MERMAID API

MERMAID

Apr 01, 2022

GETTING STARTED

The MERMAID API is the central interface through which all MERMAID data is read and written, and is the core of the MERMAID project, which seeks to accelerate coral reef conservation by making common coral reef data collection and analysis fast, less error-prone, secure, and flexible.

This documentation refers to its production instantiation at https://api.datamermaid.org/v1/, backed by an AWS-based stack including an RDS PostgreSQL database and using Django and Django REST Framework, and is actively used by

- MERMAID collection app [repository]
- MERMAID public dashboard [repository]
- mermaidr analysis package

Our code is free (as in speech) and open source.

Note: This documentation is for folks who know how to use APIs. If you are not a developer and just want to access your MERMAID data for analysis, go to mermaidr and read the excellent documentation there.

CHAPTER

ONE

USING THE API

Automatic swagger API documentation is not yet available (*volunteers welcome!*). Instead, an API client file is provided for easy experimentation:
Insomnia collection

The API client Insomnia is required to use this file. You can also export to other clients such as Postman, but Insomnia is recommended for its ability to store things like authentication parameters as variables.
Details on how to import (and re-export to) other formats

Users may also inspect the code and console output from the above apps to see how the API is used in practice.

Note: Most resources require a JWT token with every request; for more on this, see *Authentication*.

The included Insomnia collection makes use of `Environments`, which allow for the provision of arbitrary variables that can be used anywhere in the app. The `prod` environment, for example, defines `api_url` as `https://api.datamermaid.org/v1`. In this way users can easily switch between testing in different environments (including `local`).

Before you can start using the Insomnia collection, you need to:

1. Have or create a MERMAID user account appropriate to the targeted environment, easily obtainable by going to https://collect.datamermaid.org/ (production).
2. Change the `project` environment variable to the id of a project in MERMAID to which you belong, by copying/pasting from the Collect url, like this:

1.1 Formats

All responses to GET requests for non-aggregated-view urls are returned as standard JSON, with `content-type = application/json`. In addition, *Aggregated views* offer two additional formats, accessible by appending the relevant string to resources:

- `.../csv/`: returns comma-separated 2D matrix (some fields are JSON)
- `.../geojson/`: returns GeoJSON

`GET` requests will return lists of objects. Appending an object id to most resources will return a single object.

1.2 Pagination

All JSON and GeoJSON responses to GET requests are paginated with a default `page_size` of 100 that may be increased to a maximum of 5000 by appending a `limit` query parameter. Results contain `count`, `next`, and `previous` in addition to a list of actual `results`, e.g.::

```
{
  "count": 3275,
  "next": "https://api.datamermaid.org/v1/fishspecies/?page=2",
  "previous": null,
  "results": [...]
}
```

1.3 Filters

The included Insomnia collection contains examples of filters available for each resource, accessible from the *Query* tab of each request. In general, the basic "core" resources such as `fishspecies` support filtering by fields available from that resource, e.g.
`https://api.datamermaid.org/v1/fishspecies/?genus=f5263c54-ea12-4a18-8200-d52967376d1a`

These resources also support ordering by field (or related field) names, e.g.
`https://api.datamermaid.org/v1/fishspecies/?ordering=genus__name`

Many resources also allow limited searching via filters, e.g.:
`https://api.datamermaid.org/v1/fishspecies/?search=Abalistes`

Aggregated views offer searching by multiple comma-separated values, e.g.
`https://api.datamermaid.org/v1/projects/8c213ce8-7973-47a5-9359-3a0ef12ed201/beltfishes/sampleunits/?site_name=KOE01,Lhok%20Weng`

CHAPTER

TWO

AUTHENTICATION

MERMAID uses OAuth2 authentication for securing and accessing the MERMAID API's secure endpoints. The following steps are required before making requests to the API:

2.1 1. Create a MERMAID Account

Before you can make a request for a JSON Web Token (JWT) or accessing the API, you must first create a MERMAID user account. If you already have an account jump to section 2, if not, an account can be created at https://collect.datamermaid.org/.

2.2 2. Requesting Tokens

OAuth2 Implicit grant type is used to fetch a valid token that can be used to securely access MERMAID API. The folowing details will be needed to setup an implicit authorization flow:

- Authorization URL
- Redirect URL
- Client ID
- Audience

These details can be requested from the MERMAID team at https://datamermaid.org/contact/.

2.3 3. Calling API

When making requests to the API the token can be included in:

- the request header

```
curl --request GET \
    --url https://api.datamermaid.org/projects/ \
    --header 'Authorization: Bearer <VALID TOKEN HERE>'
```

- the url query string

```
https://api.datamermaid.org/projects/?access_token=<VALID TOKEN HERE>
```

CHAPTER

THREE

NON-PROJECT RESOURCES

All resources not associated with particular projects are on the API root (in production, `https://dev-api.datamermaid.org/v1/`).

3.1 General

Except where noted otherwise:

Authentication: not required
Methods: `GET`, `HEAD`, `OPTIONS`

3.1.1 /health/

Returns `ok` unconditionally.

3.1.2 /version/

Returns current versions of registered apps – currently `collect` and `api`.

Authentication: required

3.1.3 /me/

Profile information for current user.

Authentication: required
Permissions: Only current user may access.
Methods: `GET`, `PUT`
Additional routes:

- `change_password/` (`POST`)

3.1.4 /profiles/

List of user profiles. For privacy, only the `id`s of users are returned, without personally identifiable information.

3.1.5 /projecttags/

List of project tags ("organizations"). Read-only; new tags are created with *proposed* status via editing a project.

3.1.6 /sites/

List of project sites. Includes a `validations` item listing the results of the validation that ran the last time the site was validated.

Additional filters:

- `exclude_projects`: comma-separated list of project ids
- `unique`: returns list of sites not belonging to the provided project id that are a unique combination of `name`, `country_id`, `reef_type_id`, `reef_zone_id`, `exposure_id`, `location`

Note: A site belongs to only one project. Two sites might have exactly the same attributes (other than `id`) but are considered different sites.

3.1.7 /managements/

List of project management regimes. Includes a `validations` item listing the results of the validation that ran the last time the management regime was validated.

Additional filters:

- `exclude_projects`: comma-separated list of project ids
- `unique`: returns list of sites not belonging to the provided project id that are a unique combination of `name`, `country_id`, `reef_type_id`, `reef_zone_id`, `exposure_id`, `location`
- `est_year_min`/`est_year_max`: earliest/latest year established

Note: A management regime belongs to only one project. Two management regimes might have exactly the same attributes (other than `id`) but are considered different management regimes.

3.2 Choices and attributes

Choices and attributes provide most of the "lookup" entities that MERMAID transects and observations require. "Attributes" are "things that can be observed" – coral and other taxa as well as nonorganic benthic substrates for benthic transects and bleaching surveys, fish species/genera/families as well as arbitrary fish groupings for fish belt transects, and so on.

All attributes have a filterable `regions` property that detail which of the 12 MEOW regions the attribute belongs to.

All attributes also allow `POST`, `PUT`, and `DELETE` methods, where an authenticated user may use `POST` to suggest a new attribute (marked on creation as `status=PROPOSED`), and edit/delete an existing attribute that has `status=PROPOSED`.

3.2.1 /choices/

Convenience resource that returns a list of objects, each one of which has a `name` item (e.g., `countries`) and a `data` item that is a list of available choice objects.

Additional routes:

- `updates/` (`GET`): returns object of following form where items have changed since the passed `timestamp` parameter:

```
{
  "added": [...],
  "modified": [...],
  "deleted": [...]
}
```

3.2.2 /fishsizes/

Separate choice resource used only for looking up the actual size to record for a fish, given a particular fish size bin used for the survey.

3.2.3 /benthicattributes/

List of MERMAID benthic attributes. Includes nonorganic substrates like "rubble" as well as coral and other taxa.

3.2.4 /fishfamilies/

List of MERMAID fish families. Biomass constants are the calculated means of all species belonging to each family.

3.2.5 /fishgenera/

List of MERMAID fish genera. Biomass constants are the calculated means of all species belonging to each genus.

3.2.6 /fishspecies/

List of MERMAID fish species. Includes biomass constants and maximum observed length as well as useful analytical properties such as vulnerability score, trophic level, trophic group, and functional group.

3.2.7 /fishgroupings/

Fish groupings are arbitrary (but useful) groupings of fish species, genera, and families that are treated as a single taxon for purposes of observation and analysis (typically some form of "other"). As with fish genera and families, biomass constants and regions are calculated from member taxa; additionally, a `fish_attributes` property is returned listing each member species, genus, and family.

3.3 /projects/

The projects resource at the root of the API, without query parameters, returns a list of projects of which the user is a member. The `showall` query parameter may be used to return projects unfiltered by the user's membership. `showall` is important when the user is unauthenticated.

Authentication: required for `PUT` and `POST` requests.
Permissions: Read-only when unauthenticated. To update, the user must be an `admin` for the project, unless they are using the `find_and_replace_sites/` or `find_and_replace_managements/` routes, in which case the user may be a project member of any non-readonly type.
Methods: `GET`, `PUT`, `POST`
Additional routes:

- `create_project/` (`POST`): Create new project from request body, including all related project profiles, sites, and management regimes.
- `updates/` (`GET`): returns object of following form where items have changed since the passed `timestamp` parameter:

```
{
  "added": [...],
  "modified": [...],
  "deleted": [...]
}
```

- `find_and_replace_sites/` (`PUT`): Replace the site specified by the `find` query parameter associated with all submitted and unsubmitted sample units with the site specified by the `replace` query parameter, then delete the `find` site.
- `find_and_replace_managements/` (`PUT`): Replace the management regime specified by the `find` query parameter associated with all submitted and unsubmitted sample units with the management regime specified by the `replace` query parameter, then delete the `find` management regime.
- `transfer_sample_units/` (`PUT`): Associate every sample unit in the project with the profile specified by the `from_profile` query parameter with the profile specified by the `to_profile` query parameter.

CHAPTER

FOUR

PROJECT RESOURCES

Project-related resources in MERMAID all begin, relative to the API root, with `/projects/<project_id>/`, where `<project_id>` is the UUID of a project. See *Using the API* for how to determine a `project_id` manually, or use the API to retrieve a list of project ids to which a user has access using the */projects/* resource.

4.1 Data access

All data access in MERMAID is based on projects. No top-down organizational hierarchy or ACL logic is used; rather, any user may create a new project and add any other MERMAID user to it. The concept of "organization" exists; a project may be associated with any number of organizations, as tags, useful for filtering but not access control.

Authenticated access to project data depends on the association of a user profile with a project, in different roles. Generally, the permissions associated with these roles govern access to the Project resources specified on this page.

Unauthenticated access to project data depends on the data sharing policies chosen per survey method for a project. Generally, these policies govern access to *aggregated views*, not the Project resources specified here, which all require authentication and project membership.

All resources specified on this page support: `GET`, `PUT`, `PATCH`, `POST`, `DELETE`, `HEAD`, `OPTIONS` except as noted for sample unit methods.

4.1.1 Roles

Authenticated access to project data depends on the association of a user profile with a project, in one of three roles:

- **admin** [90]: User has all permissions for project, including removing other (potentially admin) users. A user is an admin on any project they create.
- **collector** [50]: User may create/update/delete CollectRecords (unsubmitted sample units), sites, and management regimes, and may create ("suggest") benthic attributes and fish species. All other permissions are read-only.
- **read-only** [10]: User may read all data for project, but may not create, update, or delete anything.

4.2 Project entity resources

4.2.1 /projects/<project_id>/sites/

All sites for a project.

4.2.2 /projects/<project_id>/managements/

All management regimes for a project.

4.2.3 /projects/<project_id>/project_profiles/

All user profiles for a project. Note a given CollectRecord (unsubmitted sample unit) is associated with a user profile, and in the MERMAID frontend is only available to that user. Note also that though a `project_profile` has an `id`, `project_profile.profile` is the id of the user profile associated with the project.

4.2.4 /projects/<project_id>/observers/

An `observer` is a relationship between a user `profile` and a sample unit method; a given sample unit method may have multiple observers, and the profile associated with a CollectRecord may or may not be one of the profiles of those observers. Note also that though an `observer` has an `id`, `observer.profile` is the id of the user profile associated with the observer.

4.2.5 /projects/<project_id>/collectrecords/

See *Collect Records* for more detail.

A `CollectRecord` is a nested JSON representation of all the objects and values that together represent an unsubmitted sample unit for a project. A `GET` request to this resource returns the CollectRecords created by the user, unless the `showall` query parameter is used to return projects unfiltered by the user's membership.

Permissions: Regular project-based permissions apply, but only the user who created a CollectRecord may use the `validate` and `submit` routes.
Additional routes:

- `validate/` (`POST`): Runs all relevant validations for a CollectRecord, stores those validations in the CollectRecord itself, and returns them in the response.
- `submit/` (`POST`): Submits CollectRecord, i.e. attempts to store all the component parts of the unsubmitted sample unit in the correct places. Runs validations as part of submission.

4.3 Observations

Observation resources are the lowest level of MERMAID data, representing individual observations in sample unit methods, which belong to sample events (a set of sample unit methods at a particular site on a particular date).

4.3.1 /projects/<project_id>/obstransectbeltfishs/

Belt fish transect observations. Filters:

- `beltfish`
- `beltfish__transect`
- `beltfish__transect__sample_event`
- `fish_attribute`
- `size_min/size_max`
- `count_min/count_max`

4.3.2 /projects/<project_id>/obsbenthiclits/

Benthic LIT observations. Filters:

- `benthiclit`
- `benthicpit__transect`
- `benthiclit__transect__sample_event`
- `attribute`
- `growth_form`
- `length_min/length_max`

4.3.3 /projects/<project_id>/obsbenthicpits/

Benthic PIT observations. Filters:

- `benthicpit`
- `benthicpit__transect`
- `benthicpit__transect__sample_event`
- `attribute`
- `growth_form`

4.3.4 /projects/<project_id>/obshabitatcomplexities/

Habitat complexity observations. Filters:

- `habitatcomplexity`
- `habitatcomplexity__transect`
- `habitatcomplexity__transect__sample_event`
- `score` (lookups in `habitatcomplexityscores` object from */choices/* resource)

4.3.5 /projects/<project_id>/obscoloniesbleached/

Observations of number of coral colones bleached for a quadrat collection. Simple equality filters are available for every field.

4.3.6 /projects/<project_id>/obsquadratbenthicpercent/

Observations of percent cover for hard coral, soft coral, and algae for each quadrat in a quadrat collection. Simple equality filters are available for every field.

4.4 Sample units

In MERMAID, what are often referred to as "sample units" or "transects" are in fact "sample unit method" instances – applications of a survey methodology to a physical transect or quadrat collection. The latter are actual "sample units". Thus, a single benthic transect might be associated with a benthic PIT, benthic LIT, or habitat complexity transect method. These endpoints are rarely employed by themselves.

The only useful filters are likely to be `len_surveyed_min/len_surveyed_max` for `fishbelttransects` and `benthictransects`.

4.4.1 /projects/<project_id>/fishbelttransects/

4.4.2 /projects/<project_id>/benthictransects/

4.4.3 /projects/<project_id>/quadratcollections/

4.5 Sample unit methods

Sample unit methods are not directly creatable; they are created when a request is made to the *Collect Records* `submit/` route, after having passed validation. They have no filters. The body of a `PUT` request for updating a sample unit method is the same as that of a CollectRecord.

Methods: `GET`, `PUT`, `HEAD`, `DELETE`

4.5.1 /projects/<project_id>/beltfishtransectmethods/

4.5.2 /projects/<project_id>/benthiclittransectmethods/

4.5.3 /projects/<project_id>/benthicpittransectmethods/

4.5.4 /projects/<project_id>/habitatcomplexitytransectmethods/

4.5.5 /projects/<project_id>/bleachingquadratcollectionmethods/

4.5.6 /projects/<project_id>/sampleunitmethods/

4.6 Sample events

A sample event in MERMAID is a unique combination of site, management regime (both of which are specific to a project), and sample date. It represents all observations from all sample units (of whatever type) collected at a place on a date.

4.6.1 /projects/<project_id>/sampleevents/

Filters:

- `sample_date_before/sample_date_after`

CHAPTER

FIVE

COLLECT RECORDS

Data collected during one of the sample unit method surveys (i.e. Fish Belt, Benthic PIT, etc) is stored in a Collect Record. Collect records are different then other types of records in MERMAID and go through a staged process to help ensure the finalized record is valid, clean data. The stages in this process are `Save`, `Validate`, and `Submit`.

5.1 Save

A collect record can be saved at any point even if it is partially populated or in an invalid state. The record can be thought of as being in a "draft" state.

5.2 Validate

Saved records can be validated by calling the collect record's validation endpoint, which responds with the overall validation status (`ok`, `warning` or `error`) and an updated copy of the collect record that was validated that includes the detailed individual validation results.

Example validate response:

```
{
    "f5c8f06a-8ba0-4385-8e9e-ad154c059d94": {
        "status": "error",
        "record": {
            "id": "ffffffff-8ba0-4385-8e9e-ad154c059d94",
            ... trimmed ...
            "validations": {
                "status": "error",
                "results": {
                    "site": {
                        "wrapped": {
                            "status": "error",
                            "message": "Site record not available for similarity␣
↪validation"
                        },
                        "validate_exists": {
                            "status": "error",
                            "message": "Site: Record doesn't exist"
                        }
                    },
                    "depth": {
```

(continues on next page)

(continued from previous page)

```
                        "validate_range": {
                            "status": "warning",
                            "message": "Depth value outside range of 1 and 30"
                        }
                    },
                    "observers": {
    ... trimmed ...
}
```

In the record's detailed validations section, validations that have a status of *warning* can be suppressed by changing the validation's status to *ignore* and re-saving the record. Based on the example above, the depth warning can be suppressed to result in:

```
... trimmed ...
"depth": {
    "validate_range": {
        "status": "ignore",
        "message": "Depth value outside range of 1 and 30"
    }
},
... trimmed ...
```

The save/validate process continues until the overall record validation status is *ok*. The validated record is now ready to submit.

5.3 Submit

Submitting a collect record moves the complete and valid record from its "editing" stage to a finalized stage. The record can be submitted by calling the collect record's submit endpoint. The moved record is now:

1. included in MERMAID reporting
2. only available for edits by *admin* project users

For details on how to call Save, Validate, and Submit please refer to the Collect Record Endpoints.

CHAPTER

SIX

AGGREGATED VIEWS

MERMAID aggregated views are convenience resources that roll up many lookups and aggregate data at various levels with standardized calculations of common indicators, such as biomass. All aggregated views are read-only (i.e., support only `GET` requests). As with regular *Project resources*, aggregated view resources in MERMAID all begin, relative to the API root, with `/projects/<project_id>/`, where `<project_id>` is the UUID of a project. See *Using the API* for how to determine a `project_id` manually, or use the API to retrieve a list of project ids to which a user has access using the */projects/* resource.

All views will return data in one of three formats:

- `.../` (i.e., default) or `.../json/`: standard JSON, with `content-type = application/json`
- `.../csv/`: comma-separated 2D matrix (some fields are JSON), with `content-type = text/csv`
- `.../geojson/`: returns GeoJSON suitable for loading into a GIS (`content-type = application/json`)

To request a CSV version of the resource, append `.../csv/`, e.g.
`/projects/<project_id>/beltfishes/obstransectbeltfishes/csv/`
To request a GeoJSOn version of the resource, append `.../geojson/`, e.g.
`/projects/<project_id>/beltfishes/obstransectbeltfishes/geojson/`

Aggregated view resources tend to support a wider array of filtering methods, including range (min/max) filters and, in particular, the ability to filter by multiple values, e.g.
`?management_rule=periodic%20closure,gear%20restriction`

Note: Multiple-value comma-separated filters are marked with an asterix (*) in the resources below.

6.1 Covariates

All aggregate views include a `covariates` field with a growing list of metrics retrieved from non-MERMAID sources for the referenced MERMAID site. The idea is to speed up analysis by including commonly regressed drivers with primary ecological data. To begin with, two covariates from our friends at the Allen Coral Atlas are included; a sample looks like this:

```
"covariates": [
  {
```

(continues on next page)

(continued from previous page)


```
    "id": "75963cf9-d72a-47f9-972f-f6879c3fba17",
    "name": "aca_geomorphic",
    "value": [
      {
        "area": 1950.2176,
        "name": "Inner Reef Flat"
      }
    ],
    "display": "Alan Coral Atlas Geomorphic",
    "datestamp": "2021-02-01",
    "requested_datestamp": "2021-02-01"
  },
  {
    "id": "55c2f2d9-7168-49ee-a05a-0f65c9e777b4",
    "name": "aca_benthic",
    "value": [
      {
        "area": 1815.5039000000002,
        "name": "Sand"
      },
      {
        "area": 134.71373,
        "name": "Rubble"
      }
    ],
    "display": "Alan Coral Atlas Benthic",
    "datestamp": "2021-02-01",
    "requested_datestamp": "2021-02-01"
  }
],
```


6.2 Data sharing policies

Access to project data for all unauthenticated requests is based on the data sharing policy attached to each survey method for that project. Each survey method (e.g., fish belt transect) may be assigned one of three policies: `private`, `public summary` (default), and `public`. The three policies are summarized in the table below.

In practice, the data sharing policies mean that a user has access to data if they are authenticated and a member of the requested project, or unauthenticated and the relevant data sharing policy for the requested survey method and project is:

1. observation and sample unit views: `public`
2. sample event views: `public summary`
3. summary site view: unauthenticated; survey method-specific aggregations for each result if `public summary` or `public`

Project-level information	Private	Public summary (default)	Public
Contact info Organization, admin name and admin email	✓	✓	✓
Metadata Project name and notes, country, site name and location, survey date, depth, habitat (reef zone, reef type and exposure), management regime name, # of transects	✓	✓	✓
Site-level averages accessible via summary API:			
Average benthic cover, %	✗	✓	✓
Average total reef fish biomass, kg/ha	✗	✓	✓
Average habitat complexity scores	✗	✓	✓
Transect-level observations can be downloaded:			
Benthic observations and growth forms	✗	✗	✓
Reef fish species, size and abundance, taxonomy and functional group information, biomass coefficients	✗	✗	✓
Individual habitat complexity scores	✗	✗	✓

6.3 Observation views

Observation views are the lowest level of MERMAID aggregate views, representing individual observations with all related data from site, project, management regime, and so on followed and relatively flattened into single rows, with row-level indicators like biomass calculated in a standardized way. The csv variants of observation views are what are called from the frontend `Export to CSV` button.

All aggregate-view resources at the observations level return data if **either**

1. user is authenticated and a member of the requested project, or
2. user is unauthenticated and the relevant data sharing policy for the requested survey method and project is `public`

All aggregate-view resources at the observations level support the following filters:

- `site_id` *
- `site_name` *
- `country_id` *
- `country_name` *
- `tag_id` *
- `tag_name` *
- `reef_type`

- reef_zone
- reef_exposure
- management_id *
- management_name *
- sample_event_id *
- sample_date_before/sample_date_after
- management_est_year_min/management_est_year_max
- management_size_min/management_size_max
- management_party *
- management_compliance *
- management_rule *
- current_name *
- tide_name *
- visibility_name *
- label *
- depth_min/depth_max
- relative_depth *
- observers *

6.3.1 /projects/<project_id>/beltfishes/obstransectbeltfishes/

Flattened fish belt transect observations for a project. Additional filters:

- transect_len_surveyed_min/transect_len_surveyed_max
- reef_slope *
- transect_number
- fish_taxon *
- fish_family *
- fish_genus *
- trophic_group *
- trophic_level_min/trophic_level_max
- functional_group *
- vulnerability_min/vulnerability_max
- size_min/size_max
- count_min/count_max
- biomass_kgha_min/biomass_kgha_max

6.3.2 /projects/<project_id>/benthiclits/obstransectbenthiclits/

Flattened benthic LIT transect observations for a project. Additional filters:

- transect_len_surveyed_min/transect_len_surveyed_max
- reef_slope *
- transect_number
- length_min/length_max
- benthic_category
- benthic_attribute
- growth_form

6.3.3 /projects/<project_id>/benthicpits/obstransectbenthicpits/

Flattened benthic PIT transect observations for a project. Additional filters:

- transect_len_surveyed_min/transect_len_surveyed_max
- reef_slope *
- transect_number
- interval_size_min/interval_size_max
- interval_min/interval_max
- benthic_category
- benthic_attribute
- growth_form

6.3.4 /projects/<project_id>/habitatcomplexities/obshabitatcomplexities/

Flattened habitat complexity transect observations for a project. Additional filters:

- transect_len_surveyed_min/transect_len_surveyed_max
- reef_slope *
- transect_number
- interval_min/interval_max
- score

6.3.5 /projects/<project_id>/bleachingqcs/obscoloniesbleacheds/

Flattened number of colonies bleached quadrat collection observations for a project. Additional filters:

- `quadrat_size`
- `benthic_attribute`
- `growth_form`
- `count_normal_min/count_normal_max`
- `count_pale_min/count_pale_max`
- `count_20_min/count_20_max`
- `count_50_min/count_50_max`
- `count_80_min/count_80_max`
- `count_100_min/count_100_max`
- `count_dead_min/count_dead_max`

6.3.6 /projects/<project_id>/bleachingqcs/obsquadratbenthicpercents/

Flattened quadrat percent benthic cover observations for a project. Additional filters:

- `quadrat_size`
- `quadrat_number`
- `percent_hard_min/percent_hard_max`
- `percent_soft_min/percent_soft_max`
- `percent_algae_min/percent_algae_max`

6.4 Sample Unit views

Note: In MERMAID it is possible to have two separate sample units that differ in metadata only by the `label` property; one scenario where this commonly happens is when a transect is surveyed in two passes, one for "big fish" and one for "little fish". One advantage of the aggregated sample unit views is that they provide standardized grouping logic for calculating aggregated indicators such as biomass.

All aggregate-view resources at the sample unit level return data if **either**

1. user is authenticated and a member of the requested project, or
2. user is unauthenticated and the relevant data sharing policy for the requested survey method and project is `public`

All aggregate-view resources at the sample unit level support the same base filters as *observations views*.

6.4.1 /projects/<project_id>/beltfishes/sampleunits/

Flattened fish belt sample units with calculated biomass for a project. Additional filters:

- `transect_len_surveyed_min/transect_len_surveyed_max`
- `reef_slope` *
- `transect_number`
- `biomass_kgha_min/biomass_kgha_max`

6.4.2 /projects/<project_id>/benthiclits/sampleunits/

Flattened benthic LIT sample units with calculated calculated percent cover by benthic category for a project. Additional filters:

- `transect_len_surveyed_min/transect_len_surveyed_max`
- `reef_slope` *
- `transect_number`

6.4.3 /projects/<project_id>/benthicpits/sampleunits/

Flattened benthic PIT sample units with calculated percent cover by benthic category for a project. Additional filters:

- `transect_len_surveyed_min/transect_len_surveyed_max`
- `reef_slope` *
- `transect_number`
- `interval_size_min/interval_size_max`

6.4.4 /projects/<project_id>/habitatcomplexities/sampleunits/

Flattened habitat complexity sample units with calculated average scores for a project. Additional filters:

- `transect_len_surveyed_min/transect_len_surveyed_max`
- `reef_slope` *
- `transect_number`
- `score_avg_min/score_avg_max`

6.4.5 /projects/<project_id>/bleachingqcs/sampleunits/

Flattened bleaching quadrat collection sample units with calculated averages for both colony count and percent benthic cover observations for a project. Additional filters:

- `quadrat_size`
- `count_genera_min/count_genera_max`
- `count_total_min/count_total_max`
- `percent_normal_min/percent_normal_max`

- percent_pale_min/percent_pale_max
- percent_bleached_min/percent_bleached_max
- quadrat_count_min/quadrat_count_max
- percent_hard_avg_min/percent_hard_avg_max
- percent_soft_avg_min/percent_soft_avg_max
- percent_algae_avg_min/percent_algae_avg_max

6.5 Sample Event views

MERMAID sample event views aggregate all data collected for a given survey method at a particular place on a particular date, providing a standardized calculation of aggregate metrics.

All aggregate-view resources at the sample event level return data if **either**

1. user is authenticated and a member of the requested project, or
2. user is unauthenticated and the relevant data sharing policy for the requested survey method and project is `public summary` or `public`

All aggregate-view resources at the sample event level support the following filters:

- site_id *
- site_name *
- country_id *
- country_name *
- tag_id *
- tag_name *
- reef_type
- reef_zone
- reef_exposure
- management_id *
- management_name *
- sample_event_id *
- sample_date_before/sample_date_after
- management_est_year_min/management_est_year_max
- management_size_min/management_size_max
- management_party *
- management_compliance *
- management_rule *
- current_name *
- tide_name *
- visibility_name *

6.5.1 /projects/<project_id>/beltfishes/sampleevents/

Aggregated view of all fish belt transect data collected for a sample event. Additional filters:

- `biomass_kgha_avg_min/biomass_kgha_avg_max`
- `sample_unit_count_min/sample_unit_count_max`
- `depth_avg_min/depth_avg_max`

6.5.2 /projects/<project_id>/benthiclits/sampleevents/

Aggregated view of all benthic LIT transect data collected for a sample event. Additional filters:

- `sample_unit_count_min/sample_unit_count_max`
- `depth_avg_min/depth_avg_max`

6.5.3 /projects/<project_id>/benthicpits/sampleevents/

Aggregated view of all benthic PIT transect data collected for a sample event. Additional filters:

- `sample_unit_count_min/sample_unit_count_max`
- `depth_avg_min/depth_avg_max`

6.5.4 /projects/<project_id>/habitatcomplexities/sampleevents/

Aggregated view of all habitat complexity transect data collected for a sample event. Additional filters:

- `sample_unit_count_min/sample_unit_count_max`
- `depth_avg_min/depth_avg_max`
- `score_avg_avg_min/score_avg_avg_max`

6.5.5 /projects/<project_id>/bleachingqcs/sampleevents/

Aggregated view of all bleaching quadrat collection data collected for a sample event. Additional filters:

- `sample_unit_count_min/sample_unit_count_max`
- `depth_avg_min/depth_avg_max`
- `quadrat_size_avg_min/quadrat_size_avg_max`
- `count_genera_avg_min/count_genera_avg_max`
- `count_total_avg_min/count_total_avg_max`
- `percent_normal_avg_min/percent_normal_avg_max`
- `percent_pale_avg_min/percent_pale_avg_max`
- `percent_bleached_avg_min/percent_bleached_avg_max`
- `quadrat_count_avg_min/quadrat_count_avg_max`
- `percent_hard_avg_avg_min/percent_hard_avg_avg_max`
- `percent_soft_avg_avg_min/percent_soft_avg_avg_max`

- percent_algae_avg_avg_min/percent_algae_avg_avg_max

6.6 Summary views

MERMAID provides two "summary" endpoints that aggregate metrics from **all** surveys associated with a Site, either for a specific Sample Event (i.e. on the same date) or for all dates. For each sample event or site, a `protocols` field contains an object for each survey method conducted at that site, with calculated indicators for each if the data sharing policy for that survey method is `public summary` or `public`, and just `sample_unit_count` otherwise. These views additionally differ from other aggregated views because:

1. They are not project-specific; urls are relative to the API root. Thus `/sampleevents/` provides data for **all** surveys associated with each Site, while `/projects/<project_id>/beltfishes/sampleevents/` provides just beltfish data for a particular project.
2. They never require authentication
3. They are not refreshed immediately; under the hood, they draw from tables that are refreshed every 30 minutes.

Available filters:

- site_id *
- site_name *
- country_id *
- country_name *
- tag_id *
- tag_name *
- reef_type
- reef_zone
- reef_exposure
- management_id *
- management_name *
- project_id *
- project_name *
- project_admins *
- date_min/date_max
- data_policy_beltfish
- data_policy_benthiclit
- data_policy_benthicpit
- data_policy_habitatcomplexity
- data_policy_bleachingqc

6.6.1 /summarysampleevents/

Provides aggregated results for each survey conducted at a given place on a given date.

6.6.2 /summarysites/

Provides aggregated results for each survey conducted at a given place, across all dates. This resource is used by the MERMAID public dashboard.

CHAPTER

SEVEN

DEVELOPER DOCUMENTATION

For information on installing and running this code locally, please see the repository README.

CHAPTER

EIGHT

CONTRIBUTING TO THE MERMAID API

MERMAID is an open-source project, and needs your help! If you are a developer who can work with Django or just Python, and are interested in contributing your time, please give us a shout at contact@datamermaid.org.

All pull requests are welcome at https://github.com/data-mermaid/mermaid-api/. We have a lot on our roadmap, but because many tasks require thorough working knowledge of the system, we are particularly interested in help with items that do not require huge amounts of onboarding. Some current high-level areas in which we could use help are:

- improving this documentation, and creating swagger documentation
- unit tests
- writing analysis libraries or SDKs, particularly Jupyter notebook demos – the Python equivalent of mermaidr
- QGIS plugin/toolbar to provide a user-friendly way for users to connect to the MERMAID API, authenticate, choose which projects and survey methods to query, and then load that data with default symbologies