# Tethys MATLAB Interface Cookbook

Tethys, Antioch mosaic, 3rd century from Baltimore Museum of Art

Ally Rice – Scripps Institution of Oceanography
Heidi Batchelor – Scripps Institution of Oceanography
Marie A. Roch – San Diego State University

# Table of Contents

# 1 Getting Started

MATLAB can be used as a client to interface with the Tethys database. Common tasks include adding data to the database and pulling data from the database based on specified criteria, such as time, location, or species. This document provides a list of available functions and examples of how to use some of the most useful/common functions.

In many cases, the reader will wish to provide parameters relevant to their own data, our convention is to highlight these values.

The examples provided below use the most common MATLAB functions for interfacing with Tethys. The Appendix includes a more complete list of functions with brief definitions. You can also access these definitions from the MATLAB command line by typing:

```
>>help function_name
```

OR

```
>>doc function_name
```

## 1.1 Set your MATLAB path

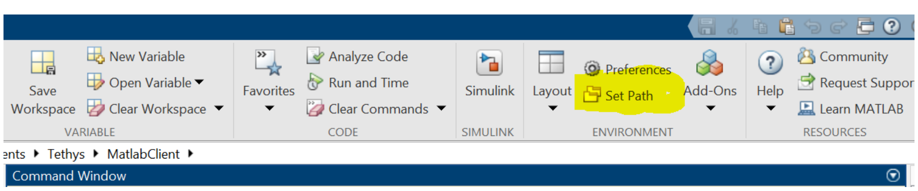
1. Start the Tethys server or verify that it is already running.
2. Locate the *MatlabClient* folder.

   The *MatlabClient* folder will contain the subfolders *db*, *db/c*, and *vis*. The functions under *db* are related to accessing the database while the functions in the *vis* directory provide support for visualizing data.

3. Start MATLAB

4. Add the *db*, *db\c*, and *vis* directories to your path

   a. This can be done using (1) MATLAB’s 'Set Path' tool, located in 'Home' tab, to save the path for the next time you start MATLAB, or




   b. using the `addpath` command in the *startup.m* file that is executed when MATLAB starts. Add the following to your *MATLAB\startup.m* file:

```
% Set path to MatlabClient appropriately
clientroot = "…\Tethys\MatlabClient";
% Add folders we need to the search path
for p = ["db", "db\c", "vis"]
  addpath(fullfile(clientroot, p))
end
```

where the …\Tethys denotes the path to your Tethys folder. See the MATLAB documentation for more details. We recommend using approach b. as it is more robust when you upgrade your version of MATLAB.

## 1.2 Set a query handle object

All calls that interact with the Tethys database require a query handle object to be passed as the first argument. The query handle object lets the functions know where the server is and defines the communication protocol.

If done at a command prompt, the handle is valid for the life of the MATLAB session (unless variables are cleared).

Any variable name can be used for the query handle object. All subsequent examples assume that a query handle has been set up and it has the name `query_h`. The name should be consistent, but need not be "query_h" and any valid variable name may be used.

To use the default server:

```
query_h = dbInit();
```

To use a specific server:

```
query_h = dbInit("Server", "YourServerName");
```

If the server is running on a different port than the default one, the Port option may be used to set the port.

## 1.3 Set species abbreviations

Tethys uses Latin species names by default. Many organizations use abbreviations such as "Lo" or "Lobl" for Pacific white-sided dolphin (*Lagenorhynchus obliquidens*) or common names. The query handler supports methods for setting how Tethys expects species names to be written in queries and how they should be displayed in output. `query_h.SetSpeciesIdInput`, `query_h.SetSpeciesIdOutput` and `query_h.SetSpeciesIdInputOutput` permit you to specify how species names will be input to functions and how they should be returned. Like the `dbInit` function, this only needs to be called once per MATLAB session. Before using these, we can explore our options for abbreviations using `dbSpeciesAbbreviations(query_h)`:

```
>> dbSpeciesAbbreviations(query_h)

1×4 string array

    "NOAA.NMFS.v1"  "NOAA.NMFS.v2". "NOAA.NMFS.v3"  "SIO.SWAL.v1"
```

If you want to explore the abbreviations for one of these options, use another query to return a table:

```
>>nmfs_v1 = dbSpeciesAbbreviations(query_h, "NOAA.NMFS.v1")

55×4 table
```

| completename | tsn | coding | Group |
|---|---|---|---|
| {'Balaenoptera borealis' } | 1.8053e+05 | {'Bb' } | {0×0 double} |
| {'Balaenoptera brydei' } | 6.126e+05 | {'Be' } | {0×0 double} |
| {'Balaenoptera musculus' } | 1.8053e+05 | {'Bm' } | {0×0 double} |
| {'Balaenoptera physalus' } | 1.8053e+05 | {'Bp' } | {0×0 double} |
| {'Eschrichtius robustus' } | 1.8052e+05 | {'Er' } | {0×0 double} |
| {'Megaptera novaeangliae' } | 1.8053e+05 | {'Mn' } | {0×0 double} |
| {'Balaenoptera acutorostrata'} | 1.8052e+05 | {'Ba' } | {0×0 double} |
| {'Eubalaena glacialis' } | 1.8054e+05 | {'Eg' } | {0×0 double} |
| {'Eubalaena japonica' } | 6.1259e+05 | {'Ej' } | {0×0 double} |
| {'Eubalaena australis' } | 5.5277e+05 | {'Ea' } | {0×0 double} |
| {'Balaena mysticetus' } | 1.8053e+05 | {'BM' } | {0×0 double} |
| {'Mysticeti' } | 5.523e+05 | {'UW' } | {0×0 double} |
| {'Delphinus delphis' } | 1.8044e+05 | {'Dd' } | {0×0 double} |
| {'Delphinus capensis' } | 5.5565e+05 | {'Dc' } | {0×0 double} |
| {'Grampus griseus' } | 1.8046e+05 | {'Gg' } | {0×0 double} |
| {'Globicephala macrorhynchus'} | 1.8047e+05 | {'Gm' } | {0×0 double} |
| {'Lissodelphis borealis' } | 1.8045e+05 | {'Lb' } | {0×0 double} |
| … | | | |

The `completename` column is the Latin species name and `tsn` is the Integrated Taxonomic Information System's taxonomic serial number for the species (https://itis.gov/). Tethys always stores species using TSNs and translates them back and forth to representations that humans can understand. The species abbreviations can also contain Group entries which allow us to distinguish different groups of animals within the same taxonomic rank.

We recommend that anthropogenic sounds be encoded as *Homo sapiens* (with a distinguishing call type, e.g., "pile driving"), and that geophonic sounds be encoded as Other phenomena, an entry that has been inserted into the ITIS encodings with a value of -10.

We can use the query handler to set a species representation to be used (Latin, a vernacular name such as "blue whale," or "Ballena azul," or an abbreviation defined in a species abbreviation table such as the one above). Note that ITIS does not have vernacular entries for many obscure species and frequently only has vernacular entries in English. When a vernacular entry is not present, the Latin name is substituted.

For example, if you want to input species in English vernacular, Latin, or the NOAA.NMFS.v1 abbreviation set, we could use one of the following:

- `query_h.setSpeciesIdInput("Vernacular", "English")`
- `query_h.setSpeciesIdInput("Latin")`
- `query_h.setSpeciesIdInput("Abbrev", "NOAA.NMFS.v1")`

Note that the system does not currently check for a valid vernacular or abbreviation, although this is likely to change in future releases.

The query handler functions setSpeciedIdOutput and set SpeciesIdInputOutput may be used in the same manner to set how the results of queries or both input and output, e.g.:

- `query_h.setSpeciesIdInputOutput("Abbrev", "SIO.SWAL.v1")`
- `query_h.setSpeciesIdOutput("Abbrev", "SIO.SWAL.v1")`

and functions getSpeciesIdInput, getSpeciesIdOutput, and getSpeciesIdInputOutput will report the current settings, e.g.

```
>> query_h.getSpeciesIdInputOutput()
Input: SIO.SWAL.v1 Output: SIO.SWAL.v1
```

## 2 README: Important note about examples

Examples in this document are designed to work on the demonstration database that is provided with the distribution of Tethys. Results will vary when used with a different database, but the "recipes" given in this cookbook can be easily adapted for different situations.

All examples in this document use the species abbreviation set "SIO.SWAL.v1". **For examples that involve querying species, you MUST set the input abbreviation to SIO.SWAL.V1, or queries will not match any results, causing many of the examples given here to fail.** Assuming query_h contains a query handler, the result of a call to dbInit, one would type:

```
query_h.setSpeciesIdInputOutput("Abbrev", "SIO.SWAL.v1")
```

Setting abbreviations need only be done once per Matlab session. If you wish to set a permanent species abbreviation (Input or Output), the commands to set abbreviations can be placed in the user's startup.m file (type: `doc startup` for details on the startup file.) along with creation of the query handler.

Remember that the Tethys server **must be started** for the examples to work. If the server is not started, an incorrect server address is given, or the server is blocked by network security (firewall) protocols, you will see an error message produced by any routine that tries to connect to the server. In such cases, the first line of the error backtrace should state "Connection refused: connect." In such cases, discuss the problem with your Tethys administrator of the Tethys development staff.

## 2.1 Release Notes

***Tethys 3.2***

Tethys 3.2 introduces methods to communicate with multiple ERDDAP servers. Users create an object for each server they wish to communicate with and can search or retrieve data from that server. Ephemeris data for civil twilight can now be computed without requiring a Mapping Tool license.

***Tethys 3.1***

*Tethys 3.1* has moved setting and retrieval of TSN translation into the query handler that is returned by dbInit. The old dbSpeciesFmt functions still work, but please use the species handler equivalents for new code. In addition, the localization schema has changed significantly to conform with recommendations from ASA/ANSI working group 7 which is developing standards for bioacoustics metadata. If you are managing localization data with an earlier version of Tethys (most of our users are not), please consult with us about migration of your data.

***Tethys 3.0***

*Tethys 3.0* has changed how MATLAB communicates with the database server. In most cases, this should be transparent to users. User-visible changes to retrieving information from the database are:

- Selection and return criteria can now be specified either as a complete path (`"Detections/Effort/Kind/SpeciesId"`) or partially (`"SpeciesId"`). If the criteria are ambiguous, an error message will be returned that describes the ambiguous keyword and how it might be interpreted. When possible, Tethys attempts to guess the right thing to do. For example, in the context of a detection effort query, we would interpret `"SpeciesId"` as `"Detections/Effort/Kind/SpeciesId"` and when querying detections, we would look for on-effort detections (`"Detections/Effort/OnEffort/SpeciesId"`) of the specified species.
- `dbDeploymentInfo` is being replaced by `dbGetDeployments` to be consistent with the other data retrieval functions. `dbDeploymentInfo` still works but is

accompanied by a warning message indicating that it will be removed in a future release.

- `[timestamps, info] = dbGetDetections`; now returns two possible outputs instead of 3 (`[timestamps, EndP, info]`). End times are optional for detections as some detectors only produce start times. When detections with and without end times are the result of a query, the second column of timestamps may be invalid. The second output (`EndP`) previously contained a vector of booleans which indicated which rows of timestamps (`timestamps`) contained valid end times. As this can easily be determined by:
  ```
  valid = ~ isnan(timestamps(:,2));
  ```
  the second argument (`EndP`) was removed.

***Tethys 2.5***

Users of Tethys prior to *2.5* will notice a change in how results are returned. Some values in Tethys can occur more than once. For example, when recording call types that occurred in a 15-minute bin, one might want to report both A and B calls for blue whales. In the past, if only one call was reported, the field name would contain the string, and if more than one call was reported, a cell array would be returned. This meant that sometimes one would reference the data as “Call”, and other times as “Call{1}” and “Call{2}”. To avoid this complication, all values are now returned as cell arrays.

In *Tethys 2.5*, the ERDDAP server is changed from coastwatch.pfeg.noaa.gov to upwell.pfeg.noaa.gov to maintain a more comprehensive NOAA-wide catalog.

# 3 Querying the Database

Throughout these examples, we will be using input arguments to refine our database queries. Some of the common arguments used in database queries are:

| Argument | Definition | Example |
|---|---|---|
| Id | Unique identifier for the record | "Id", "CINMS28-C" |
| Project | Name of the project data is associated with. | "Project", "CINMS" |
| Site | Name of the site data was collected at. | "Site", "C' |
| DeploymentId | Integer describing the Nth deployment of an instrument at a site, or the Nth deployment cruise (user discretion). Can frequently be used to restrict a query to certain deployments without specifying explicit dates. To match multiple deployments, use a vector with the desired values or relational operators | "DeploymentId", 28<br><br>"DeploymentId", [28:35]<br><br>"DeploymentId", [28 32 27]<br><br>"DeploymentId", {">=", 35} |

| | | |
|---|---|---|
| | that are described later in this document. | |
| SpeciesId | The species name. Values depend on dbSpeciesFmt. | "SpeciesId", "Bm" |
| Group | Attribute of SpeciesId, usually used to denote a population or potential species within a taxonomic group when the species determination is uncertain. | "@Group", "BW70" |
| Call | The type of call. | "Call", "Whistles" |
| Subtype | The call subtype, if applicable. | "Call", {"Subtype", "Whistles<5kHz"} |
| Granularity | The type of effort (i.e., binned, call, or encounter). | "Granularity", "binned" |
| BinSize_m | For binned effort, the length of the bin in minutes. | "BinSize_m", 60 |

As seen in the examples above, these arguments consist of keyword-value pairs: 'keyword', 'value'. Most examples will use this format. However, we can also form queries using the path to the keyword within the overarching schema (a 'schema' being either Calibrations, Deployments, Detections, Ensembles, Events, or Localizations).

For example, to use "Project" as an argument in our query we can either use:

```
"Project", "ProjectName"
```

or

```
"Deployment/Project", "ProjectName"
```

To look for detections of a given species we could use:

```
"SpeciesId", "SpeciesName"
```

or

```
"Detections/OnEffort/Detection/SpeciesId", "SpeciesName"
```

For a complete list of arguments, and to better understand the underlying schema, use `dbOpenSchemaDescription(query_h, "Detections")`, where `Detections` is the top-level name in the schema you want to view, e.g. Calibration, Detections, Localize, or Localization. Note that the top-level name is not necessarily the same name as the collection to which the element belongs (e.g., Localize is part of Localizations) and that the Tethys server must be running.

**CAVEAT: Character array vs. string**

If you are new to MATLAB, you may not be familiar with the difference between the character array type and the string type. Character arrays are sets of characters in single quotes, e.g., 'character array'. In contrast, strings are sets of characters in double quotes, e.g. "string". There are subtle differences between the two and in most places character arrays or strings can be used as arguments. The main difference occurs when these values are aggregated. Strings can be treated as a normal array:

```
s = ["Save the", "vaquita"];
```

However, character arrays can only be aggregated using the [ ] notation if they are all of the same size. Instead, the cell array notation { } can be used to aggregate these.

```
s = {'Save the', 'vaquita'};
```

These two appear similar, but they are referenced differently. To access the second element, one would use s(2) in the first case and s{2} in the second one. When possible, we recommend that the string notation be used.

## 3.1 Deployments

To retrieve deployment information from the database, use the function `dbGetDeployments`.

### 3.1.1 Retrieve information for all deployments

If we wanted a list of all deployments in the database

```
DeploymentInfo = dbGetDeployments(query_h)
```

The result would be something similar to:

```
  1×857 struct array with fields:

    Deployment
```

Let's examine the information for one deployment:

```
DeploymentInfo.Deployment(392)

ans =

  struct with fields:

                  Id: {'SOCAL40-R'}
             Project: {'SOCAL'}
        DeploymentId: {[40]}
                Site: {'R'}
              Cruise: {'Socal40'}
            Platform: {'mooring'}
              Region: {'Socal'}
          Instrument: [1×1 struct]
```

```
      SamplingDetails: [1×1 struct]
     QualityAssurance: [1×1 struct]
                 Data: [1×1 struct]
    DeploymentDetails: [1×1 struct]
      RecoveryDetails: [1×1 struct]
              Sensors: [1×1 struct]
```

Notice that most fields are cell arrays. This is needed as some fields can be repeated more than once and it can be difficult for users to know which fields are eligible to be repeated and which cannot.

Let us drill down to see that this deployment was sampled at 200 kHz with 16 bit resolution:

```
DeploymentInfo.Deployment(392).SamplingDetails.Channel.Sampling.R
egimen

ans =

  struct with fields:

         TimeStamp: {[7.3434e+05]}
    SampleRate_kHz: {[200]}
        SampleBits: {[16]}
```

The `TimeStamp` is a MATLAB serial date (use `datestr` to see a human interpretable representation) indicating when we started this sampling regimen. It is rare, but possible to change sampling regimens in the middle of a deployment, and if this had happened, there would have been additional `Regimen` entries.

### 3.1.2 Get a list of projects from retrieved deployment info

From the deployment list generated previously, let us find all the individual projects

```
% Convert cell array of Project to a string array
% DeploymentInfo.Deployment.Project returns multiple values,
% putting it inside the cell array notation { } lets us group
% these into a cell array of character strings.
projects = string({DeploymentInfo.Deployment.Project});
% Return an array of distinct project names
unique(projects)
```

Note that we could have done this all with a single line by using `arrayfun` to apply the operation to a function. To do this, we need to use what MATLAB calls an anonymous function which has the format:

```
@(arg) expression(arg)
```

Here's an example summing two numbers whose values have been truncated:

```
truncsum = @(x, y)  floor(x) + floor(y);

truncsum(3.9, 4.8)
```

The return value for this example would be 7.

We call `arrayfun` with two arguments (see `doc arrayfun` for more details):

1. The function to be applied. In this case, we want to extract Deployment.Project from DeploymentInfo, so we use @(d) d.Deployment.Project.
2. The array to which we wish the function applied.

The call:

```
string(unique(...
   arrayfun(@(dep) dep.Project, DeploymentInfo.Deployment)))
```

will return the same values as the previous calls shown above.

### 3.1.3 Retrieve deployment info for specific projects and/or sites

If we wanted to find deployment info only for deployments within the project 'SOCAL' and at site 'H', we could use the following query:

```
atH = dbGetDeployments(query_h, "Project", "SOCAL", "Site", "H");
```

To select multiple sites (or other query parameters), an array of values can be used, e.g., "Site", ["H", "N"]. Additional arguments could be used to restrict to a specific latitude range (see next example), species, etc.

### 3.1.4 Find all deployments in a specific latitude range

Often studies are limited to specific geographic areas. Querying the deployment information in the Tethys database can provide a list of deployments for a given range of Latitude and Longitude.

```
deployments = dbGetDeployments(query_h, ...
   "DeploymentDetails/Latitude",{'>', 45}, ...
   "DeploymentDetails/Latitude",{'<', 60});
```

## 3.2 Detections

### 3.2.1 Detection Effort

To retrieve detection effort information from the database, use the function `dbGetDetectionEffort`.

#### 3.2.1.1 *Retrieve effort for a specific project*

To retrieve the effort for a specific project, use the query:

```
[effort, details] = dbGetDetectionEffort(query_h, "Project",
"SOCAL");
```

Additional arguments could be used to restrict the effort to a specific time period, geographic location, species, etc.

The returned variable `effort` is a two-column matrix where each row contains a list of start and end effort times as MATLAB serial dates. This is primarily to support older Tethys client code, new code may find the data in details easier to work with. The effort can be converted to a readable format using `datestr`:

```
for idx=1:size(effort, 1);
   fprintf('%s %s\n',
   datestr(effort(idx,1)),datestr(effort(idx,2)))
end
```

The returned structure `details` contains information about the type of effort. Within `details`, `effort_table` is a table sorted by effort start time. It shows the effort start and end timestamps as easily readable Matlab datetime objects, the detection document Id, and a RecordIdx. The RecordIdx can be used to match the start and end times with other elements of details, such as the kinds_table. The kinds_table contains a row for each type of detection effort that was done for a specific effort. As an example, in one query, the first few rows of the effort table might look like this (some fields are omitted for brevity):

| Start | End | Id | RecordIdx |
|---|---|---|---|
| 13-Jan-2009 06:00:00 | 08-Mar-2009 11:41:26 | {'SOCAL31M_LF_logs_jsb'} | 1 |
| 11-Mar-2009 00:00:00 | 04-May-2009 06:00:00 | {'SOCAL32M_LF_logs_lkr'} | 2 |
| 17-May-2009 00:00:00 | 21-May-2009 07:00:27 | {'SOCAL33M_LF_logs_lmm1'} | 3 |

The kinds_table within details contains a list that indicates what species we were looking for and what level of granularity was used for detection annotations. A sample table might look like this:

| RecordIdx | SpeciesId | Group | Cal | Granularity | BinSize_min | Subtype |
|---|---|---|---|---|---|---|
| 1 | "Bryde's whale" | {0×0 double} | "Be4" | "binned" | 60 | <missing> |
| 1 | "Blue Whale" | {0×0 double} | "A" | "binned" | 60 | <missing> |
| 1 | "Blue Whale" | {0×0 double} | "B" | "binned" | 60 | <missing> |

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | “Blue Whale” | {0×0 double} | “D” | “binned” | 60 | <missing> |
| 1 | “Fin Whale” | {0×0 double} | “20Hz” | “binned” | 60 | <missing> |
| | : | : | : | : | : | : |
| 15 | “Other” | {0×0 double} | “Motorboat” | “binned” | 60 | <missing> |
| 16 | “Human” | {0×0 double} | “Active Sonar” | “encounter” | NaN | “MFA<5kHz” |
| 16 | “Human” | {0×0 double} | “Active Sonar” | “encounter” | NaN | “Echosounder” |
| 16 | “toothed whales” | {0×0 double} | “Whistles<5kHz” | “encounter” | NaN | <missing> |

The RecordIdx from the effort table matches the RecordIdx entries in the effort. Consequently, we can see that between Jan 13, 2009 and Mar 8, 2009 (RecordIdx 1), we looked for Bryde’s whale Be4 calls, Blue whale A/B/D calls, and Fin whale 20 Hz calls. For these calls, we see that the effort was binned with a bin size of 60 min, meaning that analysts only reported presence/absence within hourly bins. We see other types of effort in the table, such as encounter-level effort for anthropogenic signals and toothed whales, and that the detections are recorded as the start and end of a set of calls (an acoustic encounter).

Other fields within the details structure include deployments, a set of deployment identifiers corresponding to effort table record indices, detection document Ids, and the raw data returned from the query.

Note that text values, or strings, are accessed with curly brackets. In some cases, there may be more than one value (although not in this example), so the strings are formed in a format that supports multiple values. These are called cell arrays, see the MATLAB documentation if you wish more information about cell array structures.

#### *3.2.1.2 Retrieve effort for a specific deployment*

This example is very similar to the previous one, except that we are further limiting our search. In this case, we are interested in the deployment with an Id of “SOCAL38-M”:

```
[effort, details] = dbGetDetectionEffort(query_h, ...
   "DeploymentId", "SOCAL38-M");
```

One can find deployment ids using any of the interactive tools such as the web client, data explorer, executing dbGetDeployments, or visiting the URL TETHYSSERVER/Deployments, where TETHYSSERVER is the address of your Tethys server.

Alternatively, one can add criteria related to the deployment. In this case, we specify the site and a deployment number associated with the deployment of interest. Tethys finds the appropriate deployment (potentially sets of deployments) and then finds all effort associated the deployment. We do this by querying Deployment/Site and Deployment/DeploymentId.

```
[effort, details] = dbGetDetectionEffort(query_h, "Project",
"SOCAL", ...
            "Site", "M", "Deployment/DeploymentId", 38);
```

As there is a Detections/DataSource/DeploymentId field which contains "SOCAL-38M," we needed to clearly indicate that the DeploymentId number which indicates the $n^{th}$ deployment or cruise to that site should be looked for within the deployment record by specifying the parent of DeploymentId: Deployment/DeploymentId. dbGetDetectionEffort always tries to resolve field names within the Detections schema first. Specifying Deployment/DeploymentId ensured that the criterion was evaluated with respect to the deployment record rather than the detections one.

Caveat: there are sometimes multiple efforts for the same species. For example, running two different detectors for the same species can result in duplicate effort. Therefore, when performing analyses on data, be careful that you don't double-count detections. When querying effort (or detections), you can always specify queries for a specific type of effort (see the function's help).

### 3.2.2 Detections

To retrieve detection information from the database, use the function `dbGetDetections`.

#### *3.2.2.1 Retrieve detections for a specific project/site/species*

To retrieve all detections of killer whales at Site BD from the Aleut Project:

```
[detections, info] = dbGetDetections(query_h, "Project", "Aleut",
"Site", "BD", "SpeciesId", "Oo");
```

The returned variable `detections` will be a one- or two-column matrix of detection start and end times as MATLAB serial dates. Whether `detections` contains one or two columns depends upon whether an end time was recorded for the detection. Some detection algorithms only record the start time of a detection. If we wanted to easily read the detection dates/times:

```
dates = dbSerialDateToISO8601(detections);
```

The returned variable `info` contains information about the returned detections. `deployments` provides the deployment Ids for the detections. `data` contains information about the detection start/end times and species information, while `detection_table` provides the same information in a more user-friendly format. `Id` provides the detection document filename.

#### 3.2.2.2 Retrieve detections for a species group

It is possible to query for specific group attributes. This is necessary for many beaked whale signals for which species identification is unknown.

To retrieve all detections of an unidentified beaked whale signal (BW43 in this case) we could use the query:

```
detections = dbGetDetections(query_h, '@Group', 'BW43');
```

Note the use of '@' required to query an attribute. This is also required for a query formatted using paths:

```
detections = dbGetDetections(query_h, ...
      'OnEffort/Detection/SpeciesId/@Group', 'BW43');
```

**CAVEAT:** Table creation for attributes is not yet supported.

#### 3.2.2.3 Retrieve detections for a given date and time range

To request all the detections for a given date and time range in the database for toothed whales:

```
detections = dbGetDetections(query_h, 'SpeciesId', 'UO', ...
  'Effort/Start',{'>', '2001-10-17T19:09:00Z'}, ...
  'Effort/End',{'<', '2009-05-19T00:00:00Z'} );
```

If you have a datetime object or a MATLAB serial date (datenum) that needs to be concerted to a string, use the function dbSerialDateToISO8601 that converts from serial dates or datetime objects to ISO 8601 date and time text.

## 3.3 Localizations

Effort for localizations can be accessed using function dbGetLocalizationEffort, and the location/direction information can be retrieved from dbGetLocalizations.

Selection criteria are handled as with other queries by using keyword/value pairs. To examine the localization schema, use:

```
% Localize is the root element of documents in the localization
% collection.  Typing the collection name will also work.
dbOpenSchemaDescription(q, "Localize")
```

### 3.3.1 Localization Effort

Function dbGetLocalizationEffort returns a structure that describes localization effort within the database. Like the other database retrieval functions, the query handler must be the first argument and subsequent arguments provide constraints on what is returned.

For example, to return effort for tracks recorded in WGS84 longtiude and latitude, one could specify:

```
loceff = dbGetLocalizationEffort(q, "Subtype", "Geographic", ...
 "Name", "WGS84", "LocalizationType", "Track")
```

Assuming that this matched data, loceff will be a structure with two fields:

- data – MATLAB structures corresponding to the XML returned from the database
- effort – A table showing the deployment or ensemble identifier, the start and end of the effort, and information about the type of localization effort. Due to the constraints of the query, this would only be for Geographic/WGS84 coordinate systems and would only report efforts to track a series of calls or sounds.

### 3.3.2 Localizations

```
loc = dbGetLocalizations(q, ...
 "Name", "WGS84", "LocalizationType", "Track")
```

Function dbGetLocalizations returns a structure that describes any directional or positional information that meet the query criteria. Assuming that localizations have been found, a structure is returned with the following fields:

- data – MATLAB structures corresponding to the XML returned from the database
- loc – A table showing information about individual localizations. In addition to fields describing the localization information, DataSourceSet is a many-to-one index that can be matched to DataSourceSet in the groups table below.
- groups – A table indicating the Id of localization documents and the data source (DeploymentId or EnsembleId) with which they are associated.

Example tracks from the above query:

```
>> loc.loc(1:5,:)

ans =

  5×9 table

    LocalizationSet         TimeStamp           Track_TimeStamps    Coordinates_Longitude    Coordinates_Latitude    NorthWest_Longitude
NorthWest_Latitude    SouthEast_Longitude    SouthEast_Latitude
    _______________    ____________________    ________________    _____________________    ____________________    ___________________
__________________    ___________________    __________________

           1           10-Jan-2012 20:23:55    [1×36 datetime]         [1×36 double]            [1×36 double]             199.96
22.811                200.05                 22.789
           1           10-Jan-2012 20:32:09    [1×69 datetime]         [1×69 double]            [1×69 double]             200.02
22.458                200.12                 22.403
           2           12-Jan-2012 17:55:58    [1×19 datetime]         [1×19 double]            [1×19 double]             199.95
22.276                199.97                 22.265
           2           12-Jan-2012 18:41:50    [1×36 datetime]         [1×36 double]            [1×36 double]             200.14
22.777                200.16                 22.747
           2           12-Jan-2012 19:28:23    [1×38 datetime]         [1×38 double]            [1×38 double]             199.99
22.486                200.09                 22.453
```

## 3.4 Diel plots

### 3.4.1 Retrieve day/night information and produce diel plot with defined times and coordinates

Set the start (`Time1`) and end (`Time2`) times for the period you want day/night information for:

```
Time1 = "10-Jan-2011 00:00:22";
Time2 = "27-Feb-2011 15:00:22";
```

Convert the times to MATLAB serial dates:

```
starttime = datenum(Time1);
endtime = datenum(Time2);
```

Set the latitude and longitude (in this example we're using a location near the Aleutian Islands):

```
Latitude = 52.7234;
Longitude = 174.7654;
```

Run a query to retrieve the sunset/sunrise information for the times and location you defined:

```
night = dbDiel(query_h, Latitude, Longitude, starttime, endtime);
```

`night` will contain the sunset and sunrise at the specified coordinates in MATLAB serial dates.

If a plot is desired, we can use the function `visPresence`.

Because all database times are in UTC, we will first set a UTC offset so that our plot will be in local time for this position close to the Near Islands (part of the Aleutian island chain) and uses the Hawai'i – Aleutian time zone.

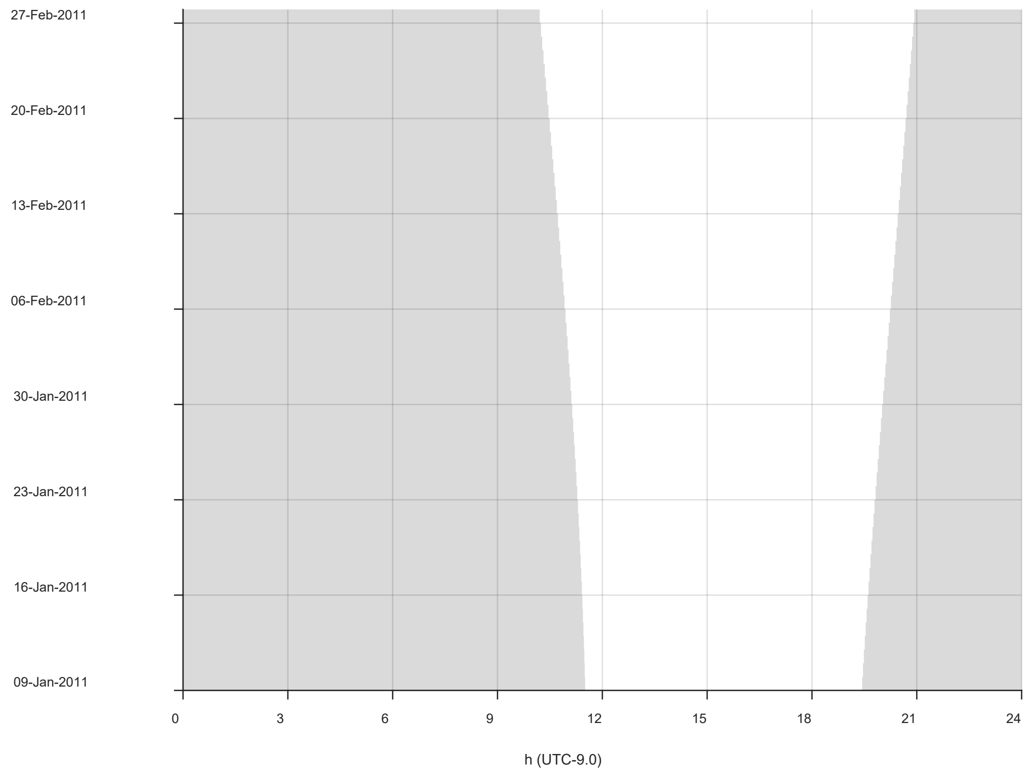
```
UTCOffset = -10;
```

We could have used dbTimeZone to compute the offset using nautical timezones which ignore geopolitical boundaries: dbTimeZone(query_h, Latitude, Longitude, 'nautical'), although this would return an offset of 12 hours.

We can plot the day/night pattern with `visPresence` and our `night` variable (output from `dbDiel`) as an argument:

```
nightH = visPresence(night, "Color", "black", "LineStyle",...
   "none", "Transparency", .15, "Resolution_m", 1/60, ...
```

```
"DateRange",[starttime, endtime], "UTCOffset", UTCOffset);
```




**Example code to copy/paste:**

```
% Retrieve day/night information and produce diel plot with defined
% times and coordinates

% Set the start (Time1) and end (Time2) times for the period you want
% day/night information for:
Time1 = "10-Jan-2011 00:00:22";
Time2 = "27-Feb-2011 15:00:22";

% Convert the times to MATLAB serial dates:
starttime = datenum(Time1);
endtime = datenum(Time2);

% Set the latitude and longitude (in this example we're using a
% location near the Aleutian Islands):
Latitude = 52.7234;
Longitude = 174.7654;

%Run a query to retrieve the sunset/sunrise information for the times
and
%location you defined:
night = dbDiel(query_h, Latitude, Longitude, starttime, endtime);
% night will contain the sunset and sunrise at the specified
```

```
% coordinates in MATLAB serial dates.

% If a plot is desired, we can use the function visPresence.

% Because all database times are in UTC, we will first set a UTC
% offset so that our plot will be in local time.
UTCOffset = -9;

% Then we will run visPresence using the night variable (
% output from dbDiel) as an argument:
nightH = visPresence(night, "Color", "black", "LineStyle",...
    "none", "Transparency", .15, "Resolution_m", 1/60, ...
    "DateRange",[starttime, endtime], "UTCOffset", UTCOffset);
```

### 3.4.2 Retrieve day/night information automatically and produce a diel plot with detections

Let's produce a diel plot by automatically pulling the location and times from the deployment(s) across which we are querying.

Define the query parameters:

```
project = "Aleut";
deployment = 2;
species = "Oo";
```

Query for detections matching the defined parameters:

```
detections = dbGetDetections(query_h, "Project", project, ...
   "Deployment/DeploymentId", deployment, "SpeciesId", species);
```

Find the first and last detection time:

```
starttime = min(detections(:, 1));
endtime = max(detections(:, 2));
```

Query for deployment coordinates:

```
sensor = dbGetDeployments(query_h, "Project", ...
   project, "Deployment/DeploymentId", deployment);
```

Query for sunset/sunrise information:

```
night = dbDiel(query_h,...
      sensor(1).Deployment.DeploymentDetails.Latitude{1},...
      sensor(1).Deployment.DeploymentDetails.Longitude{1},...
      starttime, endtime);
```

Plot the data, as in the previous example:

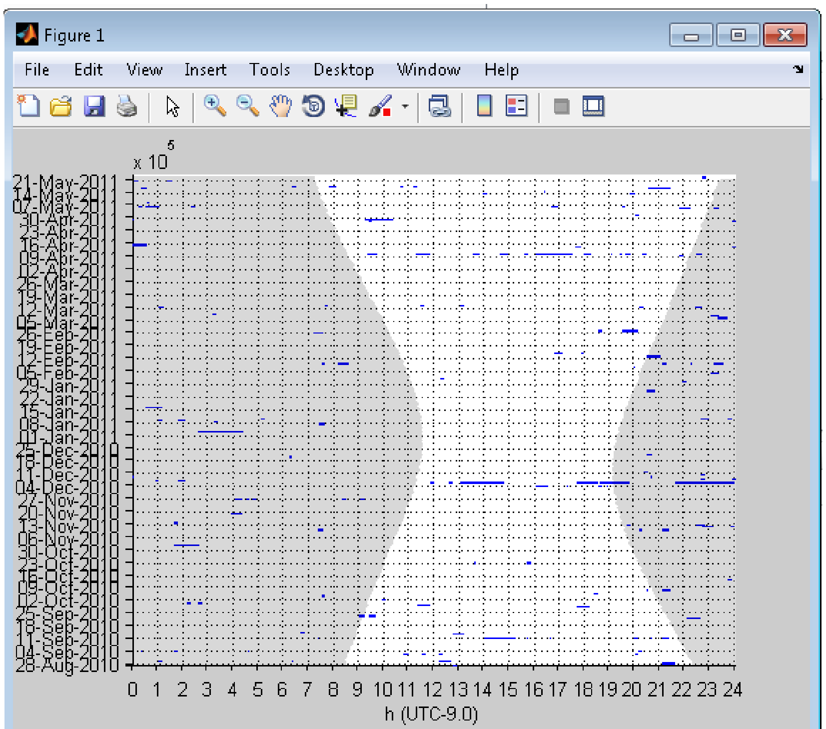
```
UTCOffset = -9;
```

```
nightH = visPresence(night, "Color", "black", "LineStyle",...
   "none", "Transparency", .15, "Resolution_m", 1/60, ...
   "DateRange",[starttime, endtime], "UTCOffset", UTCOffset);
```

This time, let's add in the killer whale detections we queried:

```
speciesH = visPresence(detections, "Color", "b", ...
    "Resolution_m", 5, "UTCOffset", UTCOffset);
```

Note that if you have dates that describe your detection effort, they can be passed to visPresence with the ‘Effort’ keyword. Portions of time that do not have effort will be grayed out. See the function documentation for details, or look at the example in dbDemo case 5 which creates a local time diel plot.




**Example code to copy/paste:**

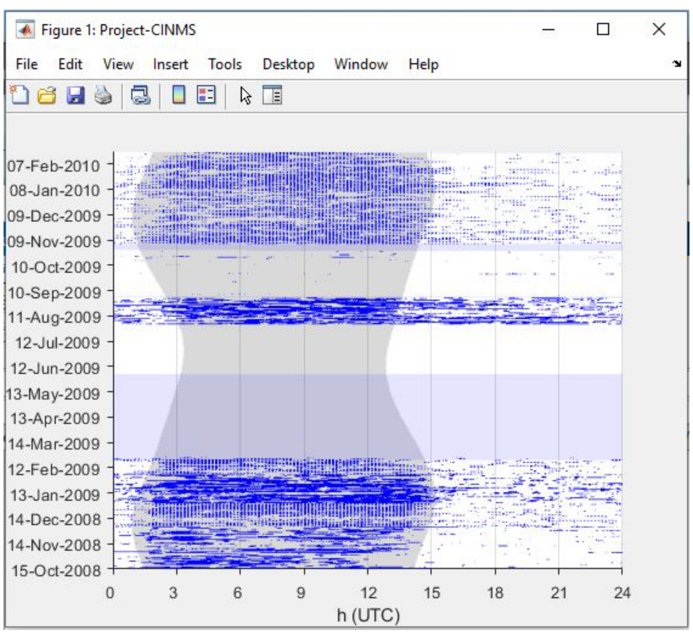
```
%Produce a diel plot by automatically pulling the location and times
%from the deployment(s) across which we are querying.

%Define the query parameters:
project = "Aleut";
deployment = 2;
species = "Oo";

%Query for detections matching the defined parameters:
detections = dbGetDetections(query_h, "Project", project, ...
   "Deployment/DeploymentId", deployment, "SpeciesId", species);
```

```
%Find the first and last detection time:
starttime = min(detections(:, 1));
endtime = max(detections(:, 2));

%Query for deployment coordinates:
sensor = dbGetDeployments(query_h, "Project", ...
   project, "DeploymentId", deployment);

%Query for sunset/sunrise information:
night = dbDiel(query_h,...
    sensor(1).Deployment.DeploymentDetails.Latitude{1},...
    sensor(1).Deployment.DeploymentDetails.Longitude{1},...
    starttime, endtime);

%Plot the data, as in the previous example:
UTCOffset = -9;

nightH = visPresence(night, "Color", "black", "LineStyle",...
   "none", "Transparency", .15, "Resolution_m", 1/60, ...
   "DateRange",[starttime, endtime], "UTCOffset", UTCOffset);

%This time, let's add in the killer whale detections we queried:
speciesH = visPresence(detections, "Color", "b", ...
    "Resolution_m", 5, "UTCOffset", UTCOffset);
```

### 3.4.3 Produce a long-term diel plot

We can produce a long-term plot to visualize data. In this example, we will create a diel plot containing all data for a specified project:

```
dbYearly(query_h, "Project", "CINMS");
```



This plot could be made more specific with additional arguments (project, species, call, etc.). The default for `dbYearly` is to include shading for nighttime in the plot but this can be turned off if desired (use the arguments: "Diel", false). With `"Diel", true`, `dbYearly` also outputs sunrise/sunset times. Note: the vertical striped pattern in the example plot is the result of some deployments having a duty cycle.

### 3.4.4 Produce a diel plot with lunar illumination

#### *3.4.4.1 The time interval and coordinates are set explicitly*

Define the parameters for the query:

```
Time1 = "10-Jan-2011 00:00:22";
Time2 = "27-Jun-2011 15:00:22";

starttime = datenum(Time1); endtime = datenum(Time2);

Latitude = 52.7234;
Longitude = 174.7654;
```

Set the time interval over which we will check (interval minutes must evenly divide 24 hours; must be no more than 30 min):

```
interval = 30;
```

Smaller intervals sample illumination at a higher resolution, but take longer to compute. Use `dbGetLunarIllumination` to get moon illumination:

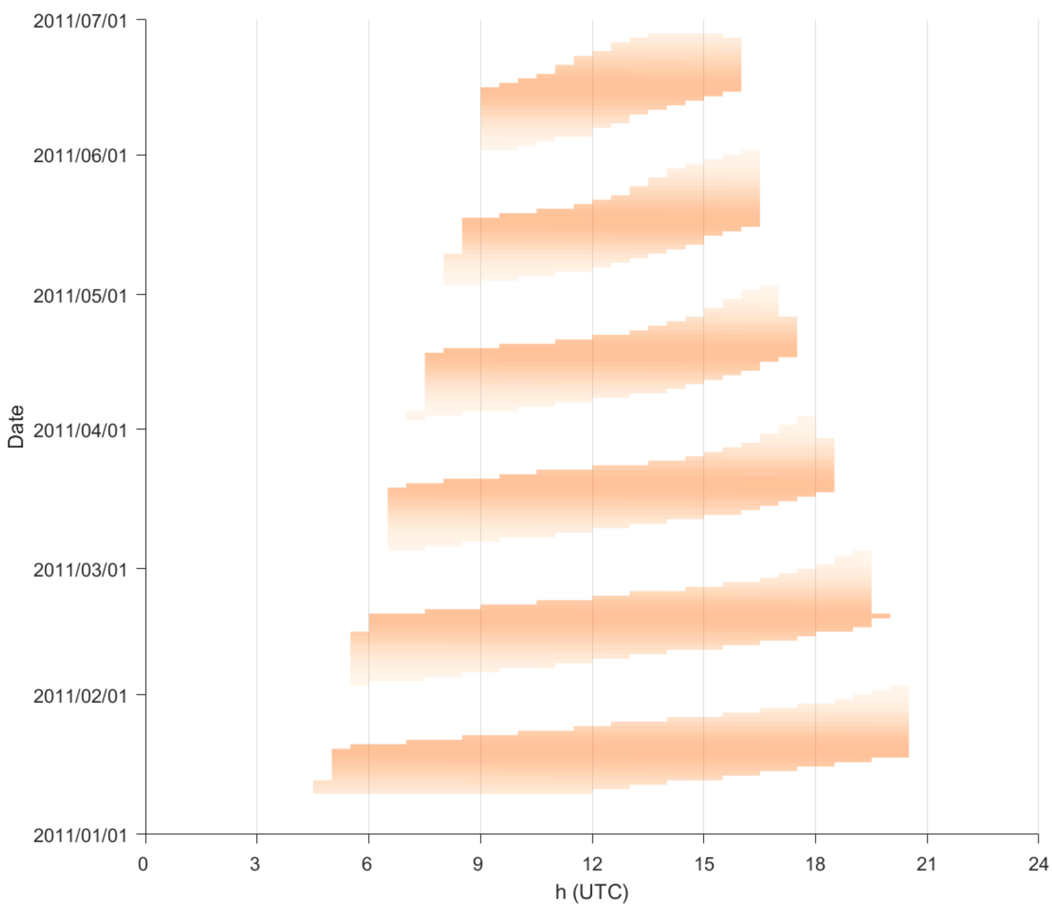
```
illu = dbGetLunarIllumination(query_h, Latitude, Longitude,...
   starttime, endtime, interval);
```

The output will have MATLAB serial dates in column 1 and percent lunar illumination in column 2.

**Example code to copy/paste:**

```
%Retrieve lunar illumination data with explicitly set time period and
%coordinates

%Define the parameters for the query:
Time1 = "10-Jan-2011 00:00:22";
Time2 = "27-Jun-2011 15:00:22";

starttime = datenum(Time1); endtime = datenum(Time2);

Latitude = 52.7234;
Longitude = 174.7654;

%Set the time interval over which we will check (interval minutes must
%evenly divide 24 hours; must be no more than 30 min):
```

```
interval = 30;

%Use dbGetLunarIllumination to get moon illumination:
illu = dbGetLunarIllumination(query_h, Latitude, Longitude,...
   starttime, endtime, interval);

% The output will have MATLAB serial dates in column 1 and percent
% lunar illumination in column 2.  Create a new figure and use
% visLunarIllumination to display it.

figure;
visLunarIllumination(illu);  % show the illumination pattern
```




### *3.4.4.2 The time period and coordinates are retrieved automatically to produce a diel plot with detections*

Plot detections with day/night and lunar illumination shown and position and time derived from the deployment(s) across which we are querying.

Define the parameters for the query:

```
project = "Aleut";
deployment = 2;
species = "Oo";
```

Query for detections and deployment information:

```
detections = dbGetDetections(query_h, "Project", project, ...
   "Deployment/DeploymentId", deployment, "SpeciesId", species);

starttime = min(detections(:, 1));
endtime = max(detections(:, 2));

sensor = dbGetDeployments(query_h, "Project", ...
   project, "DeploymentId", deployment);

Lat = sensor.Deployment.DeploymentDetails.Latitude{1};
Long = sensor.Deployment.DeploymentDetails.Longitude{1};
```

Retrieve lunar illumination and diel information:

```
interval = 30;

illu = dbGetLunarIllumination(query_h, Lat, Long,...
   starttime,endtime, interval);

night = dbDiel(query_h, Lat, Long, starttime, endtime);
```

Make a diel plot in local time:

```
UTCOffset = -9;

nightH = visPresence(night, "Color", "black", "LineStyle",...
   "none", "Transparency", .15, "Resolution_m", 1/60, ...
   "DateRange",[starttime, endtime], "UTCOffset", UTCOffset);
```

Add the killer whale detections to the plot:

```
speciesH = visPresence(detections, "Color", "b",...
     "Resolution_m", 5, "UTCOffset", UTCOffset);
```

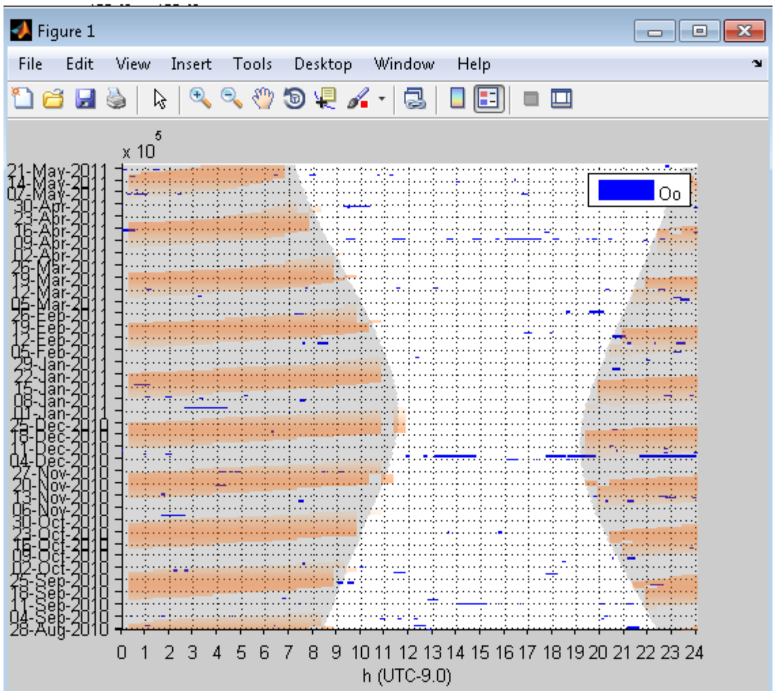
Add the lunar illumination to the plot:

```
lunarH = visLunarIllumination(illu, "UTCOffset", UtcOffset);
```

Add a legend for the species:

```
legendH = legend(speciesH(1), species);
```

Result:




## Example code to copy/paste:

```
% Plot detections with day/night and lunar illumination shown.
% Position/time derived from the deployment(s) across which
% we are querying.

%Define the parameters for the query:

project = 'Aleut';
deployment = 2;
species = 'Oo';

%Query for detections and deployment information:

detections = dbGetDetections(query_h, 'Project', project, ...
    'Deployment/DeploymentId', deployment,'SpeciesId', species);

starttime = min(detections(:, 1));
endtime = max(detections(:, 2));

sensor = dbGetDeployments(query_h, 'Project', ...
    project,'DeploymentId', deployment);

Lat = sensor.Deployment.DeploymentDetails.Latitude{1};
Long = sensor.Deployment.DeploymentDetails.Longitude{1};

%Retrieve lunar illumination and diel information:

interval = 30;
```

```
illu = dbGetLunarIllumination(query_h, Lat, Long,...
    starttime,endtime, interval);

night = dbDiel(query_h, Lat, Long, starttime, endtime);

%Make a diel plot in local time:
UTCOffset = -9;

nightH = visPresence(night, 'Color', 'black', 'LineStyle',...
    'none', 'Transparency', .15,'Resolution_m', 1/60, ...
    'DateRange',[starttime, endtime],'UTCOffset', UTCOffset);

%Add the killer whale detections to the plot:
speciesH = visPresence(detections, 'Color', 'b',...
    'Resolution_m', 5, 'UTCOffset', UTCOffset);

%Add the lunar illumination to the plot:
lunarH = visLunarIllumination(illu, 'UTCOffset', UTCOffset);

%Add a legend for the species:
legendH = legend(speciesH(1), species);
```

### 3.4.5 Convenience function for diel and lunar illumination plots

The function `visDiel` will use query input to automatically create a diel plot of detections, day/night shading, and (if desired) lunar illumination, without having to manually collect each component yourself, as shown previously.

To create the plot produced in the last example (killer whale detections in deployment 2 of the Aleutian Islands Project with night shading and lunar illumination) we could use the query:

```
visDiel(query_h, 'Project', 'Aleut', ...
   'Deployment/DeploymentId', 2, ...
   'SpeciesId', 'Oo', 'Lunar', true, 'UTC', false);
```

This function defaults to no lunar illumination, 5 min bin size, and UTC time unless specified. Within `visDiel` it is possible to modify the UTC offset and the number of date ticks on the y-axis.

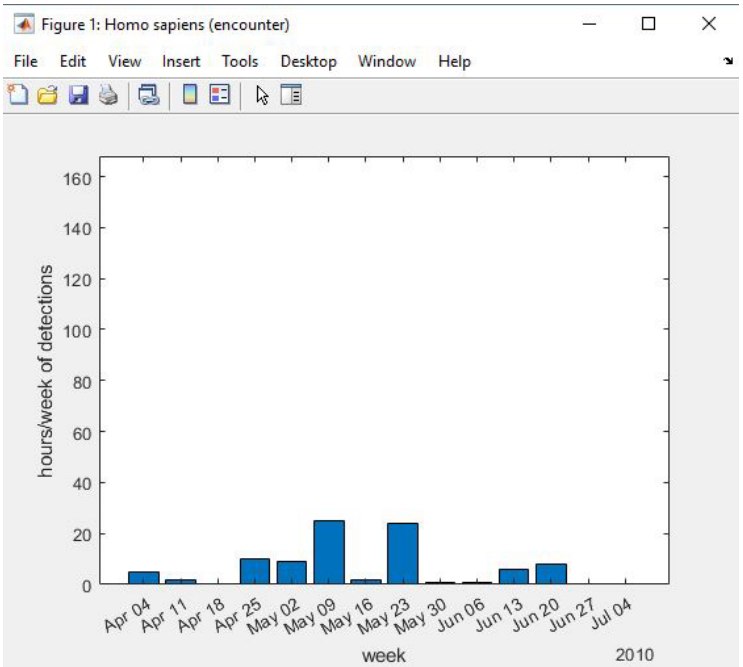
## 3.5 Weekly plots

### 3.5.1 Produce a weekly plot of detections

Plots the number of hours per week with detections for the specified criteria. Note that the weeks start on Sundays.

In this example, we'll make a weekly plot of active sonar detections from deployment 38 at SOCAL site M.

```
visWeekly(query_h, 'Project', 'SOCAL', 'Site', 'M',...
      'Deployment/DeploymentId', 38, ...
     'SpeciesId', 'Anthro', 'Call',...
      'Active Sonar', 'Granularity', 'encounter');
```




**CAVEAT:** Note that this call will have produced a warning that duplicate effort has been selected. Pay close attention when this happens as double counts can occur and display of regions of no effort may be incorrect.

# 4 Environmental Datasets

Tethys is designed to interface with the NOAA Environmental Research Division Data Access Program (ERDDAP). This allows users to choose any of the data sets hosted through ERDDAP and bring the data into MATLAB via a Tethys query. We support gridded and table data, but do not support the map service.

For more information on ERDDAP, see http://coastwatch.pfeg.noaa.gov/erddap/index.html and the Tethys Manual.

Tethys 3.2 and later replaces the old ERRDDAP functionality (dbERDDAP and dbERDDAPSearch) with a class called dbERDDAPServer. The old functionality has not been removed, but the new class provides the following advantages:

- Easily specify a specific ERDDAP server.
- Arrays of text values are string arrays instead of cell arrays.
- Timestamps are converted to Matlab datetime objects instead of serial dates, making it easier to inspect values.

ERDDAP servers are relatively easy to set up and as a consequence many institutions provide access to data using ERDDAP. The Irish Marine Institute maintains a list of a number of ERDDAP servers as well as alternative packages for interfacing with ERDDAP data.

We need access to a Tethys query handler object to create an ERDDAP object:

```
erd = dbERDDAPServer(query_h);
```

This creates the interface. The object supports three methods:

- search – look for datasets on the server
- data – retrieve data
- getServer – Report the ERDDAP server being used.

To use an ERDDAP server other than the default one, provide a second argument that contains the URL of the ERDDAP server. Usually, this is https://machine.domain/erddap, but can vary.

For instance, the NOAA NCEI server is at:

```
ncei = dbERDDAPServer(querys_h, ...
"https://www.ncei.noaa.gov/erddap");
```

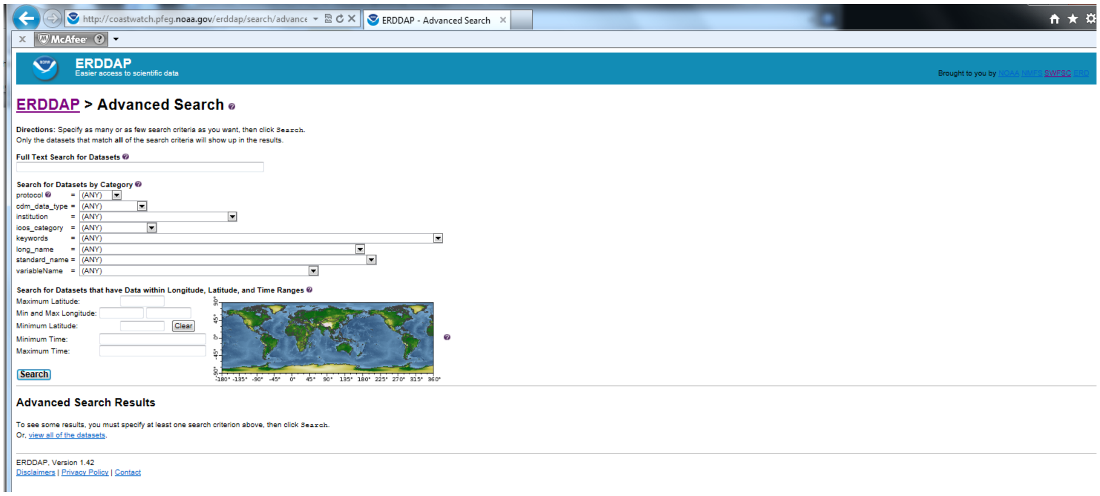
In the examples in this section, we will be using the NOAA GEO-IDE server at https://upwell.pfeg.noaa.gov/erddap.

```
upwell= dbERDDAPServer(query_h, ...
"https://upwell.pfeg.noaa.gov/erddap");
```

Find ERDDAP Datasets

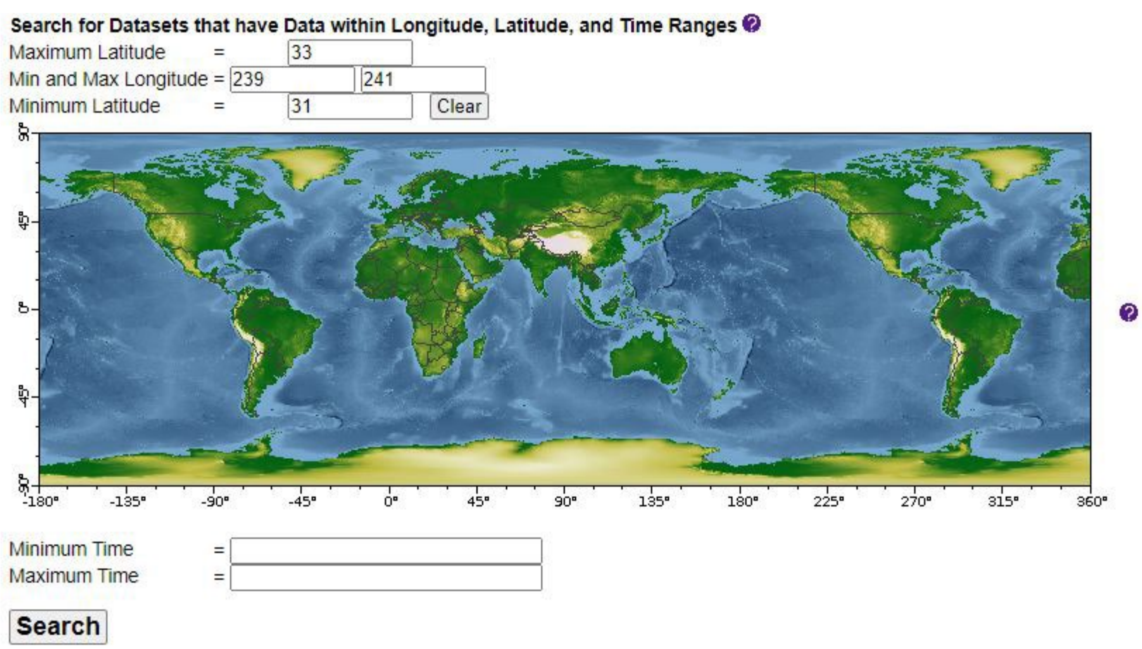
To explore available data, an Advanced Search is suggested. The ERDDAP search page can be found by going to the ERDDAP server directly. You can also use the Tethys MATLAB client to open the search page for you:

```
upwell.search();  % Open the search page to explore
```




Users can search for available data using space and/or time limits. To search spatially, users can input latitude and longitude limits, or click on the map of the earth to create a box. To search for data collected within a specified time frame, the minimum and maximum time can be added to the search.

For example, to find all of the available data for a region of the Pacific, the spatial limits are set to latitude between 31 and 33 degrees, and longitude between 239 and 241 degrees.




The results show a table of matching data sets. For more information about a specific data set, there are columns with a summary of the metadata:

- The data links in GridDAP or TableDap lead to a data access form page about the data showing the variables measured and fields that let one specify a subset of the data to download.
- The Background info link provides metadata, and the FGDC and ISO links provide metadata as XML files.  The metadata link shows a summary of the metadata in a web page.

**Advanced Search Results**

upwell.pfeg.noaa.gov/erddap

5537 matching datasets, listed in alphabetical order.  View page: 1 (current) 2 ... 6 .

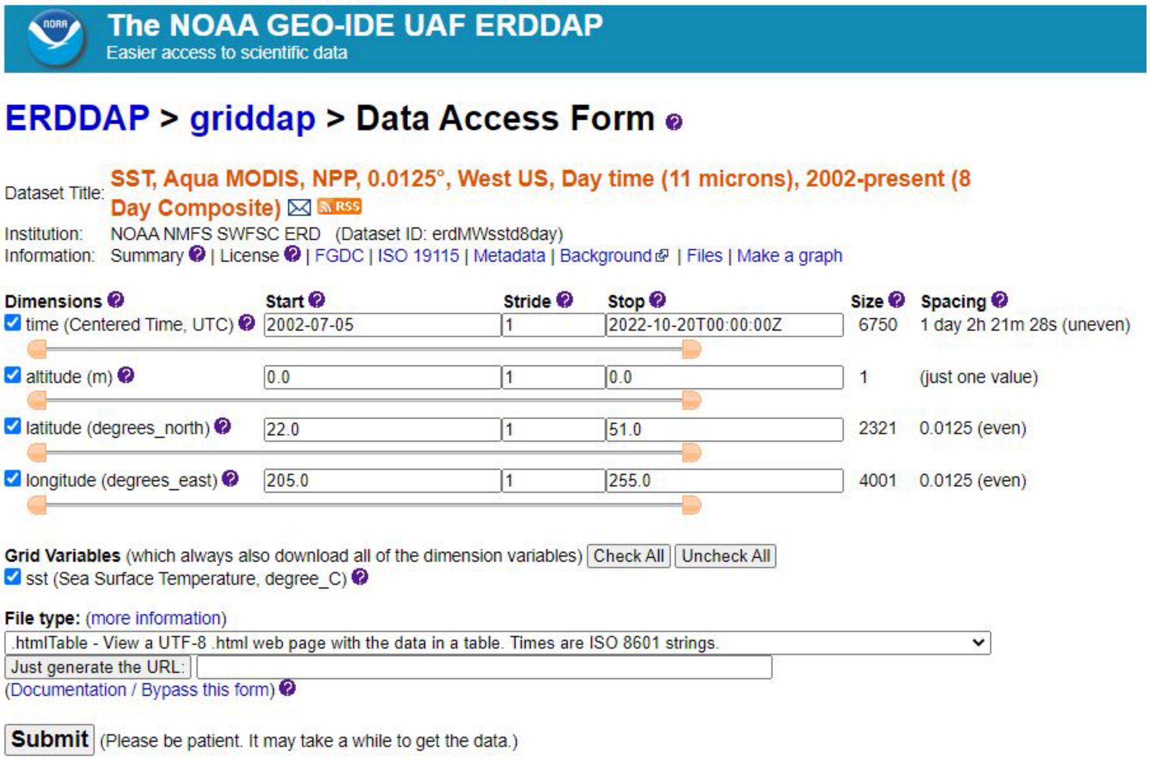
| Grid DAP Data | Sub-set | Table DAP Data | Make A Graph | W M S | Source Data Files | Acces-sible | Title | Sum-mary | FGDC, ISO, Metadata | Back-ground Info | RSS | E mail | Institution | Dataset ID |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| data | | | graph | M | | public | 4X daily ltm air from the NCEP Reanalysis (air.4Xday.1981-2010.ltm), 2.5°, 0001 | ? | F I M | background | RSS | | NOAA NWS NCEP, NCAR | noaa_psl_fc64_537a_05bc |
| data | | | graph | M | | public | 4X daily ltm air from the NCEP Reanalysis (air.4Xday.ltm), 2.5°, 0001 | ? | F I M | background | RSS | | NOAA NWS NCEP, NCAR | noaa_psl_faa8_a0e9_a4c8 |
| data | | | graph | M | | public | 4X daily ltm hgt from the NCEP Reanalysis (hgt.4Xday.1981-2010.ltm), 2.5°, 0001 | ? | F I M | background | RSS | | NOAA NWS NCEP, NCAR | noaa_psl_94d8_c6d8_26b8 |
| data | | | graph | M | | public | 4X daily ltm hgt from the NCEP Reanalysis (hgt.4Xday.ltm), 2.5°, 0001 | ? | F I M | background | RSS | | NOAA NWS NCEP, NCAR | noaa_psl_48a3_fa09_c1ba |
| data | | | graph | M | | public | 4X daily ltm omega from the NCEP Reanalysis (omega.4Xday.1981-2010.ltm), 2.5°, 0001 | ? | F I M | background | RSS | | NOAA NWS NCEP, NCAR | noaa_psl_3e61_d289_d624 |
| data | | | graph | M | | public | 4X daily ltm omega from the NCEP Reanalysis (omega.4Xday.ltm), 2.5°, 0001 | ? | F I M | background | RSS | | NOAA NWS NCEP, NCAR | noaa_psl_b907_aed9_cd87 |
| data | | | graph | M | | public | 4X daily ltm rhum from the NCEP Reanalysis (rhum.4Xday.1981-2010.ltm), 2.5°, 0001 | ? | F I M | background | RSS | | NOAA NWS NCEP, NCAR | noaa_psl_4116_d2ba_e08f |
| data | | | graph | M | | public | 4X daily ltm rhum from the NCEP Reanalysis (rhum.4Xday.ltm), 2.5°, 0001 | ? | F I M | background | RSS | | NOAA NWS NCEP, NCAR | noaa_psl_80c3_17cf_c0c2 |
| data | | | graph | M | | public | 4X daily ltm shum from the NCEP Reanalysis (shum.4Xday.1981-2010.ltm), 2.5°, 0001 | ? | F I M | background | RSS | | NOAA NWS NCEP, NCAR | noaa_psl_ccb7_3970_14e9 |
| data | | | graph | M | | public | 4X daily ltm shum from the NCEP Reanalysis (shum.4Xday.ltm), 2.5°, 0001 | ? | F I M | background | RSS | | NOAA NWS NCEP, NCAR | noaa_psl_8b17_7480_4902 |
| data | | | graph | M | | public | 4X daily ltm uwnd from the NCEP Reanalysis (uwnd.4Xday.1981-2010.ltm), 2.5°, 0001 | ? | F I M | background | RSS | | NOAA NWS NCEP, NCAR | noaa_psl_ea90_ade5_b4c4 |
| data | | | graph | M | | public | 4X daily ltm vwnd from the NCEP Reanalysis (vwnd.4Xday.1981-2010.ltm), 2.5°, 0001 | ? | F I M | background | RSS | | NOAA NWS NCEP, NCAR | noaa_psl_4654_d3f5_0c23 |
| data | | | graph | M | | public | 4x daily NCEP reanalysis (hgt.sfc.gauss), 1985 | ? | F I M | background | RSS | | NOAA NWS NCEP, NCAR | noaa_psl_947a_5b13_e9bb |

If the same geographic limits are used and the minimum/maximum time boxes are populated 2010-03-01T00:00:00Z and a maximum time of 2011-01-01T00:00:00Z, the table is filtered for data sets that span that time frame. .

The search can also be narrowed by keyword. To find a list of the keywords, see the ERDDAP website to use the pull down menu in the advanced search. By typing “sst” or “sea_surface_temperature” in the keyword drop-down, the table is filtered to datasets that include sea surface temperature.

If there is a specific dataset you want to use for analysis, make note of the DatasetID (from the last column on the right of the search return). This ID can be used with other the dbERDDAPServer class’s data method to directly download the data (see the next example).

Let's pick a dataset and look at it in more detail in ERDDAP. We will select an Aqua MODIS 8-day composite SST data with 0.0125° of resolution from NOAA (DatasetID: erdMWsstd8day). By clicking on the “data” under the first column, a complete list of variables and of the dimensions needed for a query can be viewed.




We see that the data are indexed by time, altitude, latitude, and longitude. The limits for each index variable are provided, for example, at the time of this writing these data are available between January 2006 and February 2025. The spacing tells us that the data measurements are taken about 26.5 h apart (due to the satellite's orbital path) and there are 6,750 of these nearly daily observations. There is data for exactly one altitude (sea level) and we can see the geographic extent of the data as well as the spacing between measurements (0.0125 degrees).

For the erdMWsstd8day data set, sst is the name of the variable. The required dimensions are time, altitude, latitude, and longitude. When used with the search method of the ERDDAP interface class, the DatasetID is followed by a question mark and then the variable.

In the next section, we will use erdMWsstd8day?sst is used to download sea surface temperature data. Remember that datasets are ERDDAP server specific, and that all not servers will have the same data sets.  These data are from the https://upwell.pfeg.noaa.gov/erddap server.

Once you are familiar with ERDDAP search terms, you can specify them in the `search` method of your ERDDAP server object, separating each term by an ampersand (&). For example, the Integrated Ocean Observing System (IOOS) maintains a set of data categories that include terms such as bathymetry, co2, currents, dissolved_o2, ice_distribution, etc. We can use ioos_category=bathmetry to search for bathymetry. ERDDAP's standard_name provides a wide set of variable names where spaces between words are replaced with underscores. These are reasonably intuitive, e.g. sea_surface_temperature. The ERDDAP server class has method categories that will open up a web browser page listing attributes (e.g., ioos_category) and standardized values. More details on these search terms can be found at any ERDDAP server, e.g. the NOAA GEO-IDE UAF ERDDAP server; follow the search for dataset by category links.

As an example, suppose we wished to search for sea surface temperature provided by the National Centers for Environmental Information (NCEI). We would run the query:

```
coast.search('keywords=sea_surface_temperature&institution=ncei')
```

The search method will return the web address corresponding to your search. You could cut and paste this into a browser, but this is not needed as MATLAB will open a new browser window displaying the results. The window should show a number of datasets that include sea surface temperature produce by NCEI. You can refine or change your search from this page.

## 4.1 Download ERDDAP data

### 4.1.1 Example 1

Suppose we wished to access a subset of the sea surface temperature (SST) dataset identified in the previous section: **erdMWsstd8day?sst**. We will use the upwell dbERDDAPServer object that we created in the last section that is connected to https://upwell.pfeg.noaa.gov/erddap . From the previous example, we know that the SST data are indexed by time, altitude, latitude (degrees North) and longitude (degrees East). We can use method `get` to pull in the data.

In this example, we will search for data on a small grid of the coast of southern California on November 13th, 2012. We need to specify each axis. ERDDAP requires a set of array indices indicating the portion of the dataset to retrieve. As there are four index variables, there will be four sets of array indices [ ]. Each array index must have the form

```
[start:stride:end]
```

where `start` is either an index number into the data or is specified in the units of measure, e.g. a timestamp for a time axis. For grid a grid axis of size N, indices range from 0 to N-1. When referencing by unit, you must surround the value with parentheses ( ). We indicated that we wanted to retrieve data from November 13th, 2012. We would specify this as follows using ISO8601 time notation: YYYY-MM-DDTHH:MM:SSZ where Z indicates that the time is in UTC.

```
[(2012-11-13T00:00:00Z):1:(2013-02-13T00:00:00Z)]
```

Subsequent indices are handled similarly.

```
% We created the upwell server object in the last section
% If you have closed your MATLAB, be sure to create a query
% handler and recreate the ERDDAP object:
% upwell= dbERDDAPServer(query_h, "https://upwell.pfeg.noaa.gov/erddap");

data = upwell.get('erdMWsstd8day?sst[(2012-11-13T00:00:00Z):1:(2013-02-13T00:00:00Z)][(0.0):1:(0.0)][(33.47):1:(33.59)][(240.7):1:(240.80)]');
```

The returned `data` is a structure that contains three fields:

```
Axes: [1×1 struct]
Data: [1×1 struct]
dims: [9 10 1 1]
```

- Axes – Description of the axes
- Data – A structure with the returned data.
- dims – The dimensions of the data

The Axes structure contains fields that describe the data axes:

- names – An ordered array of the axes names indicating how the returned data are organized, e.g. `data.Axes.names(1)` is “longitude” with the remaining values being latitude, altitude, and time.
- units – String array of measurement units for each axis. In this case: degrees_east, degrees_north, m, and UTC.
- types – String array of data types for the axes units. Here, all units are doubles except for the time measurements which are datetime objects.
- values – Cell array of values corresponding to grid positions on each of the axes. The names entry lets us know the order of these, so data.Axes.values{4} is a list of datetime objects defining the days The value that corresponds to points on the label axes. For example, to see the latitudes, we would examine the 2nd cell entry: `>> data.Axes.values{2}`

The Data field contains the actual data and contains the following information:

- names – String array of variables returned. As we only requested SST, `data.Data.names(1)` is 'sst'.
- units – String array indicating the unit of measurement for each variable name (degree_C in this case).
- types – String array describing the data type for each value. Here, the data were returned as type 'float'. Even though MATLAB stores these as double-precision numbers, ERDDAP stored them as single-precision numbers. If numerical precision to many digits is important to your research question, this may be important to you.
- values – A cell array with one entry per variable returned. As we only requested SST, `values{1}` contains a 9 x 10 matrix of doubles that corresponds to the temperatures we requested.

The sst data can now be plotted using the mapping toolbox or saved for use in other software packages.

Retrieval times for ERDDAP data depend on the amount of data queried, the complexity of the data set, and the current load on the server. It is not uncommon for large queries to take non-trivial amounts of time to complete.

### 4.1.2 Example 2

Here's a more complex example that finds bathymetry 400 $km^2$ around a specified point in the Southern California Bight. Note: this example requires the MATLAB mapping toolbox.

We must define our point (`center`) and the area we want bathymetry data for (`box`) by converting square kilometers (`range_km`) to degrees (`delta_deg`):

```
center = [33.515 240.753];
range_km = 20;
% If you have the Mapping toolbox: delta_deg = km2deg(range_km);
delta_deg = 0.18;
box = [center - delta_deg; center + delta_deg];
```

Then we must identify a bathymetry dataset that covers the area we've defined. We first define our criteria:

```
geospec = sprintf('minLat=%f&maxLat=%f&minLon=%f&maxLon=%f',
box(:));
criteria = ['ioos_category=bathymetry', '&', geospec];
```

And then run our query:

```
dbERDDAPSearch(query_h, criteria);
```

We see that there are 10 bathymetry data sets that might meet our purposes. We settle on the San Diego, California Tsunami Forecast Grids for MOST Model: **noaa_ngdc_ec9d_8632_6ca3** which has unevenly spaced data sampled approximately 0.017 degrees apart.

```
dataset = 'noaa_ngdc_ec9d_8632_6ca3';
geoind = sprintf('[(%f):1:(%f)][(%f):1:(%f)]', box(:));
data = upwell.get(sprintf('%s?bathy%s', dataset, geoind));
```

The bathymetry data are in `data.Data.values{1}`

**Example code to copy/paste:**

```
% Download and use ERDDAP data Find bathymetry in a 400 km2 sqaure
% centered at a specified point in the Southern California Bight

% We define a center point lat/long for our bathymetry
% and add offsets corresponding to 10 km.
center = [33.515 240.753];
% We need the change in lat/long corresponding to a 10 km offset
% in each direction.
range_km = 10;
% If we have the mapping toolbox, we can use
% delta_deg = km2deg(range_km);
% which would produce the following value:
delta_deg = 0.089932160591873;
box = [center - delta_deg; center + delta_deg];

% Identify a bathymetry dataset that covers or box
% Set our search criteria:
geospec = sprintf('minLat=%f&maxLat=%f&minLon=%f&maxLon=%f', box(:));
criteria = ['ioos_category=bathymetry', '&', geospec];

%And then run our query:
upwell.search(criteria);

% We see that there are multiple bathymetry data sets that might
% suit our purposes. We settle on the San Diego, California Tsunami
% Forecast Grids for MOST Model: noaa_ngdc_ec9d_8632_6ca3
% which has unevenly spaced data sampled approximately
% 0.017 degrees apart.
dataset = 'noaa_ngdc_ec9d_8632_6ca3';
geoind = sprintf('[(%f):1:(%f)][(%f):1:(%f)]', box(:));
data = upwell.get(sprintf('%s?bathy%s', dataset, geoind));

% The bathymetry data are in data.Data.values{1}
```

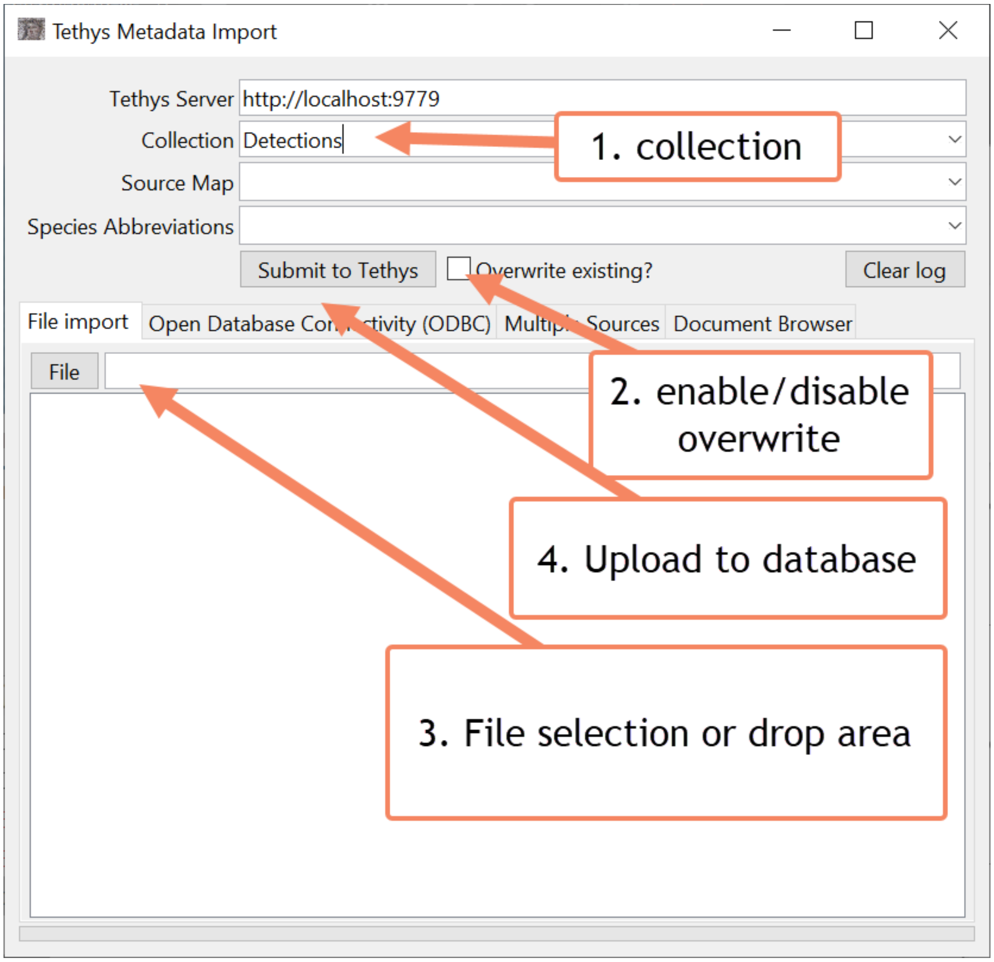
# 5 Add Files to the Database

An interface for submitting documents to the database can be accessed from MATLAB with:

```
dbSubmit();
```

To use a specific server, use:

```
dbSubmit('Server', 'yourserverName');
```



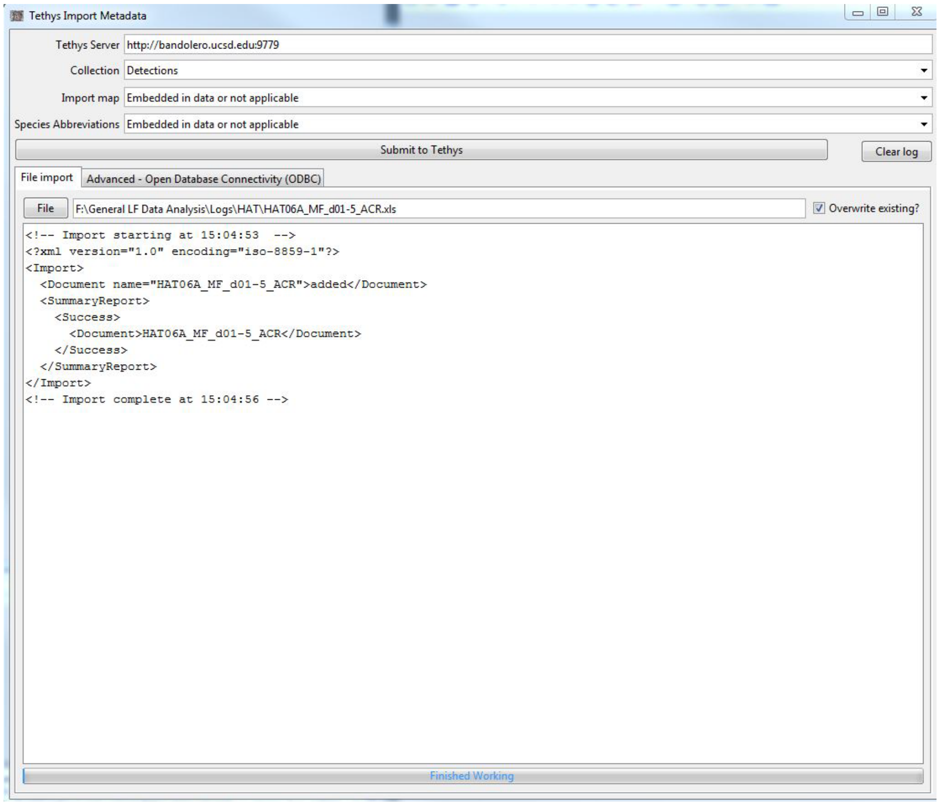

On this popup, the first input is your server address. In many cases, this will be your local host address which can be written as http://127.0.0.1:9779.

The second input is a drop-down to choose the appropriate collection to submit your document to. This includes Detections, Calibrations, Deployments, Ensembles, Localizations, Source Maps, and Species Abbreviations.

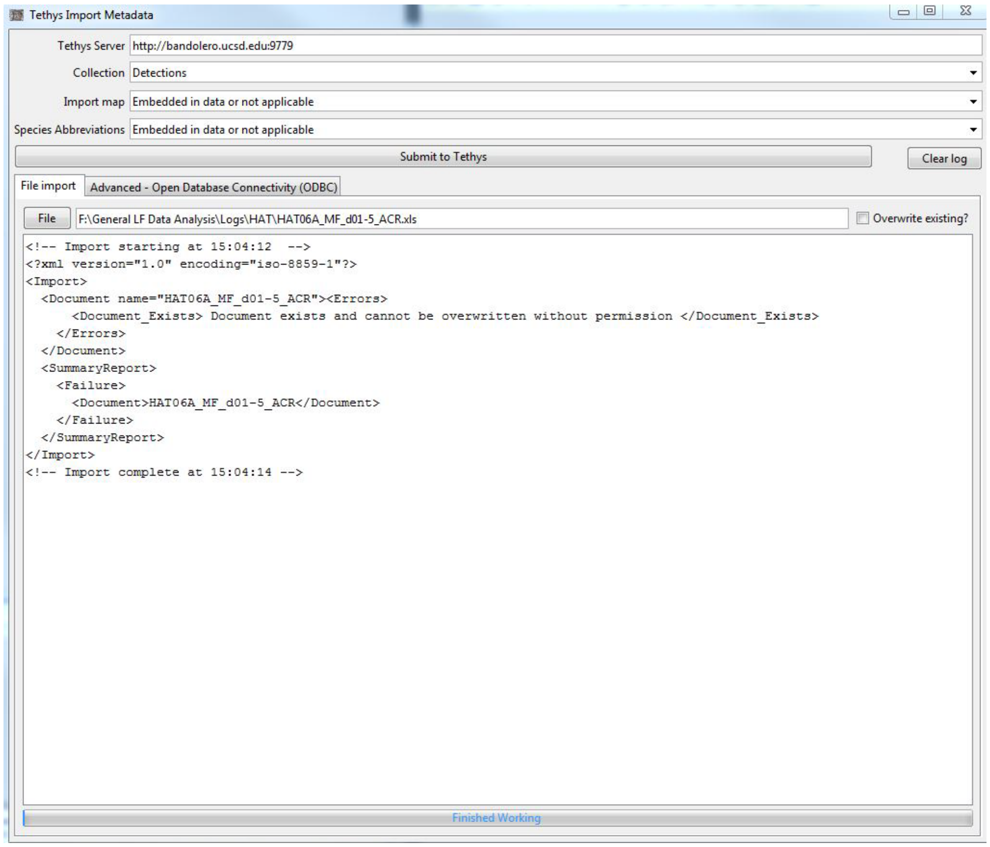
The third input is a drop-down to indicate the appropriate source map. Source maps provide directions on how data contained in the document you are submitting are mapped to Tethys when your data are not already in Tethys-ready XML format. If there is a source map listed in your input file (for example, a Detections Excel Sheet under the Metadata tab would list the parser) you can choose "Embedded in data". The Source Map needs to be part of the Tethys server, if it is new, you will need to import the Source Map first.

The fourth input is a drop-down to indicate the appropriate species abbreviations to use.

Next, there are several tabs to select the file to be added to Tethys. To add an individual file from a network location, click on the "File import " button and navigate to the file to be added to Tethys. Click the “Submit to Tethys” button and your document will be submitted, with confirmation (shown below) or errors displayed in the message areas.




If you wish to overwrite an existing Tethys document, click the overwrite existing checkbox. Otherwise, trying to submit a document twice will fail and you will receive an error message:




More details on the other tabs can be found in the Data Import manual. Briefly, 'Multiple Sources' allows one to combine data from multiple files or databases into one document. The ODBC tab allows one to import data from databases and requires that the source map contain database queries. ODBC allows one to treat many types of data as if they were a database. As an example, one can import Excel documents using this interface. The 'Document Browser' populates a tree view of submitted documents by name.

# 6 Appendix: MATLAB Function List

Note: For function definitions within MATLAB, type the following into the command window:

```
>>help FunctionName
```

or

```
>>doc FunctionName
```

For example:

```
>>help dbDemo
```

produces the output:

```
dbDemo(exampleN, OptionalArgs)
  Examples of using the Tethys database.
```

```
example - example N, see switch statement for details.
Optional keyword value pair arguments:
  'Server' - Override default server name
  'Port' - Overrride default port
  'QueryHandler', q - Use an existing query handler rather than
     a new one. Note that Server and Port arguments are ignored
    if this is specified.
  'Debug', true|false - Produce debug information for some
    plots.
```

**dbCannedReports**

Allows the user to quickly create a summary of the data returned in a query. The output includes a file of summary statistics. **dbCannedReports** differs from **dbCannedReportsLoSubtype** in that summary statistics are included.

**dbCannedReportsLoSubtype**

Allows the user to quickly create a summary of the data returned in a query.

**dbDateToOffsets**

Converts one or two columns of serial dates to day numbers and resolution_m bins. For one-column data, a second column will be added which is resolution_m minutes after the start time. For two-column data, the second column is rounded up to the start of the next bin.

**dbDemo**

Provides a demonstration of some common uses of the Tethys database and tools. The examples can be used to confirm that the software is installed and configured correctly, or as a template for the user's metadata analyses. These examples use sample data included with the Tethys download. Example 1 queries for all unidentified whale detections associated with a specific deployment and site. Example 2 queries for three types of whale calls. Example 3 displays information on the project, deployment, and site for the data used in a query. Example 4 uses the entire database, rather than a query subset, to create a summary of effort for the entire database. In example 5, a diel plot of detections for a given species, deployment, and site is produced. Example 6 returns all the detections for a given project. Example 7 lists all of the species and all of the call types found in the database. Example 8 produces a diel plot and a weekly effort plot. Example 9 produces a diel plot with detections and lunar illumination. Example 10 is an example of a query written in XQuery. Example 11 grabs chlorophyll data from ERDDAP and plots it with call presence. Example 12 is another ERDDAP demonstration, where an animation of sea surface temperature is displayed for specific coordinates. Similarly, example 13 shows an animation of wind speed.

**dbGetDeployments**

Retrieves information for specified deployments. Returns an array where each element is a structure with fields about fixed deployments. Note: This function was called **dbDeploymentInfo** in older versions of Tethys.

**dbDeployments2kml**

Writes a KML file with all deployments meeting the criteria and displays them in Google Earth. Note: this requires the MATLAB kml toolbox.

**dbDetections2XML**

Generates XML from a set of detections.

**dbDiel**

Returns the sunrise and sunset times for a specific location and date range. All times are in UTC, not local times.

**dbERDDAP (deprecated)**

Returns the results of an ERDDAP query.

**dbERDDAPSearch (deprecated)**

Searches NOAA's Environmental Research Division Data Access Program (ERDDAP) catalog for datasets matching desired parameters. Search parameters are any valid set of ERDDAP keywords. Each keyword is followed by an = sign with a search value. Multiple keywords are joined by &. Some common keywords are: bathymetry, calcofi, chlorophyll-a, goes, ice, noaa, ocean-color. For a full list of keywords go to http://coastwatch.pfeg.noaa.gov/erddap/categorize/keywords/

**dbERDDAPServer**

Class for finding environmental data sets and retrieving their data using the protocol defined by NOAA's Environmental Research Division Data Access Program (ERDDAP). ERDDAP server is specified when an object is initialized, and defaults to one chosen by the server if not specified. ERDAP servers will have different datasets. See example in this tutorial for usage details.

search – method for finding data. Search parameters are any valid set of ERDDAP attributes and standardized categorical values. Data shown in the search result will contain the specified criteria. Syntax of the search string is "attribute1=value1&attribute2=value2&…&attributeN=valueN". See method categories for determining attributes and categories.
get – method for retrieving data
getServer – return the ERDDAP server that is being used
categories – Open a browser window to explore the list of attribute names and categorical values.

**dbFindFiles**

Searches for files in the current directory or a given directory.

**dbGetCalltypes**

Given a database query, return a list of call types meeting the associated metadata and detection data predicates.

**dbGetCannedQuery**

Returns a canned (previously saved) query.

**dbGetDetections**

Retrieves all detections meeting specified criteria from database. Detections are returned as a timestamps matrix of MATLAB serial dates. The timestamps will either be single times that represent a detection within a binned interval or span a time interval.

**dbGetDetectionEffort**

Retrieves effort information from Tethys detection effort records. Effort is returned as a matrix of MATLAB serial dates containing the start and end times in each row.

**dbGetEvents**

Retrieves all events meeting specified criteria from the database. Events are returned as a timestamps matrix of MATLAB serial dates. The timestamps will either be instantaneous or span an interval.

**dbGetLunarIllumination**

Returns information from the database about the lunar illumination percentage between the start and end UTC serial timestamps (datenums) in the specified interval.

**dbGetSpecies**

Determines which species have been detected for a given expedition and site.

**dbGetUsers**

Returns a cell array with users that have detection effort.

**dbInit**

Creates a connection to the Tethys database. With no arguments, a connection is created to the default server defined within this function. Returns a handle to a query object through which Tethys queries are served.

**dbISO8601toSerialDate**

Given a cell array of ISO8601 format dates: YYYY-MM-DDTHH:MM:SS.FFFZ (e.g. 2010-02-09T07:39:22.325Z) convert to MATLAB serial dates.

**dbJavaPaths**

Makes sure Java classes on path.

**dbNormDiel**

Given a set of detections and diel information specifying nighttime, renormalizes detections to represent a 12-hour day/night period by linear interpolation. Assumes that both detections and night are sorted by timestamp and converted to local time (or in UTC with a provided UTCoffset) so that night fall is after sunrise each day. Assumes that there are no detections outside of the night intervals except for the day before and after the first and last night respectively.

**dbParseDates**

Given a set of records returned from a dbXPathDOMQuery, parses timestamp fields and returns them as a matrix of MATLAB serial dates. Each row corresponds to the timestamps associated with a single record.

**dbPresenceAbsence**

Computes presence/absence in resolution_m increments. Presence is a one or two column matrix giving starting (and possibly ending) times as MATLAB serial dates. If end time is unavailable, only the resolution_m segment containing the start time will be selected. Dates are assumed to be UTC and sorted.

**dbRelOp and dbRelOpChar**

Helper functions for translating numeric comparisons into XQuery fragments. Not intended to be called directly by the user.

**dbRemoveDocument**

Removes the specified document from the database.

**dbRemoveOverlap**

Given a matrix of row-oriented start and end dates, returns a new matrix where overlapping rows have been removed.

**dbRunQuery**

Run the query based on a query string.

**dbRunQueryFile**

Run the query based on a filename.

**dbSerialDateToISO8601**

Converts a set of MATLAB serial dates to ISO8601 format. Assumes that the dates are in UTC.

**dbSpeciesFmt**

Sets the species naming format used for XQueries (tsn, Latin name, or abbreviation) as well as how those results will be displayed.

**dbStats**

Generates statistics on daily and hourly bins with calls and percentages in regards to effort from Tethys database.

**dbSubmit**

Submits files to the database. Files may be a single filename, a cell array of filenames, or omitted in which case a GUI prompts for a single file submission.

**dbSpeciesAbbreviations**

Tethys servers maintains named lists of custom ecoding systems (species maps) for taxonomic names. This function retrieves the a list of coding system names or the mapping between taxonomic classes and abbreviations for a specific coding scheme.

**dbTimeZone**

Retrieves offset from UTC time for specified longitude and latitude.

**dbDetectionsForUser**

Returns a list of documents submitted by the specified user.

**dbXPathDomQuery**

Given a document object model representation of a document, runs an XPath query on it.

**dbYearly**

Produces a long-term plot containing all data for a given site.

**dbYearlyReport**

Generates reports from Tethys database.

**visDiel**

Convenience function for querying detections and plotting them in a diel plot.

**visCyclic**

Plot cyclic data in a polar plot with labels as specified.

**visLunarIllumination**

Parses an illumination query return and plots it on the given figure.

**visPresence**

Shows a presence/absence plot in specified increments. Dates are assumed to be UTC and sorted.

**visPresenceAbsence**

Shows a presence/absence plot in specified increments. Dates are assumed to be UTC and sorted.

**visWeekly**

Plots the number of hours per week with detections for the specified criteria. Weeks start on Sunday, which may cause slight shifts in distributions from other tools that may choose to start weeks on the first day of effort

**visWeeklyEffort**

Generates a plot of detections and effort by week for a given species. Detections can be narrowed down by call, call subtype, and/or species group. Multiple deployments for a given site can be appended to the same plot, but multiple sites will have their own plot. The right-hand y-axis will always correspond to the percentage of effort for that week, denoted by a dot if less than 100%. The left-hand axis will be either "Cumulative hours per week" for encounter granularity, or "Total Detections per week" for call granularity.