalCOFI data workflow.

The original raw **data**, most often in tabular format [e.g., comma-separated value (*.csv)], gets **ingest**ed into the **database** by R scripts that use functions and lookup data tables in the R package **`calcofi4r`** 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) 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), biogeographic analysis (OBIS), long-term archive (DataOne, NCEI) or metadata discovery (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)

- 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
  web-enabled Condition Report
  - Forage Fish
    example of using calcofi4r functions that pull from the API
- UCSB Student Capstone

3 Applications

- CalCOFI Oceanography
  oceanographic summarization by arbitrary area of interest and sampling period
- UCSB Student Capstone

4 API

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

- 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

There are only two hard things in Computer Science: cache invalidation and naming things. – Phil Karlton (Netscape architect)

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

- Learn SQL: Naming Conventions
- Best Practices for Database Naming Conventions - Drygast.NET

5.1.1 Name tables

- Table names are singular and use all lower case.

5.1.2 Name columns

- To name columns, use **snake-case** (i.e., lower-case with underscores) so as to prevent the need to quote SQL statements. (TIP: Use `janitor::clean_names()` to convert a table.)
- Unique **identifiers** are suffixed with:
  - `*_id` for unique integer keys;
  - `*_uuid` for universally unique identifiers as defined by RFC 4122 and stored in Postgres as UUID Type.
  - `*_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.
- Set geometry column to **`geom`** (used by PostGIS 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 is unicode (`UTF8`), which allows for international characters, accents and special characters. Improper encoding can royally mess up basic text.

Logging into the server, we can see this with the following command:

```
docker exec -it postgis psql -l
```

```
                                    List of databases
        Name         | Owner | Encoding |  Collate   |   Ctype    | Access privileges
---------------------+-------+----------+------------+------------+--------------------
 gis                 | admin | UTF8     | en_US.utf8 | en_US.utf8 | =Tc/admin         +
                     |       |          |            |            | admin=CTc/admin   +
                     |       |          |            |            | ro_user=c/admin
 lter_core_metabase  | admin | UTF8     | en_US.utf8 | en_US.utf8 | =Tc/admin         +
                     |       |          |            |            | admin=CTc/admin   +
                     |       |          |            |            | rw_user=c/admin
 postgres            | admin | UTF8     | en_US.utf8 | en_US.utf8 |
 template0           | admin | UTF8     | en_US.utf8 | en_US.utf8 | =c/admin          +
                     |       |          |            |            | admin=CTc/admin
 template1           | admin | UTF8     | en_US.utf8 | en_US.utf8 | =c/admin          +
                     |       |          |            |            | admin=CTc/admin
 template_postgis    | admin | UTF8     | en_US.utf8 | en_US.utf8 |
(6 rows)
```

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`** to read (`read_csv()`) and write (`to_csv()`) 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', chunks
  ```

```
# 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` to read (`read_csv()`) and write (`write_excel_csv()`) to force UTF-8 encoding.

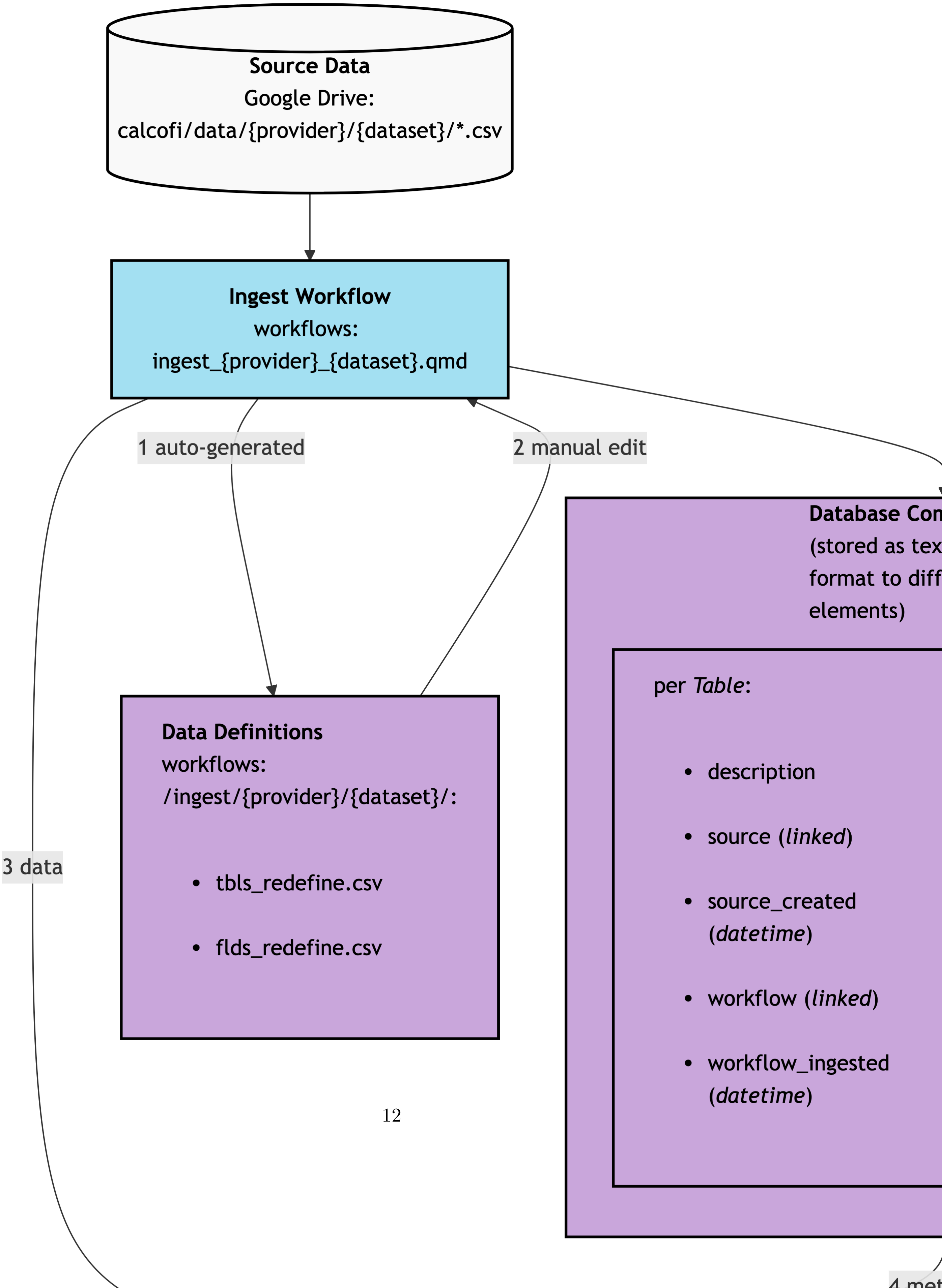
```r
library(readr)
library(DBI)
library(RPostgres)

# connect to PostgreSQL
con <- dbConnect(RPostgres::Postgres(), dbname = "dbname", host = "localhost", port = 54

# 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 Ingest datasets with documentation

Use Quarto documents with chunks of R code in the workflows Github repository to ingest datasets into the database. For example, see the ingest_noaa-calcofi-db workflow.

5.3.1 Using calcofi4db package

The calcofi4db package provides functions to streamline dataset ingestion, metadata generation, and change detection. The standard workflow is:



Source Data
Google Drive:
calcofi/data/{provider}/{dataset}/*.csv
Ingest Workflow
workflows:
ingest_{provider}_{dataset}.qmd
1 auto-generated
2 manual edit
Database Con
(stored as tex
format to diff
elements)
per Table:
description
source (linked)
source_created (datetime)
workflow (linked)
workflow_ingested (datetime)
Data Definitions
workflows:
/ingest/{provider}/{dataset}/:
tbls_redefine.csv
flds_redefine.csv
3 data
4 met

1. **Load data files**: `load_csv_files()` reads CSV files from a directory and prepares them for ingestion
2. **Transform data**: `transform_data()` applies transformations according to redefinition files
3. **Detect changes**: `detect_csv_changes()` compares data with existing database tables
4. **Ingest data**: `ingest_csv_to_db()` writes data to the database with proper metadata

For convenience, the high-level `ingest_dataset()` function combines these steps:

```r
library(calcofi4db)
library(DBI)
library(RPostgres)

# Connect to database
con <- dbConnect(
  Postgres(),
  dbname = "gis",
  host = "localhost",
  port = 5432,
  user = "admin",
  password = "postgres"
)

# Ingest a dataset
result <- ingest_dataset(
  con = con,
  provider = "swfsc.noaa.gov",
  dataset = "calcofi-db",
  dir_data = "/path/to/data",
  schema = "public",
  dir_googledata = "https://drive.google.com/drive/folders/your-folder-id",
  email = "your.email@example.com"
)

# Examine changes and results
result$changes
result$stats
```

5.3.2 Workflow details

Google Drive *.csv files get ingested with a **workflow** per **dataset** (in Github repository calcofi/workflows as a Quarto document). Data definition CSV files (`tbls_redefine.csv` , `flds_redefine.csv`) are auto-generated (if missing) and manually updated to rename and describe tables and fields. After injecting the data for each of the tables, extra metadata is added to the `COMMENT`s of each table as JSON elements (links in markdown), including at the ***table*** level:

- **description**: general description describing contents and how each row is unique
- **source**: CSV (linked to Google Drive source as markdown)
- **source_created**: datetime stamp of when source was created on GoogleDrive
- **workflow**: html (rendered Quarto document on Github)
- **workflow_ingested**: datetime of ingestion

And at the ***field*** level:

- **description**: general description of the field
- **units**: using the International System of Units (SI) as much as possible

These comments are then exposed by the API db_tables endpoint, which can be consumed and rendered into a tabular searchable catalog with calcofi4r::cc_db_catalog.

5.3.3 Change detection strategy

The `calcofi4db` package implements a comprehensive change detection strategy:

1. **Table changes**:
    - New tables are identified for initial creation
    - Existing tables are identified for potential updates
2. **Field changes**:
    - Added fields: New columns in CSV not present in the database
    - Removed fields: Columns in database not present in the CSV
    - Type changes: Fields with different data types between CSV and database
3. **Data changes**:
    - Row counts are compared between source and destination
    - Data comparison is handled with checksum verification

If changes are detected, they are displayed to the user who can decide whether to: - Create new tables - Modify existing table schemas - Update data with appropriate strategies (append, replace, merge)

Additional workflows will publish the data to the various Portals (ERDDAP, EDI, OBIS, NCEI) using ecological metadata language (EML) and the EML R package, pulling directly from the structured metadata in the database (on table and field definitions).

5.3.4 OR Describe tables and columns directly

- Use the `COMMENT` clause to add descriptions to tables and columns, either through the GUI 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 f
  ```

- Note the use of **markdown** 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

- It is especially helpful to link to any ***workflows*** that are responsible for the ingesting or updating of the input data.

5.3.5 Display tables and columns with metadata

- These descriptions can be viewed in the CalCOFI **API** api.calcofi.io as CSV tables (see code in calcofi/api: `plumber.R`):
  - 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`
    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()`.

5.4 Relationships between tables

- See calcofi/workflows: **clean_db**

- `TODO`: add calcofi/apps: db to show latest tables, columns and relationsips

5.5 Spatial Tips

- Use `ST_Subdivide()` when running spatial joins on large polygons.

6 Portals

6.1 Overview

CalCOFI data is available through various portals, each serving different purposes and user needs. This document outlines the main access points and their characteristics.

6.2 Data Flow

While it would be ideal for CalCOFI data to be available through a single portal, each portal has its strengths and limitations. The following diagram illustrates one possible realization of data flow between CalCOFI data and the portals: from raw data to the integrated database to portals and meta-portals.

In practice, CalCOFI is a partnership with various contributing members, so the authoritative dataset might flow differently, such as from EDI to the database to the other portals. The other portals, such as OBIS or ERDDAP, serve different audiences or purposes. The meta-portals like ODIS and Data.gov then index these portals to provide broader discovery of CalCOFI datasets.

6.3 Portals

While some portals serve as data repositories, others provide advanced data access and visualization tools. The following sections describe the main portals where CalCOFI data is available and their key features.

6.3.1 EDI

Environmental Data Initiative

- Complete dataset archives using DataOne software and EML metadata
- DOIs issued for all datasets ensuring citability
- Full archive allowing for any data file types
- Basic spatial and temporal filtering through web interface

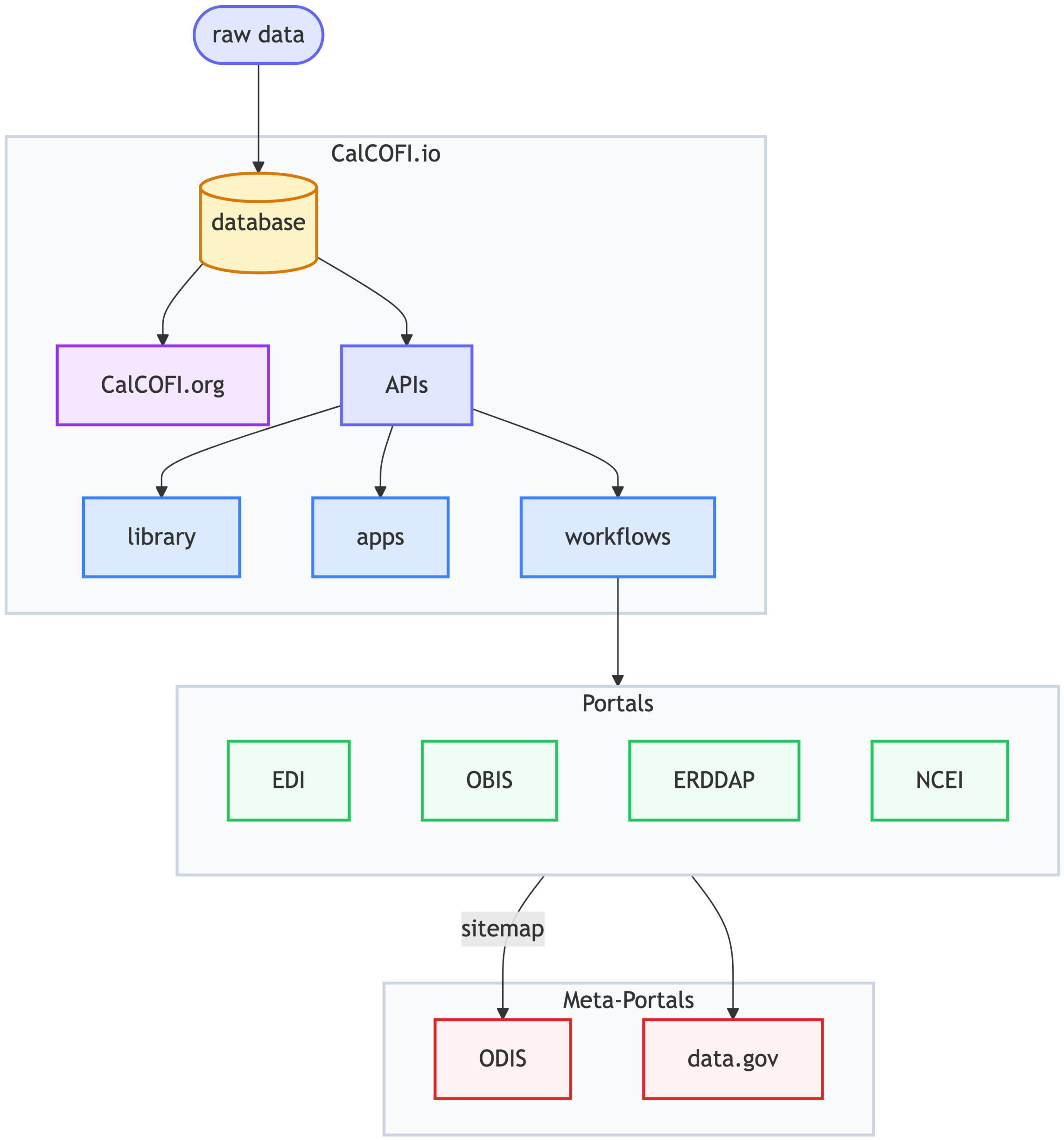


Figure 6.1: Flow of data from raw to database to portals and meta-portals.

Table 6.1: Portal Capabilities.

	Full Archive	Versioning	DOI Issued	Query by xyt	Query by taxa	Multiple formats
EDI	🟩	🟩	🟩	🟨	🟨	🟥
NCEI	🟩	🟩	🟩	🟥	🟥	🟥
OBIS	🟨	🟨	🟨	🟩	🟩	🟨
ERDDAP	🟨	🟥	🟥	🟩	🟨	🟩

Capability Legend: = full, = partial, = none

- Download in original formats with metadata
- Access through DataOne API
- Links:
  - EDIrepository.org
  - CalCOFI datasets: EDI query "CalCOFI"

6.3.2 NCEI

National Centers for Environmental Information

- Long-term archival of oceanographic data
- DOIs issued for dataset submissions
- Standardized metadata using ISO 19115-2
- Basic search interface with geographic and temporal filtering
- Data preserved in original submission formats
- Access through NCEI API services
- Links:
  - NCEI Ocean Archive
  - CalCOFI datasets: NCEI search "CalCOFI"

6.3.3 OBIS

Ocean Biodiversity Information System

- Specialized in marine biodiversity data
- Standardized using DarwinCore fields
- Extended measurements supported via extendedMeasurementOrFact
- Powerful filtering by space, time, and taxonomic parameters
- Multiple download formats (CSV, JSON, Darwin Core Archive)
- Full REST API access

- Links:
  - OBIS.org
  - CalCOFI datasets: obis.org/dataset + "calcofi" Keyword

6.3.4 ERDDAP

Environmental Research Division Data Access Program

- Tabular and gridded data server
- Advanced subsetting by space, time, and parameters
- Multiple output formats (CSV, JSON, NetCDF, etc.)
- RESTful API with direct data access
- Built-in data visualization tools
- No persistent identifiers but stable URLs
- Links:
  - ERDDAP
  - CalCOFI datasets:
    - ERDDAP, OceanView - CalCOFI seabirds
    - ERDDAP, CoastWatch - CalCOFI oceanographic

6.4 Metadata

The Ecological Metadata Language (EML) (and using R package EML in workflows) serves as a key standard for describing ecological and environmental data. For CalCOFI, EML metadata files are generated alongside data files, providing structured documentation that enables interoperability across different data portals. This metadata-driven approach allows automated ingestion into various data systems while maintaining data integrity and provenance.

The EML specification provides detailed structure for describing datasets, including:

- Dataset identification and citation
- Geographic and temporal coverage
- Variable definitions and units
- Methods and protocols
- Quality control procedures
- Access and usage rights

This standardized metadata enables automated data transformation and ingestion into various portal systems while preserving the original data context and quality information.

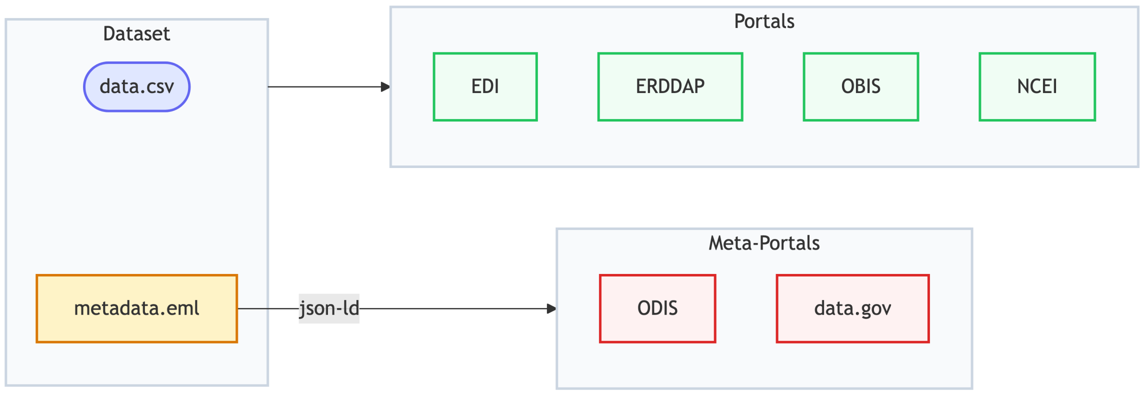


Figure 6.2: Metadata in the form of ecological metadata language (EML) is used to describe the dataset in a consistent manner that can be ingested by the portals.

6.5 Meta-Portals

6.5.1 Google Dataset Search

The JSON-LD metadata in the Portal dataset web pages get indexed by Google Dataset Search through schema.org metadata. This ensures that CalCOFI data is discoverable through Google search and other search engines.

6.5.2 ODIS

Ocean Data Information System

ODIS uses the same technology as Google Dataset Search (schema.org, JSON-LD), but focuses on ocean data. CalCOFI curates a sitemap of authoritative datasets to server to ODIS.org

This federated approach ensures that CalCOFI data remains:

- Discoverable through multiple channels
- Properly cited and attributed
- Integrated with global ocean data systems

6.6 CalCOFI.io Tools

CalCOFI is also developing an integrated database and tools that enable efficient data access and analysis:

6.6.1 APIs

- RESTful endpoints for programmatic access
- Filtering by space, time, and taxonomic parameters
- Relationship queries across tables
- Links:
  - api.calcofi.io
  - tile.calcofi.io

6.6.2 Library

- Direct data access from R
- Built-in analysis functions
- Integration with tidyverse ecosystem
- Link:
  - calcofi.io/calcofi4r

6.6.3 Apps

- Interactive data exploration with Shiny applications
- User-friendly interfaces
- Subset and download data
- Link:
  - calcofi.io, App button

7 References

7.1 R packages

- API: plumber (Schloerke and Allen 2024)
- docs: Quarto (Allaire and Dervieux 2024)
- apps: Shiny (Chang et al. 2024)

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.