



Tethys, Antioch mosaic, 3<sup>rd</sup> century from Baltimore Museum of Art  
<http://tethys.sdsu.edu> v 3.1

Marie A. Roch – San Diego State University

Jeff Cavner, Katie O’Laughlin, and David Cardoso – San Diego State University

Simone Baumann-Pickering, Heidi Batchelor, Sean Herbert, Ally Rice, and John A. Hildebrand – Scripps  
Institution of Oceanography

Erin Oleson and Lisa Munger – NOAA PIFSC

Catherine Berchok – NOAA NWFSC

Danielle Cholweiak, Denise Risch, and Sofie Van Parijs – NOAA NEFSC

Melissa Soldevilla – NOAA SESFSC

## Table of Contents

1	Overview .....	6
2	Setup and Administration .....	6
2.1	Setup.....	6
2.1.1	Hardware and software requirements.....	6
2.1.2	Download Tethys.....	11
2.2	Start the server.....	14
2.2.1	Using a batch file.....	14
2.2.2	Using the command line .....	16
2.3	Shutdown the server .....	17
2.4	Run the server as a service.....	18
3	Using Tethys.....	19
3.1	Data organization in Tethys.....	20
3.1.1	Collections.....	20
3.1.2	Document structure .....	22
3.2	Adding data to Tethys.....	32
3.2.1	Source maps.....	33
3.2.2	Importing documents to Tethys.....	35
3.2.3	Updating existing documents .....	35
3.2.4	Viewing attached images in a web browser .....	35
3.3	Removing/modifying data .....	36
3.4	Query .....	36
3.4.1	Data Explorer.....	37
3.4.2	Web client .....	37
3.4.3	Java client.....	37
3.4.4	MATLAB client.....	40
3.4.5	Python client .....	42
3.4.6	XQuery.....	44
3.4.7	Case study: Northeast Fisheries Science Center Minke Boing analysis .....	50

4	Maintenance .....	53
4.1	Checkpoints .....	53
4.2	Backups.....	53
4.3	Help! My database has fallen and cannot get up! .....	54
4.3.1	Server will not start / Server window disappears .....	54
4.3.2	Database is not responsive .....	54
4.3.3	Database is corrupted .....	54
5	Appendix: XML Schema Diagrams .....	55
5.1	Calibration .....	55
5.2	Deployment .....	58
5.3	Detections.....	65
5.4	Ensemble .....	70
5.5	Event.....	71
5.6	Localization.....	72
6	Appendix – BatchLogs.....	76
7	Appendix: Tethys.xq Module Functions.....	77
8	References .....	79
9	Licenses.....	79
9.1	Python.....	80
9.2	Berkeley DBXML .....	81
9.3	CherryPy Object oriented web framework .....	101
9.4	Libraries using the MIT License .....	102

## List of figures

Figure 1 – Office Database Connectivity: List of available ODBC drivers appears on the Drivers tab. In this example, a driver for MySQL has been installed in addition to the Microsoft drivers that were added by installing Office. ....	8
Figure 2 - File tab in Microsoft Word. We see that this version is click-to-run (arrow 2) and we will need to install the Access ODBC driver. Clicking on About Word, it will show whether this is a 32- or 64-bit version (Figure 3). ....	9
Figure 3 - About Word dialog. Note arrow pointing to information indicating this installation of Office is 64-bit. ....	9
Figure 4 – Windows dialog for system properties. ....	10
Figure 5 – Windows dialog for environmental variables. ....	10
Figure 6 - Sample windows firewall dialog (Windows 7) requesting the user to allow Python to access the Tethys port. ....	15
Figure 7 – Command line interface once server is running. ....	15
Figure 8 – Installing Tethys as a service using Non-Sucking Service Manager (NSSM) ....	19
Figure 9 - The ERDDAP search web interface allows one to search for data by specifying multiple criteria. ....	29
Figure 10 - Row from an ERDDAP data search for sea surface temperature. ....	30
Figure 11 – ERDDAP Aqua Modis 8-day sea surface temperature composite. This dataset is accessed using four dimensions: time, altitude, latitude, and longitude. Positioning the cursor (mouse pointer) over any one of these will show the possible values and the resolution of the data. ....	31
Figure 15 – Detections document attachment viewed in a web browser.....	36
Figure 16 Detection submission for MATLAB client .....	<b>Error! Bookmark not defined.</b>
Figure 17 – Calibration schema for recording information about instrument calibration. Dark lines indicate required elements; light lines indicate optional elements. ....	56
Figure 18 – MetadataInfo is a required element in the Calibration schema to provide details on who is responsible for the calibration record. ....	57
Figure 19 – QualityAssurance is a required element in the Calibration schema to provide details on the quality of the calibration.....	57
Figure 20 – Process is an optional element in the Calibration schema to provide details of the calibration process used. ....	58
Figure 21 – ResponsibleParty is an optional element in the Calibration schema to provide details on who performed the calibration.....	58
Figure 22 – Deployment schema for recording information about instrument deployments. Dark lines indicate required elements; light lines indicate optional elements. ....	60
Figure 23 – The SamplingDetails element within the Deployment schema contain information about the recordings on each channel during the deployment.....	61
Figure 24 – The Data element within the Deployment schema contains information about the location of the data from the deployment. ....	62

Figure 25 – The DeploymentDetails element within the Deployment schema contains information about where and when a deployment occurred.....	63
Figure 26 – The Sensors element within the Deployment schema describes the types of sensors associated with a deployment. ....	64
Figure 27 – The optional QualityAssurance element within the Deployment schema allows for the inclusion of details about the quality assurance process. ....	65
Figure 28 – Detection schema for recording information about detections. Dark lines indicate required elements; light lines indicate optional elements. ....	66
Figure 29 – The Effort element within the Detection schema captures the timespan and types of events that were investigated. ....	67
Figure 30 – The OnEffort element within the Detection schema is where individual detections are recorded. ....	69
Figure 31 – The Description element with the Detection schema allows for details on the detection objectives and methods. ....	70
Figure 32 – Ensemble schema used to create logical groupings of instrument deployments. Dark lines indicate required elements; light lines indicate optional elements. ....	71
Figure 33 – The ZeroPosition element in the Ensemble schema describes a point that localizations can reference. ....	71
Figure 34 – Event schema used to record phenomena derived from other knowledge sources. Dark lines indicate required elements; light lines indicate optional elements. ....	72
Figure 35 – The Localization schema is used to record localizations of sources from multiple instruments. Dark lines indicate required elements; light lines indicate optional elements. ....	73
Figure 36 – The Effort element in the Localization schema includes information about when the localization occurred. ....	74
Figure 37 – The Localization element within the Localization schema provides details about individual localizations. ....	75
Figure 38 – The IntermediateData element within the Localization schema can be used to record information about the localizations. ....	76

# 1 Overview

Tethys is a temporal-spatial database for metadata related to acoustic recordings. The database is intended to house metadata from marine mammal detection and localization studies, allowing the user to perform meta-analyses or to aggregate data from many experimental efforts based on a common attribute. This resulting database can then be queried based on time, space, or any desired attribute, and the results can be integrated with external datasets such as NASA's Ocean Color, lunar illumination, etc. in a consistent manner. While Tethys is designed primarily for acoustic metadata from marine mammals, the design is general enough to permit use in other areas as well.

Tethys provides a scientific workbench to the practitioner. Consequently, rather than providing a stand-alone graphical interface, Tethys provides methods, or subroutines, that can be called from programming environments that practitioners use to conduct their analysis. Currently, Tethys supports MATLAB, Java, and Python. The R programming language will be included in the next major release. These methods allow practitioners to access the metadata associated with a specific laboratory or project. Additionally, the tools provide access to environmental data based on spatial location and selected temporal boundaries from a wide variety of online sources.

To run Tethys, a Windows machine is required (porting to other platforms is possible with a little work). To access Tethys from other machines, the network will need to permit communication between machines. In most cases, this will require a modification of a machine's firewall rules.

This manual is divided into several major parts: you need not read them all to use Tethys effectively. Section 2 contains information about setting up and administering Tethys, while section 3 provides information for practitioners who wish to use it. Users may wish to begin by reading about data organization (section 3.1) which describes the different types of documents that Tethys can store, and the section of the manual that is appropriate for the language that they will be using to conduct their queries. Queries can either be written in the XQuery language (section 3.4) or the user can invoke specialized functions that construct common queries. The richest set of common queries is available for MATLAB (section 3.4.4), which also has a separate "cookbook" style manual (MatlabCookbook) document.

## 2 Setup and Administration

### 2.1 Setup

#### 2.1.1 Hardware and software requirements

Tethys is designed to be executed on a Microsoft Windows platform with a 64-bit version of the Microsoft Windows operating system. The user will need a Windows machine to be used as the server<sup>1</sup> for creating and housing the database. The same machine can be used for querying data and using the associated Tethys methods or additional client machines can be used. It is recommended that there be

---

<sup>1</sup> In this context, "server" means that Tethys will be providing services to other machines. The Windows Server operating system is *not required*.



ample disk space for the database and that plans be put in place for routine backup of the database. As an example, in early 2023, the database used at the Scripps Whale Acoustics Lab contained over ten million detections and used a bit over 34 GB of storage. Most of the space was used for the database records themselves (24 GB). The remaining storage was used for archival records, sample audio/images stored at analyst request, and stored queries used to accelerate processing.

#### ***2.1.1.1 Windows firewall***

While not a Tethys utility, Windows firewall is a service that provides internet security. You may need to configure Windows firewall to permit network traffic to and from your machine. A tutorial article by [Hoffman](#) (2012) explains how to set up Windows firewall rules. The firewall can be configured to provide more selective filtering, such as only allowing access from specific machines or subnetworks. Modifying the firewall rules requires administrative privileges.

#### ***2.1.1.2 Microsoft ODBC installation for spreadsheet and database import***

To import non-XML information into Tethys, you need to have the 64-bit version of Microsoft's open database connectivity (ODBC) driver installed. This will allow imports from Microsoft family products (e.g., Access and Excel) and comma-separated value files. To support non-Microsoft databases, additional vendor-specific ODBC interfaces may need to be installed (e.g., the [MySQL driver](#) for MySQL databases). Note that you will need to ask your system administrator to install these drivers if you do not have administrative privileges.

By default, ODBC drivers for Excel, Access, and comma-separated values files are not present on Windows. If you have a 64-bit Microsoft Office 365 installed, you may have access. Some older versions of Microsoft Office 365 do not support ODBC by other programs such as the Tethys server. You can check if you have access by selecting the Start menu and searching for ODBC Data Sources (64 bit). Click on the Drivers tab. If you have access, you should see sources for Microsoft Access and Excel (Figure 1).

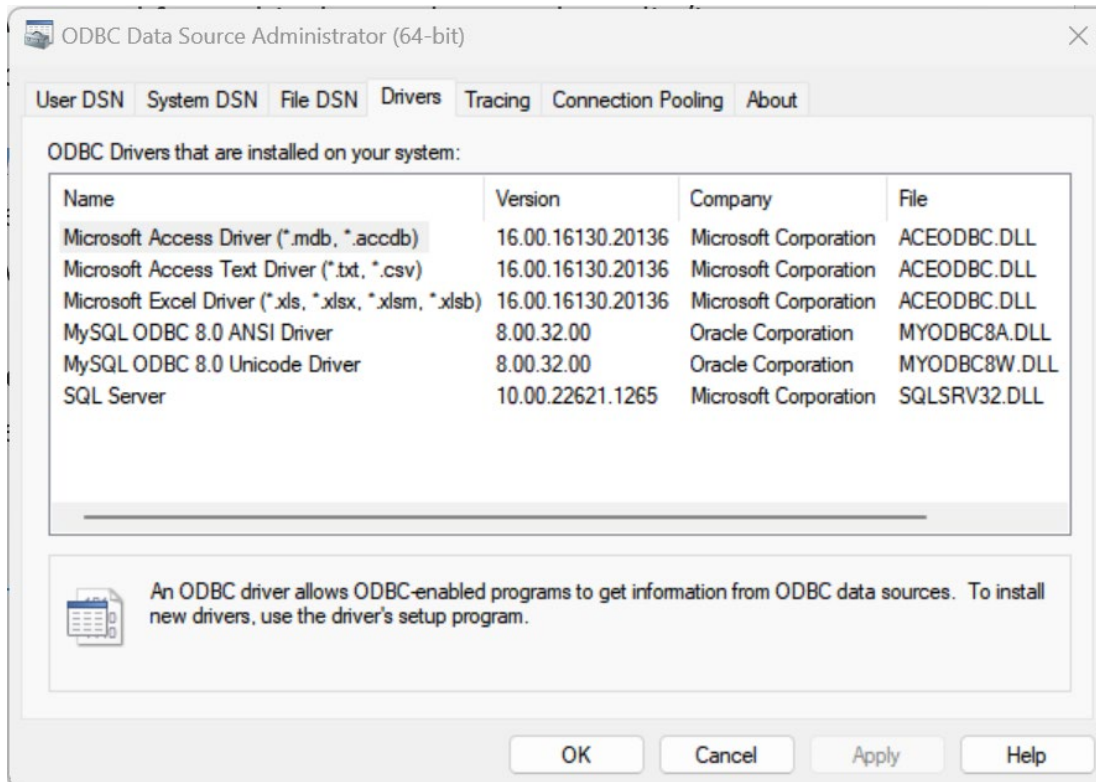


Figure 1 – Office Database Connectivity: List of available ODBC drivers appears on the Drivers tab. In this example, a driver for MySQL has been installed in addition to the Microsoft drivers that were added by installing Office.

If you do not have the Microsoft drivers, we recommend installing the freely available Microsoft Access Database Engine, version 2016 or later, which is currently available [here](#) (Microsoft changes links frequently, and you may need to search for it). See this [Microsoft knowledge base article](#) if you have any problems.

Note that the 64-bit ODBC driver cannot be run when 32-bit versions of Office are installed. To determine if a recent version of Office is 32- or 64-bit, open an office document (e.g., Word) then click on the File icon on the ribbon, resulting in the following after you click on Account (arrow 1 below):



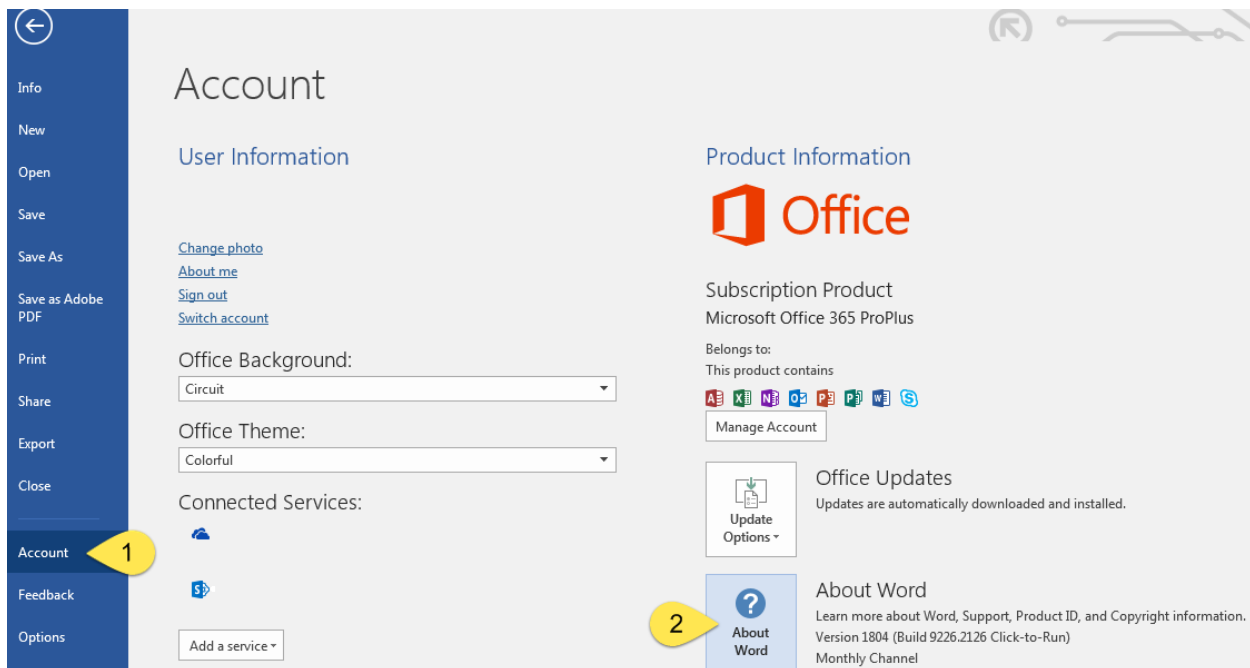


Figure 2 - File tab in Microsoft Word. We see that this version is click-to-run (arrow 2) and we will need to install the Access ODBC driver. Clicking on About Word, it will show whether this is a 32- or 64-bit version (Figure 3).



Figure 3 - About Word dialog. Note arrow pointing to information indicating this installation of Office is 64-bit.

### 2.1.1.3 R installation for data export (optional)

If you want to export query results from The Tethys web client (<http://your.server/Client>) in R format, download and install R (versions 4.2.1, 4.2.2, and 4.0 have been tested) on your server machine. The following link provides details on that installation. [Download R for Windows](#). **Note: For Tethys to automatically install required packages, you must have write permissions to the R library directory.** You can either install R locally in the account that will be running Tethys, change the R library directory to be writable, or install the following library packages manually: stringi, R6, methods, XML, data.tree.

After you have installed R you will need to set an environment variable (R\_HOME) to point to the location where R has been installed. To set environment variables, click on the Windows Start button and start typing View Advanced System Settings. Once selected, the System Properties window will open.

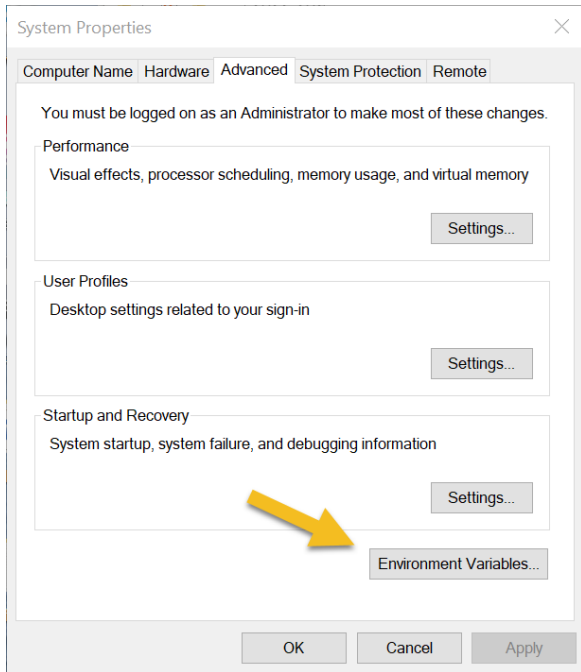


Figure 4 – Windows dialog for system properties.

Click on Environment Variables. A new window will show User and System variables. Add a new User variable named R\_HOME where the variable value is the path to the directory where R was installed.

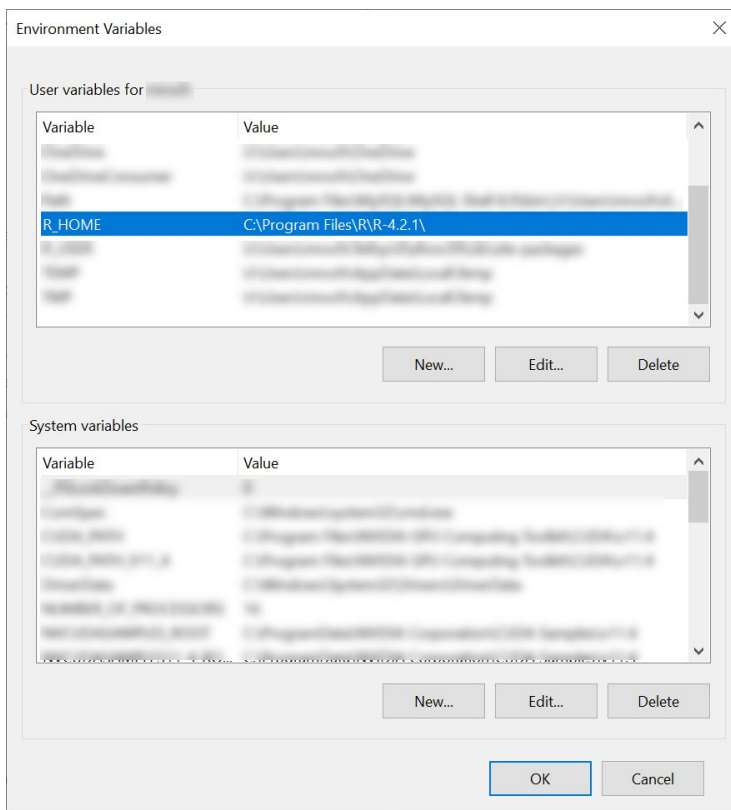


Figure 5 – Windows dialog for environmental variables.

Click OK and then click OK again to close the environmental variables window. The next time you start the Tethys Server, it should display a message indicating that R is starting.

### 2.1.2 Download Tethys

The Tethys software repository can be accessed on Google Drive. This repository contains ZIP files for the server, various clients, the database, and various utilities, which you can download as needed.

If you are downloading Tethys to get your database set up for the first time, it is recommended that you download all the ZIP files in the Tethys repository. This will allow you to run the server, as well as provide access to a sample database and an empty database ready for your data.

If you already have Tethys running on a different machine, you do not need to download everything in the repository. For example, if your machine is not storing the Tethys database, you do not need to run the server on your machine and therefore do not need to download *Server.zip*. In this case you may just want to download the file for the client you will be using (e.g., *MatlabClient.zip* or *PythonClient.zip*) and any associated files, as detailed below.

After identifying which ZIP files you need, simply download and unzip (right-click on the file and select unzip) to a folder of your choice. **IMPORTANT:** *Any folders you download must be placed in the same folder—the hierarchy of the folder structure must be maintained because some scripts and programs assume that resources are in specific folders.*

#### 2.1.2.1 Server

The files to run the server are contained in *Server.zip*. **This file must be downloaded on the machine that stores the Tethys database.** Starting the server implements a RESTful web service that is used by the client software. The server is implemented using two open-source technologies: the Python programming language and Oracle's Berkeley DBXML, which provides the extended markup language (XML) database engine.

**Requirements:** *Python39.zip* (downloaded and extracted)

#### 2.1.2.2 Clients

There are various clients that can be used to communicate with the server. There are two clients that can be used from a web browser: the Web client and the Data Explorer. There are also clients for making database queries using different programming interfaces: there are currently Java, Python, and MATLAB clients, and there are plans to include an R client in an upcoming release.

##### 2.1.2.2.1 Web client

The Web client provides a user-friendly interface for accessing and visualizing data in the Tethys database. The user can construct simple and advanced queries, save and refine queries, and export query results all from a web browser. The user can also visualize query results on maps and in weekly timeseries. This is included in *Server.zip* and does not require a separate download.

A guide to using the Web client can be found in the documentation (see *Tethys-WebClientManual*)

#### 2.1.2.2.2 Data Explorer

The Data Explorer (*DataExplorer.zip*) is an interactive tool for exploring your data using a point-and-click interface in a web browser.

**Requirement:** *Python39.zip* (downloaded and extracted)

A guide to using the Data Explorer can be found in the documentation (see *Tethys-DataExplorerManual*)

#### 2.1.2.2.3 Java client

The Java client (*JavaClient.zip*) contains Java libraries that can be used by MATLAB, R, and of course Java. The Java client will work with any language that has a Java interface. This is usually used behind the scenes (e.g., the MATLAB client relies on the Java client)

More detail on the Java client can be found in Section 3, Using Tethys (see [3.6 Java client](#))

#### 2.1.2.2.4 Python client

The Python client (*PythonClient.zip*) contains code for managing the database (e.g., graceful shutdown, checkpoints, etc.) as well as accessing data.

**Requirement:** *Python39.zip* (downloaded and extracted)

More detail on the Python client can be found in Section 3, Using Tethys (see [3.8 Python client](#))

#### 2.1.2.2.5 MATLAB client

The MATLAB client (*MatlabClient.zip*) contains code for accessing the database (in the *db* folder) and visualizing data (in the *vis* folder).

**Requirements:** MATLAB 2018b or later & the Java client

More detail on the MATLAB client can be found in Section 3, Using Tethys (see [3.7 MATLAB client](#)) and extensive documentation on making database queries from MATLAB can be found in the documentation folder (see *Tethys-MATLABCookbook*).

### 2.1.2.3 Databases

The actual database is stored in a *databases* folder (*databases.zip*). **This file must be downloaded on the machine that stores your Tethys database.** When first downloaded, the *databases* folder will contain two subfolders: *metadata* contains a new instance of a Tethys database without any deployments, detection, or localizations (i.e., this is an empty database) while *demodb* contains a demonstration Tethys database that is prepopulated with data from varying sources (e.g., SIO, SDSU, PIFSC, and SWFSC).

Table 1 – Contents of each database folder within the *databases* directory.

Folder/file name	Description
<b>db</b>	Contains files used by the Berkely DB XML database.
<b>DeletedArchive</b>	A copy of source documents that have been deleted from the repository. Note that multiple versions are not currently maintained.
<b>lib</b>	XQuery library modules and XML schema.
<b>logs</b>	Logs detailing server activity.
<b>ResultXML</b>	
<b>source-docs</b>	A copy of the source documents added to Tethys are stored in this directory. This is useful should there ever be a catastrophic failure and the database needs to be reconstructed from source material.
<b>shutdown.bat</b>	A batch file to shutdown the unencrypted server. This file may be edited to add --port option should you wish to shutdown a server that is not running on the default port of 9779.
<b>TemporaryFiles</b>	Workspace
<b>tethys.bat</b>	A batch file that will launch the Tethys server with the default options. Communication is over an unencrypted network connection.
<b>tethys-ssl.bat</b>	A batch file to launch Tethys with secure socket layer encryption enabled. You must obtain a certificate and a public/private key pair before you can start Tethys in this mode.

#### 2.1.2.4 Utilities

##### 2.1.2.4.1 Documentation

A collection of documentation (*Documentation.zip*) that provides details on the Tethys server and clients.

##### 2.1.2.4.2 NilusXMLGenerator

The NilusXMLGenerator (*NilusXMLGenerator.zip*) is a Java library for converting detection, classification, and localization data into a format that can be imported into Tethys. Tethys has an import tool that can process CSV, spreadsheets, and databases without requiring any programming, but the NilusXMLGenerator can be useful for having detectors generate output directly or for handling special case data. This will work with any language that has a Java interface.

For details on how to implement this, please read the provided documentation (see *Tethys-NilusXMLGenerator*).

##### 2.1.2.4.3 Python39

This is version 3.9 of the Python Software Foundation’s implementation of the Python language (*Python39.zip*). Many add-on packages have been installed. If you wish to see the list of packages, start python with the arguments “-m pip freeze”.

## 2.2 Start the server

When the server is started it opens a port on your computer (essentially a mailbox that lets other programs communicate with the server). You may see a firewall prompt asking for permission to use the port (see below). By default, the server runs on port 9779 but this can be reconfigured. Web traffic is usually on port 80 for unencrypted traffic, and on port 443 for encrypted traffic. We use an alternative default port to avoid conflicting with any web services that your machine may be running.

### 2.2.1 Using a batch file

The easiest way to start the server is to navigate to the directory where the database that you wish to serve is located. Let's assume that all your Tethys-associated files are in a folder called *Tethys* and you want to use the demo database. You would navigate to *.../Tethys/databases/demodb*.

The batch file, *tethys.bat*, contains the directives needed to start the server using the data located in the *demodb* directory. Double-click on *tethys.bat*.

**Note 1:** *tethys.bat* will start the server using the http: web service protocol and *tethys\_ssl.bat* will start the server using secure socket layer (SSL), the protocol used by financial institutions to encrypt data. To use SSL, your machine must have a certificate issued by a certificate serving agency. Starting the server without a certificate will fail and using self-signed (self-generated) certificates can cause client software to fail with errors indicating that the certificate is not trusted. Directions for setting this up are in the documentation (*Tethys-SecureSocketLayer*) and on the Tethys web site: [tethys.sdsu.edu](http://tethys.sdsu.edu).

**Note 2:** Some IT departments may set policies that prohibit starting batch files by double-clicking. If your administrator's security policy does not allow you to click on batch files, open a command window (Windows Key + R and type `cmd.exe`). You'll need to change the directory to your folder containing your database. For example, if Tethys was downloaded in `C:\Users\UserName\Tethys` you would type:

```
cd C:\Users\UserName\Tethys\databases\demodb
```

followed by:

```
tethys.bat
```

The first time that you run this, Windows is likely to ask you if you wish to allow the Python interpreter to access the data port that Tethys uses to communicate between the server and its clients. The dialog will likely look something like this:



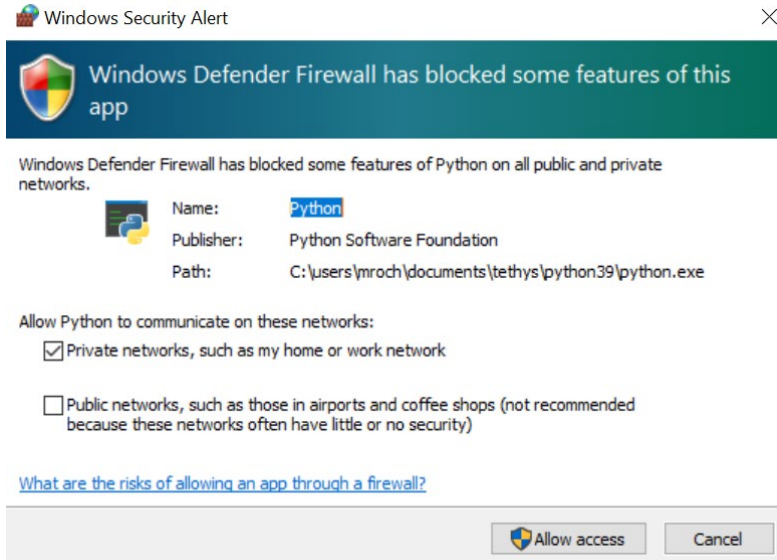


Figure 6 - Sample windows firewall dialog (Windows 7) requesting the user to allow Python to access the Tethys port.

Click to allow access. You may need authorization from your information technology team for this. **Note:** this software was developed by employees of San Diego State University and Scripps Institution of Oceanography. We rely on well-respected open-source software, such as that provided by the Python Software Foundation which provides the Python interpreter and manages repositories of open-source scientific software.

After starting the batch file, a command window should open and you should see the following:

```
C:\WINDOWS\system32\cmd.exe
any batch scripts that start Tethys. dbxmlServer.py will be
removed in a future version of Tethys.
[01/Mar/2023:10:59:42] Welcome to Tethys, 3.0 beta 0 - Server starting...
No server parameter file (D:\Tethys\Tethys\ServerDefault.xml) configured, using internal defaults

Examining logs in D:\Tethys\Tethys\databases\demodb\db to verify that database is correct.
This may require a long time (e.g. 1 h) when the logs have not
been verified recently or a large changes have been made to the database
Log processing started at 2023-03-01 10:59:42.041119
Cache size set to 1.00 GB
Log processing complete. started at: 2023-03-01 10:59:42.041119 elapsed 0 days 00:00:00.482360
Checkpointing database D:\Tethys\Tethys\databases\demodb\db... checkpoint complete
Cache size set to 1.00 GB
BSddb environment initialized
Starting DB XML in transactional mode
R[write to console]:
Attaching package: 'XML'

R[write to console]: The following object is masked from 'package:tools':

  toHTML

[01/Mar/2023:10:59:50] ENGINE Bus STARTING
[01/Mar/2023:10:59:50] ENGINE Started monitor thread 'Autoreloader'.
[01/Mar/2023:10:59:50] ENGINE Serving on http://0.0.0.0:9779
[01/Mar/2023:10:59:50] ENGINE Bus STARTED
[01/Mar/2023:10:59:50] Web interface at http://localhost:9779/Client
```

Figure 7 – Command line interface once server is running.

The server should now be running. You can verify this by visiting <http://localhost:9779/Client/> on the same machine as the server. You should see an empty map appear. If you click on the **Deployment**

button under “Submit or Refine Queries”, you should see markers for the deployments from your database appear on the map along with details for each deployment below the map. This is the Web Client, which is described in more detail in section 3.4.2 and in the documentation (see *Tethys-WebClientManual*).

### 2.2.2 Using the command line

The server can also be started by having Python execute *tethys.py* from the command line.

Throughout this section, *server\_path* will refer to the path of the *Server\src* folder that contains *tethys.py*, *python\_path* will refer to the path to the *Python39* folder that contains *python.exe*, and *database\_path* will refer to the path to the folder within *databases* that contains the database you would like to serve (e.g., *...databases\demodb* if you want to start the server using the demo database).

Note that you can also set environmental variables that point to these paths instead of having to include the full path yourself. For an example of this, see the *tethys.bat* file.

To start the server, open the command line and change the directory to *server\_path*:

```
C:> cd server_path
```

and type:

```
server_path> python_path\python.exe tethys.py -r database_path
```

You should see something similar to Figure 7 appear in the command window. The server should now be running. You can verify this by visiting <http://localhost:9779/Client/> on the same machine as the server. You should see an empty map appear. If you click on the **Deployment** button under “Submit or Refine Queries”, you should see markers for the deployments from your database appear on the map along with details for each deployment below the map. This is the Web Client, which is described in more detail in section 3.4.2 and in the documentation (see *Tethys-WebClientManual*).

#### 2.2.2.1 Optional arguments

To start the server, we used *-r* to change the resource directory to point to the database we wanted to serve. To see all the various options, add a *--help* flag to the end of the statement we used previously:

```
server_path> python_path\python.exe tethys.py -r database_path --help
```

You should see something like the following:

```
server_path> python_path\python.exe tethys.py database_path --help
Welcome to Tethys - Server starting...
Usage: tethys.py - XML Database Server
       Default values for choices are marked by an *

Optional arguments:
  -h, --help                show this help message and exit
  -s SECURE_SOCKET_LAYER, --secure-socket-layer=SECURE_SOCKET_LAYER
                             Use encrypted communication (true/false*)?
                             encrypted-->https:// unencrypted-->http://
  --port=PORT                port to run on (default=9779)
```

```
-t TRANSACTIONAL, --transactional=TRANSACTIONAL
    Use transaction processing (true*/false)?
-d DATABASE, --database=DATABASE
    Directory (folder) name where the XML database will be
    stored (must exist). Most users wishing to specify -d
    should probably use the -r switch instead.
-r RESOURCEDIR, --resourcedir=RESOURCEDIR
    Set Tethys's resource directory (folder). This is the
    parent directory for all data used by Tethys including
    the XML database.

--recovery RECOVERY  Open the database temporarily in recovery mode. If successful, recovers,
                    closes and reopens normally
```

Each option has a long name that is preceded by two dashes, and sometimes a short name, which is preceded by a single dash. Either one may be used.

Setting `secure_socket_layer` to true enables encrypted transmission. It requires the generation of certificates and keys. While the secure socket layer is currently functioning, we will focus on unencrypted communication as it is much simpler.

Computers communicate across networks by specifying an address and a port. The address is the Internet protocol (IP) address of the computer running the server and is not settable. The port can be thought of as a service address at the computer. By default, Tethys uses port 9779, but this can be overridden.

Many databases are capable of performing operations “atomically.” This means that an operation is either not performed or is completed but will never fail in a partially executed way. Should a failure occur part way through an operation (e.g., a power failure), a log is used to either undo the operation or complete it. This is known as `transactional` processing and is enabled by default in Tethys. We do not recommend running the database with transactional processing set to false as unexpected events such as power failures can lead to database corruption.

The name of the database can be overridden using the `database` option. When this flag is used, the folder will be relative to the resource directory unless it contains a path separator (e.g., `--database %USERPROFILE%/Documents/testbed`), which would use the `testbed` folder in the current user’s `Documents` folder.

Files for the database are by default stored in `C:/Users/Tethys/metadata`, but this can be overridden with the `resourcedir` option.

## 2.3 Shutdown the server

The proper way to shut down the server is to issue a shutdown command. There is a `shutdown.bat` script in the same directory as the `tethys.bat` script that will do this for you.

**The `shutdown.bat` file relies on the `shutdown.py` file included in the Python client. You must download `PythonClient.zip` and extract the Python client for this to work.**

You can also use the command line to issue a shutdown. Open a command window and navigate to `...Tethys\PythonClient\src`. Then type:

```
python_path\python.exe shutdown.py -server ServerName -port PortNumber
```

where `python_path` is the path to the `Python39` folder, `ServerName` is replaced with the name of the computer running the server (e.g., `localhost` if it is on the same machine) and `PortNumber` is the port on which the server was started. If you did not change the default port, this can be omitted.

Alternatively, you can send a terminate command (CTRL+Break, on most keyboards, the break key is in the row of function keys) and the server will shut down and the command prompt will close. Any in-progress requests will be interrupted, and in-progress attempts to add data may need to be repeated, but the database will not be corrupted.

## 2.4 Run the server as a service

The server can also be run as an operating system service, although this requires the download of additional software. This means that the server will start automatically and will restart if the server process unexpectedly dies or the server machine is restarted. We recommend using the Non-Sucking Service Manager (NSSM) developed by Iain Patterson. Source code and executable files can be downloaded from [nssm.cc/download](http://nssm.cc/download). As of this writing, most users should use version 2.24-101 or newer.

Complete details on NSSM can be found in the NSSM documentation, but installation of the service can be accomplished by opening a command window, **changing the directory to where you uncompressed the nssm download**, and typing:

```
NSSM-Path> win64\Nssm install
```

This will open a dialog that has multiple tabs (Figure 8) that will need to be populated. For the discussion below, we will assume that Tethys has been downloaded to `C:\Users\myacct\Tethys`: you will need to update all paths according to where Tethys has been installed. The **Application** tab requires a path to the Python executable. This will be in `C:\Users\myaccount\Tethys\Python39\python.exe`. The startup directory is where the server program is located: `C:\Users\myaccount\Tethys\Server\src`. Arguments must contain the name of the Tethys server program, an argument that specifies which database should be served, and any additional arguments. As an example, if we wished to serve the demonstration database on the standard web port (80), we would use the following arguments:

```
-r U:\Users\myaccount\Tethys\Databases\demodb --port 80
```

The Service name can be anything you wish, we recommend “Tethys.”

The **Details** tab simply shows how the Tethys service will appear when listed in the Windows services interface. Display name is the name that will appear in the list and description allows people looking at services to know the purpose they serve. We recommend using “Tethys Server” for the display name and “Tethys Acoustic Metadata Service” for the description. Startup type allows you to determine if the

service is started automatically when the machine starts or not. If you choose to start it automatically, we recommend using the delayed start.

The final tab that you need to populate is the **Log on as** tab. Select **this account** and enter the name and password under which the service should be run (this can either be your account or an account that you have created for Tethys, but it should have write permission to the database files). You will need to type the password twice and then press **install service**. If all goes well, you will have a message that the service has been installed. If you selected an automated startup type, the service will start the next time the machine restarts. You can start the service manually from the services manager, from nssm: nssm start Tethys.

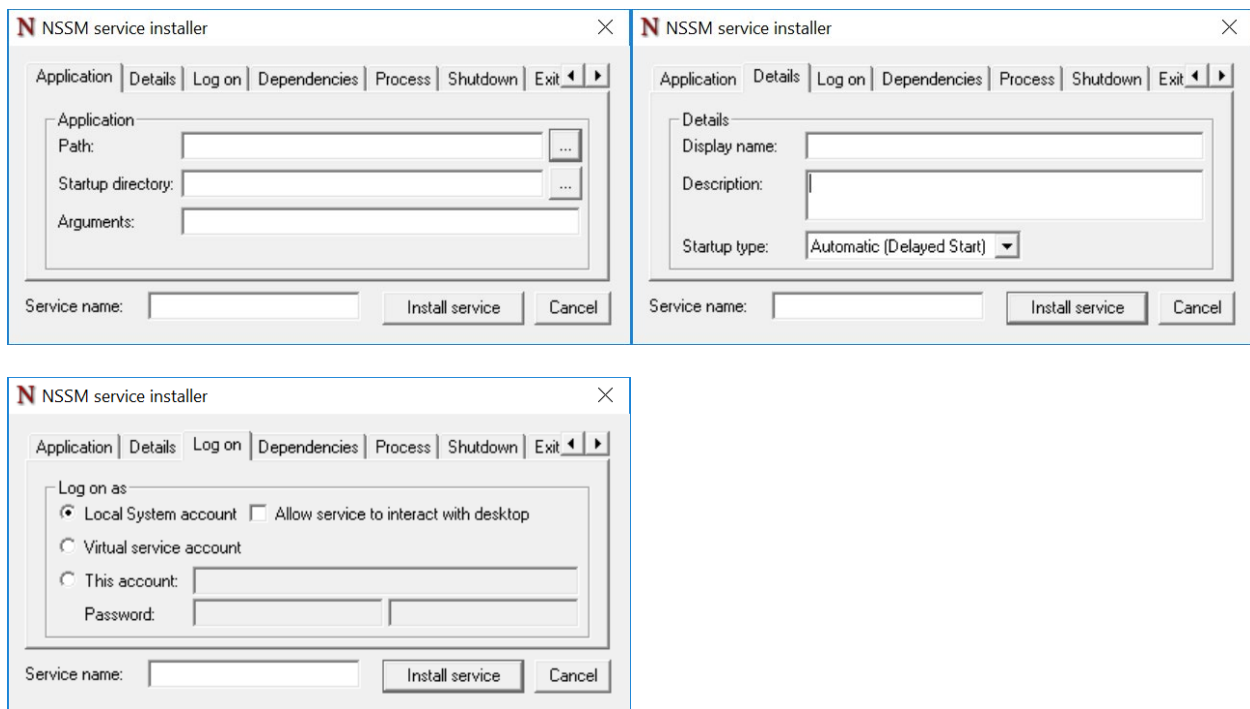


Figure 8 – Installing Tethys as a service using Non-Sucking Service Manager (NSSM)

### 3 Using Tethys

At this point, it is assumed that you have an operational Tethys database. If the server is set up on a different machine than the one you are using, your database administrator should be able to tell you the name and port of the machine where Tethys is running.

In this section, we will describe how data is organized in Tethys, how to add and remove records from your database, and how to access the records stored in your database (i.e., how to query the database).

Tethys itself is written in Python and it uses the XQuery language to access records stored in your database. However, you do not need to know Python or XQuery to use Tethys successfully and complete the abovementioned tasks. This is where the clients (i.e., Web client, Data Explorer, MATLAB, Python,

and Java) come into play, by providing a way for users to interact with their database in a way that is most comfortable for them.

For querying Tethys, the Web client and the Data Explorer can be used to construct queries without any programming knowledge. For those with some programming knowledge, many templates for common queries have already been predefined and can be accessed using function calls from one of the other clients. Currently, MATLAB has the richest set of queries. If you need to construct queries that are more advanced than the predefined options, it will be helpful to know XQuery (see section 3.4.6).

### 3.1 Data organization in Tethys

Regardless of whether you use predefined queries or write your own, it is helpful to understand the structure behind how your data is stored in Tethys.

At a high level, you should understand that within Tethys, data are organized into different categories called collections. For example, there is one collection that stores data about detections and a different collection that stores data about instrument calibrations. Even though we may at some point want to know how the instrument that detected certain calls was calibrated (and we can figure this out), these are very different types of information and are categorized differently by Tethys.

When you add data to Tethys, it will be as some sort of document (e.g., an excel spreadsheet where an analyst has noted the start and end times of calls). You will need to (1) specify which collection your document is meant for and (2) ensure that your document conforms to the document structure required for the collection you want to submit it to.

Note that, where possible, we use concepts from ISO 19115 or OpenGIS SensorML<sup>2</sup>, but our emphasis is on meeting the needs of the marine mammal community in the most user-friendly way possible. Consequently, we deviate from these standards. In addition, there are many concepts that are not covered in these standards such as recording detection effort.

#### 3.1.1 Collections

While there are a number of collections in Tethys, the six that we will discuss here are Calibrations, Deployments, Detections, Ensemble, Events, and Localizations.

- The **Calibrations** collection contains information about the calibration of individual instruments, hydrophones, or preamplifiers. Due to shared elements with the Deployments collection, it is possible to determine calibration details for an instrument used in a specific deployment (e.g., date, method, responsible party, and measurements).
- The **Deployments** collection is used to represent information about the deployment of instruments used to collect the data analyzed for detection and localization. It contains information such as the number of channels, sample rate, duty cycle, etc.

---

<sup>2</sup> [http://www.iso.org/iso/catalogue\\_detail.htm?csnumber=26020](http://www.iso.org/iso/catalogue_detail.htm?csnumber=26020) and <http://www.opengeospatial.org/standards/sensorml> respectively



- The **Detections** collection describes when events have been detected within a specific deployment and can be of varying scale. An example of a fine scale detection might be reporting individual echolocation clicks produced by a Risso's dolphin while a medium scale detection might indicate that there was an acoustic encounter of Risso's dolphins between some start and end time. Finally, one can report binned presence/absence (e.g., hourly) information. In addition to the detection events, attachments can be added. These attachments include audio files and images related to the detection events. Note that a maximum of 500 files can be attached to a given detection document (this limitation will be removed in a future release).
- The **Ensemble** collection allows multiple instruments to be referenced as one, which is useful when performing beamforming or localization on a large aperture array that contains separate instruments.
- The **Events** collection is used to specify events that may be of interest in the analysis. One such example might be a planned activity with possible consequences such as oil exploration. Individual detections of anthropogenic events such as airguns would be recorded in the Detections collection, but the knowledge that oil exploration was being conducted over a given time and location could be denoted in the Events collection.
- The **Localizations** collection denotes the source location of a sound source using either relative or absolute coordinates and permits the user to reference a detection in the Detections collection if appropriate.

When adding data to your database, you may only need to add a document to one collection. For example, if you wanted to record a new instrument deployment, you would add one document to the deployment collection. If you wanted to record detections that occurred at a site for which a deployment document already exists, you would add one document to the detections collection. However, sometimes you need to add documents for multiple collections. For example, if you detected calls at a new location you would need to add a document to the detections collection *and* add a document to the deployment collection.

For each collection, there is a schema that defines what data can be placed in the collection. You can think of a schema as a set of rules defining what data can be included and how it should be structured. This means that a document must match the schema of the collection it is added to. In some cases, the schema is well defined and values will be required for certain fields (e.g., if you are submitting a detections document you would be required to denote information about what you detected and where). Other times, the schema is looser and allows you to define new types of information that you want to include in the database. This is why a detection document and a calibration document are stored in different collections—while we would want to require information about what was detected and where for a detection document, this would not make sense for a calibration document.

### 3.1.2 Document structure

The various types of documents that the Tethys schema can support are described at a high level in this section. Although many of the document types correspond to the collections described previously, there are a few additional document types included in this section. The goal is to describe the types of information contained in the documents and the required elements, rather than every last detail. More detailed descriptions can be found in Appendix 5 which describes the full schema that structures each document type.

When you add a document to Tethys it is converted into extensible markup language (XML), which is a language used for structuring data. The core of XML is quite simple: data is contained within elements that provide structure. The start of each element is denoted by its name enclosed in < > and the end by the element name within </ >. A small XML fragment from a Deployment document is shown below:

```
<Deployment>
  <Id>SOCAL32-A</Id>
  <Project>SOCAL</Project>
  <DeploymentId>32</DeploymentId>
  <Platform> Mooring </Platform>
  other entries...
</Deployment>
```

Note that units are standardized when recorded in the database. Date and time follow the ISO 8601:2004 standard and are recorded in universal coordinated time (UTC). As an example, January 30<sup>th</sup>, at 6:22:30 PM UTC would be written 2013-01-30T18:22:30Z. Latitudes and longitudes are recorded in decimal form from 90 degrees (N) to -90 degrees (S) and 0 to 360 degrees (E).

Call types are not standardized in Tethys. While we offer recommendations in the supplemental material of Roch et al. (2016), there is not a consensus in the scientific community for all call names, and thus we do not enforce a standard.

#### 3.1.2.1 Calibration documents

Calibration documents record information about the calibration of an instrument. Information such as when a calibration occurred, who was responsible, the methods used, as well as the actual frequency responses are described here.

Required elements in a calibration document include **Id** (identifier for the instrument being calibrated), **TimeStamp** (date and time the calibration occurred), **Type** (the type of calibration, e.g., hydrophone), **QualityAssurance** (details on the data quality), **IntensityReference\_μPa** (a reference intensity for the measurements), **FrequencyResponse** (a list of frequencies and responses), and **MetadataInfo** (details on who is responsible for the calibration metadata).

#### 3.1.2.2 Deployment documents

Deployment documents record information about the deployment of an instrument. As many instrument designers have existing databases for their instruments, the goal of deployment documents is to provide enough information to access an instrument database and then to describe how the instrument was used. Information such as how and where an instrument was deployed, references to

tracklines for moving platforms, sensor packages, and configuration are all described here. While the emphasis is on acoustic data (e.g., sampling rates, duty cycles, quantization), the schema permits the description of arbitrary instrumentation.

Required elements in a deployment document include **Id** (a unique identifier for the deployment), **Project** (the name of the project the deployment is associated with), **DeploymentId** (numeric identifier for the deployment), **Platform** (the platform on which the instrument was deployed, e.g., mooring), **Instrument** (the instrument type and identifier), **SamplingDetails** (information about the recordings on each channel, e.g., sample rate), **Data** (data the instrument collected; typically a URI for audio data), **DeploymentDetails** (information about the deployment, such as location and time), and **Sensors** (the sensors on the instrument).

### **3.1.2.3** *Detection documents*

Detection documents record information about the process used to perform the detections, the source data, and the effort, which indicates which species and calls were searched for in the detection process. Recording effort is essential as a lack of detections for a specific species/call type is not relevant unless one was actually looking for them.

Required elements in a detection document include **Id** (a unique identifier for the document), **DataSource** (acoustic data identifier), **Algorithm** (details on the detection method), **UserId** (identification of the user that submitted the document), **Effort** (span and scope of detection effort), **OnEffort** (information about individual detections).

### **3.1.2.4** *Ensemble documents*

Ensemble documents provide a means of logically grouping instrument deployments. This is most useful for large aperture localization where separate instruments may be deployed individually, but the data within them are to be treated as if they originated from a single instrument.

Required elements in an ensemble document include **Id** (a unique identifier for the ensemble) and **Unit** (identifies an instrument within an ensemble and associates it with a deployment).

### **3.1.2.5** *Event documents*

Event documents denote phenomena or events that are derived from other knowledge sources. Examples include planned Naval exercises, whale watching cruises, pile driving, oil exploration, earthquakes, etc. Information about the type of event and the time period during which it took place are described here.

Required elements in an event document include **Name** (the name of the event), **Description** (information about the type of event and responsible party), **Start** (event start time), and **End** (event end time).

### **3.1.2.6** *Localization documents*

Localization documents provide location information in the form of absolute locations or bearings. Localizations can be derived from data sources consisting of a single instrument deployment or an ensemble of multiple instrument deployments.

The localization data itself consists of either a bearing or a location. Bearings consist of a horizontal and optional vertical angle, specified in degrees. Locations are x, y, and optional z distances, specified in meters relative to the zero location. For either type of localization, standard error may be specified in the same units. Finally, when a location is a result of several crossed bearings, an optional list of bearings (event identifiers from previous localizations) may be provided to link the position to the bearings used to produce it.

Required elements in a localization document include **Id** (unique identifier for the document), **DataSource** (the deployment or ensemble from which localizations were obtained), **Algorithm** (description of detection algorithm), **ResponsibleParty** (person or organization that generated the metadata), **UserId** (user that submitted the document), **Effort** (type of localization and when it occurred), and **Localizations** (information about individual localizations).

### 3.1.2.7 ITIS document

An essential element for being able to conduct meta-studies is to use consistent naming conventions. To that end, we have adopted the integrated taxonomic information system (ITIS, [www.itis.gov](http://www.itis.gov)) for describing species.

The ITIS collection contains one document, which is a snapshot of ITIS. Each entry has a taxonomic serial number (TSN), completename (scientific name), and vernacular entries.

Tethys's XML representation of ITIS supports physical phenomena by defining them with negative taxonomic serial numbers. Currently, there is a single TSN entry, Other, for physical phenomena. Due to the structure of the Tethys schema, the other category is represented as a Kingdom which is of course incorrect. As the taxonomic serial numbers (TSNs) used by ITIS are not user-friendly (e.g., 180514 is the TSN for Stejneger's beaked whale, *Mesoplodon stejnegeri*), library functions permit the translation between TSNs and common or scientific names.

### 3.1.2.8 ITIS ranks document

The ITIS ranks document contains a non-hierarchical version of the ITIS document. In ITIS, there are some Latin taxonomic names that are duplicated in different hierarchies. For example, *Hoplophthiracarus spiniformis* is a species that occurs in multiple subgenera. In the ITIS ranks document, only the first occurrence is used. A list of duplicate entries that are not included in the ITIS ranks document can be found in 'Notes on provided source documents.docx' located in *databases/yourdatabase/source-docs*.

### 3.1.2.9 Species Abbreviation documents

Detections are attributed to entries in the ITIS collection and are stored in Tethys as TSNs, which are not conducive to human interpretation. Although the ITIS collection can map to vernacular names in several languages, as well as scientific names for each species, many research organizations have their own set of names or abbreviations that they use to refer to species. For example, at the Scripps Whale Acoustics Lab, it is common to use abbreviations based on the genus and species names, such as Zc to denote *Ziphius cavirostris*, or Cuvier's beaked whale.

Unfortunately, the California sea lion, *Zalophus californianus*, would also be abbreviated Zc using this system, and consequently, it is very difficult to develop an abbreviation/local name list that would work for all groups. The Species Abbreviations collection provides a method for labs to use their own set of local names or abbreviations.

A species abbreviation document provides mappings between a local name or abbreviation and the Latin species name. Here's an example of how this would look if you wanted to use the abbreviation system used at the Scripps Whale Acoustics Lab:

```
<Abbreviations>
<Name>SIO.SWAL.v1</Name>
<Map>
  <completename>Megaptera novaengliae</completename>
  <coding>Mn</coding>
</Map>
<Map>
  <completename>Balaenoptera acutorostrata</completename>
  <coding>Ba</coding>
</Map>
...
</Abbreviations>
```

Where we have a <Name> element for the name of our species abbreviation document, and multiple <map> elements where you can provide the species Latin name for the <completename> element and the abbreviation you would like to use for that species for the <coding> element.

### 3.1.2.10 External document types

Tethys provides the ability to access external data sources which are represented as collections prefixed with the name *ext:* followed by a collection name. Tethys provides what is known as a mediation service, providing a consistent way of accessing these external data sources. These are returned as XML documents that can be manipulated like any other data that Tethys returns.

Data access is constructed in a manner that looks similar to XML document navigation. The user specifies the collection they want, followed by a set of slash (/) separated parameters and terminated with an exclamation point:

```
collection("ext:MediatorServiceName")/parameter1/.../parameterN!
```

Currently, mediation is provided for several types of external collection types described in the following sections.

When Tethys accesses an external collection, it caches the results for approximately seven days (default) on the Tethys server. Consequently, if the same data is requested multiple times, subsequent queries are faster. This is particularly helpful when developing routines to analyze data where the same query may be executed dozens of times as the analysis routine is written. The cached results are stored in *yourdatabase/db/mediator\_cache*. In rare cases, one may wish to disable this behavior. Examples of this include services that provide different results for the same parameters (e.g., report current conditions) and when one knows that a service has been recently corrected.

There are two ways to ensure that mediated services retrieve values directly from the Internet. The first is simply to add a colon followed by `cacheupdate` after the mediation service name, e.g.,

```
collection("ext:MediatorServiceName:cacheupdate")/parameter1/.../parameterN!
```

Data are retrieved from the mediator service and the mediator cache will be updated with the new results. A second and more drastic way to clear the cache is to empty the `mediator_cache` using one of the client programs such as the Python client's `clear_documents.py`. Both methods will work but emptying the `mediator_cache` clears the cache of all documents for every user, so the `cacheupdate` parameter is usually the preferred mechanism for ensuring a fresh copy of the data.

### 3.1.2.10.1 Ephemeris data

Ephemeris, information about astronomical objects, can be obtained through an interface to NASA JPL's [Horizons](#) Web Service (Giorgini et al., 1996). While NASA's system provides information on a variety of astronomical objects, the mediator interface has been primarily tested for solar and lunar information.

The Horizons service is accessed via the `ext:horizons` collection. An example query might look like this:

```
collection("ext:horizons")/target="sol"/latitude=32.8/longitude=243.8/start="2009-10-01T00:00:08:00"/stop="2009-10-04T00:00-08:00"/interval="5m tvh"!
```

The arguments may appear in any order and are as follows:

- **target** – Celestial body. The horizons mediator knows the names “sol” and “sun” for the sun, “moon” and “luna” for the moon.
- **latitude** and **longitude** – Position of the location in degrees for which the ephemerides are to be computed. Latitude must be in the interval between 90° (N) and -90° (S), and longitude must be between 0 and 360° E. Note that this is the format in which Tethys stores latitude and longitude, so no conversion is needed.
- **start** and **stop** – Time in ISO8601 format (YYYY-MM-DDTHH:MM:SS). All times are assumed to be in universal coordinate time (UTC), which is the Tethys default.
- **interval** – How often the ephemeris should be computed. When determining transit (rise/set), this should be in intervals of 5 m with the `tvh` flags set as in the example above.

Tethys will return XML describing the ephemerides. The document consists of a set of entries describing information about the celestial object in question. The result of the query above with manually added comments is:

```
<?xml version="1.0" encoding="utf-8"?>
<ephemeris>
  <entry>
    <date>2009-10-01 01:35:00</date>
    <sun type="civil">day</sun>    <!-- Civil sunset -->
    <moon>set</moon>
  </entry>
  <entry>
    <date>2009-10-01 13:44:00</date>
    <sun>day</sun>
    <moon>rise</moon>
```



```

</entry>
<entry>
  <date>2009-10-01 19:38:00</date>
  <sun>day</sun>
  <moon>transit</moon>
</entry>
<entry>
  <date>2009-10-02 01:32:00</date>
  <sun type="civil">night</sun> <!-- Civil sunrise -->
  <moon>set</moon>
</entry>
<entry>
  <date>2009-10-02 13:41:00</date>
  <sun>day</sun>
  <moon>rise</moon>
</entry>
<entry>
  <date>2009-10-02 19:35:00</date>
  <sun>day</sun>
  <moon>transit</moon>
</entry>
<entry>
  <date>2009-10-03 01:29:00</date>
  <sun type="civil">night</sun>
  <moon>set</moon>
</entry>
<entry>
  <date>2009-10-03 13:43:00</date>
  <sun>day</sun>
  <moon>rise</moon>
</entry>
<entry>
  <date>2009-10-03 19:37:00</date>
  <sun>day</sun>
  <moon>transit</moon>
</entry>
</ephemeris>

```

Note that, in most cases, information about the moon is returned as well.

### 3.1.2.10.2 Time zone data

The time zone collection can provide the time zone for a specific longitude and latitude based on nautical time zones that consist of 15° gores centered on the prime meridian or based on civil boundaries. In general, the nautical gores are preferred as the civil boundaries are from a community effort maintained at <http://new.earthtools.org/webservices.htm> and may be subject to error. An example of a nautical time zone query (the default) is:

```
collection("ext:timezone")/latitude=32.8/longitude=243.8!
```

Arguments can appear in any order and are:

- **longitude** and **latitude** – Position of the location in degrees for which the time zone is to be computed. Latitude must be in the interval between 90° (N) and -90° (S), and longitude should be between 0 and 360° E, although the mediator will also take degrees west as a negative

number. Note that this is the format in which Tethys stores latitude and longitude, so no conversion is needed.

- **tztype** – Must be “nautical” or “civil.” Nautical is the default if omitted. Use civil with extreme caution as it has not been well verified.

The query above produces the following XML:

```
<?xml version="1.0" encoding="utf-8"?>
<timezone xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.earthtools.org/timezone-1.1.xsd">
  <version>1.1</version>
  <location>
    <latitude>32.800000</latitude>
    <longitude>-116.200000</longitude>
  </location>
  <offset>-8</offset>
  <suffix>U</suffix>
  <localtime/>
  <isotime/>
  <utctime/>
  <dst>Unknown</dst>
</timezone>
```

In most cases, the element of interest in a time zone query is the offset which in this case is -8 hours from UTC.

### 3.1.2.10.3 NOAA ERDDAP data

NOAA’s Environmental Research Division Data Access Program ([ERDDAP](#)) provides a method of accessing a wide variety of environmental and biological data. ERDDAP provides code to access most environmental data directly but using the Tethys interface permits the queries to be driven by the results of other queries.

ERDDAP organizes data either as a grid or a table and uses data access protocols named griddap and tabledap respectively. Latitude and longitudes follow the same conventions as Tethys, latitude is expressed in degrees North and longitude in degrees East.

The first step in using ERDDAP is to decide what type of data is to be used and then to find an appropriate data set. We will begin by looking for sea surface temperature. Throughout this section, we will use examples from the MATLAB client (section 3.4.4). In this example, we will begin by looking for sea surface temperature and assume that we do not know anything about ERDDAP’s naming conventions. Consequently, we begin by invoking `dbERDDAPSearch` with no search parameters:

```
dbERDDAPSearch(queryH); % Assumes that queryH = dbInit() has been executed
```

This function returns the URL of the ERDDAP search page, but, more importantly, opens a web browser that allows one to search for a desired dataset. The ERDDAP search page allows full-text search as well as search by a number of categories as well as by geo-temporal constraints. In our case, we will use the keywords category to search for `sea_surface_temperature`:

## ERDDAP > Advanced Search

**Directions:** Specify as many or as few search criteria as you want, then click **Search**. Only the datasets that match **all** of the search criteria will show up in the results.

### Full Text Search for Datasets

### Search for Datasets by Category

protocol = (ANY)   
 cdm\_data\_type = (ANY)   
 institution = (ANY)   
 ioo5\_category = (ANY)   
 keywords = sea\_surface\_temperature   
 long\_name = (ANY)   
 standard\_name = (ANY)   
 variableName = (ANY)

### Search for Datasets that have Data within Longitude, Latitude, and Time Ranges

Maximum Latitude:   
 Min and Max Longitude:    
 Minimum Latitude:     
 Minimum Time:   
 Maximum Time:

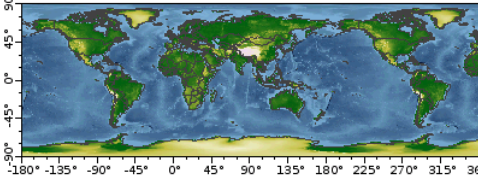


Figure 9 - The ERDDAP search web interface allows one to search for data by specifying multiple criteria.

Geographic constraints can be found by specifying a bounding box either in the boxes or by dragging a rectangle on the map. Time constraints are specified in the ISO 8601:2004 time format used by Tethys (e.g., 2013-01-30T18:22:30Z). After pressing the Search button, the results will show a list of datasets meeting the specified criteria. Each dataset has a unique identifier (Dataset ID) that will be used in all queries involving that dataset.

As one learns the search vocabulary, it becomes relatively simple to search for specific datasets from the command line. Each of the search parameters is joined by an ampersand (&). The following MATLAB and XQuery statements both find the URL for a region within approximately 5 km of an instrument deployed on the southeast side of the Santa Cruz Basin in the Southern California Bight:

```
dbERDDAPSearch(queries, 'keywords=sea_surface_temperature&minLat=33.47&maxLat=33.56&minLong=240.71&maxLong=240.80')
```

Although not covered until the section on the XQuery language, it is worth noting that this MATLAB function simply translates the search to an XQuery which returns a URL that is then opened by the MATLAB function:

```
collection("ext:erddap_search)/keywords=sea_surface_temperature&minLat=33.47&maxLat=33.56&minLong=240.71&maxLong=240.80!
```

Once the ERDDAP data set has been identified, it can be queried to retrieve the data. In this example, we will use the NOAA Coastwatch eight-day composite sea surface temperature dataset from the Aqua MODIS satellite which has the dataset identifier erdMWsst8day. When we locate this dataset in the

web page, we see that the data is listed in the Griddap column indicating that we should expect gridded data to be returned (Figure 10).



<b>Dataset ID</b>	erdMWsst8day
<b>Institution</b>	NOAA CoastWatch
<b>E-mail</b>	
<b>RSS</b>	
<b>Background Info</b>	background
<b>FGDC,ISO,Metadata</b>	F I M
<b>Summary</b>	
<b>Title</b>	SST, Aqua MODIS, NPP, West US, Daytime (8 Day Composite)
<b>WMS</b>	M
<b>Make A Graph</b>	graph
<b>Table DAP Data</b>	
<b>Sub-set</b>	
<b>GriddAP Data</b>	data

Figure 10 - Row from an ERDDAP data search for sea surface temperature.

Other sources, such as buoys, might be expected to return tabular or vector data as shown in the Tabledap column.

Clicking on the data link (Figure 10) exposes more information about the dataset (Figure 11). It shows the dimensions of the dataset indices. To query this dataset, we need to specify four sets of indices: time, altitude, latitude, and longitude. The minimum and maximum values of the dataset can be seen by moving the cursor over the dimension or in the more detailed data attribute structure which is shown beneath the sliders although it has not been reproduced here.



## ERDDAP > griddap > Data Access Form

Dataset Title: **SST, Aqua MODIS, NPP, West US, Daytime (8 Day Composite)**

Institution: NOAA CoastWatch, West Coast Node (Dataset ID: erdMwSstd8day)

Information: [Summary](#) | [License](#) | [FGDC](#) | [ISO 19115](#) | [Metadata](#) | [Background](#) | [Make a graph](#)

Dimensions	Start	Stride	Stop	Size	Spacing
<input checked="" type="checkbox"/> time (Centered Time, UTC)	2012-11-13T00:00:00Z	1	2012-11-13T00:00:00Z	3371	1 day 2h 56m 54s (uneven)
<input checked="" type="checkbox"/> altitude (m)	0.0	1	0.0	1	(just one value)
<input checked="" type="checkbox"/> latitude (degrees_north)	33.47	1	33.59	2321	0.0125 (even)
<input checked="" type="checkbox"/> longitude (degrees_east)	240.7	1	240.80	4001	0.0125 (even)

**Grid Variables** (which always also download all of the dimension variables)

sst (Sea Surface Temperature, degree\_C)

Figure 11 – ERDDAP Aqua Modis 8-day sea surface temperature composite. This dataset is accessed using four dimensions: time, altitude, latitude, and longitude. Positioning the cursor (mouse pointer) over any one of these will show the possible values and the resolution of the data.

The sea surface temperature variable is called sst as shown in the Grid Variables section (Figure 11). To construct a query, we need to specify the following:

- dataset identifier (erdMwSstd8day, in this example)
- variable(s) to be returned (sst, in this example)
- list of dimensions.

Each dimension is enclosed in square brackets, [ ], and this dataset will require four sets of these. Each set of brackets has the following syntax: [StartValue:StrideValue:StopValue]. The values may either be indices from 1 to the number of grid points, or in the units associated with the grid axis (e.g., time, longitude, etc.). When values are specified in units as opposed to indices, they must be enclosed in parentheses ( ). The StrideValue indicates how often data should be returned. A stride of one indicates that all data points are returned, two would be every other one, etc.

Continuing our example, we would have the following values:

- time: [(2012-11-13T00:00:00Z):1:(2012-11-13T00:00:00Z)]
- altitude: [(0.0):1:(0.0)]
- latitude: [(33.47):1:(33.59)]
- longitude: [(240.7):1:(240.80)]

These are all assembled as follows in an XQuery:

```
collection("ext:erddap")/erdMwSstd8day?sst[(2012-11-13T00:00:00Z):1:(2012-11-13T00:00:00Z)][(0.0):1:(0.0)][(33.47):1:(33.59)][(240.7):1:(240.80)]!
```

Tethys will return an XML document with the data:

```
<?xml version="1.0" encoding="utf-8"?>
<table>
  <header>
    <time units="UTC" type="String"/>
    <altitude units="m" type="double"/>
    <latitude units="degrees_north" type="double"/>
    <longitude units="degrees_east" type="double"/>
    <sst units="degree_C" type="float"/>
  </header>
  <row>
    <time>2012-11-13T00:00:00Z</time>
    <altitude>0.0</altitude>
    <latitude>33.475</latitude>
    <longitude>240.7</longitude>
    <sst>16.83</sst>
  </row>
  <row>
    <time>2012-11-13T00:00:00Z</time>
    <altitude>0.0</altitude>
    <latitude>33.475</latitude>
    <longitude>240.7125</longitude>
    <sst>16.77</sst>
  </row>
  ... more entries
</table>
```

which consists of a header element describing the data and the units in which they are represented. Following the header is a series of row elements, where each element describes a grid point.

Language-specific interfaces will parse this information into a usable format. As an example, the MATLAB command `dbERDDAP` expects a query handler and the portion of the query string between `collection("ext:erddap)/` and the exclamation point (!):

```
data = dbERDDAP(queries, 'erdMWSstd8day?sst[(2012-11-13T00:00:00Z):1:(2012-11-13T00:00:00Z)][(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 – Description of the axes
- Data – A structure with the returned data.
- dims – The dimensions of the data

## 3.2 Adding data to Tethys

Data can be imported into Tethys in a variety of formats. When the data being imported is not stored in XML, a translator needs to be used to map the row-oriented data to XML. This translator is called a source map. This section contains a brief introduction to source maps (more details can be found in Appendix 6) and explanations of how to import documents, update previously imported documents, and view attachments to imported documents.



Such sources can come from comma-separated value files, spreadsheet workbooks, databases, and anything else that supports the open database connectivity protocol.

### 3.2.1 Source maps

When possible, we recommend that detectors generate XML conforming to the Tethys schema (see section 5). However, it is not uncommon to want to import detections from an already established format. Most such formats can be thought of as tables, and facilities exist to import them from comma-separated value lists, spreadsheet workbooks, and a wide variety of database products. Regardless of the data source, a common issue is how the data source, with its own organization and field names, can be translated into the XML required by Tethys. To do this, we use what is called a source map. Source maps are described in more detail in the Importing Data Into Tethys (DataImport) manual, but will be briefly introduced in this section. While it is useful to understand how source maps are structured, the Web client provides an interface for creating a source map where you can import your source data and then drag and drop the required Tethys schema elements onto the corresponding fields of your source data.

Each source map consists of an XML element called `<Mapping>` with three children:

`<Name>` – Unique name used to specify which mapping should be used. In our example, the map Name is *SIO.SWAL.Detections.Analyst.v1*, but any name may be used.

`<DocumentAttributes>` – This section contains information that will be added to the document so that the database knows which schema should be used to validate the document. In most cases, this section should just be copied from one of the existing examples.

`<Directives>` – This element contains children that specify how the translation is to be done. Within the Directives elements, one can specify sheets of a workbook to use or sequential query language (SQL) queries to databases. Each row of these data sources is then processed according to the instructions.

The following example shows the details of a `<Directives>` element from *SIO.SWAL.Detections.Analyst.v1* which is included in the sample database:

```
<Mapping>
  <Name> SIO.SWAL.Detections.Analyst.v1 </Name>
  <DocumentAttributes> ... omissions ... </DocumentAttributes>
  <Directives>
    <Detections>
      ... omissions ...
      <OnEffort>
        <Sheet name="Detections">
          <Detection>
            <Entry>
              <Source> [Input file] </Source>
              <Dest> Input_file </Dest>
            </Entry>
            <Entry>
```

```

    <Source> [Start time] </Source>
    <Kind> DateTime </Kind>
    <Dest> Start </Dest>
  </Entry>
  <Condition>
    <Predicate>
      <Operand> [End time] </Operand>
      <Operation op="empty" retain="iffalse"/>
    </Entry>
    <Source> [End time] </Source>
    <Kind> DateTime </Kind>
    <Dest> End </Dest>
  </Entry>
  ... omissions ...
</OnEffort>
... omissions ...
</Detections>
</Directives>
</Mapping>

```

For each row in the *Detections* sheet of a workbook, a <Detection> element will be produced as shown below with many of the elements omitted:

```

<Detections ...attributes...>
  ... many omissions, only Start shown for each Detection ...
  <OnEffort>
    <Detection> ... <Start> 2012-06-01T14:50:22.52Z </Start> ... </Detection>
    <Detection> ... <Start> 2012-06-01T14:50:23.9Z </Start> ... </Detection>
    <Detection> ... <Start> 2012-06-01T14:50:41.32Z </Start> ... </Detection>
    ... other rows ...
  </OnEffort>
</Detections>

```

Elements nested within a <Directives> element are simply copied, with the exception of the following processing elements:

- <Sheet> and <Table>: Both of these elements are used to specify data from the current data source. The <Sheet> directive should be used with spreadsheets and expects the attribute **name** to indicate which sheet of the workbook will be used. For <Table>, the **query** attribute is used and may be any valid SQL query for the database. <Table> also works on Microsoft Excel spreadsheets, treating sheets as tables. Note that Microsoft's libraries append a dollar sign (\$) to sheet names when they are treated as database tables.
- <Entry>: Specifies how fields in the spreadsheet or database will be transformed to an element expected by Tethys. <Entry> elements are always children of <Sheet> or <Table> elements and contain children describing the translation:
  - <Source>: One or more field names, enclosed in square brackets [ ]. Multiple fields can be specified and will be merged.
  - <Kind>: Specifies the data format. For text fields, this is not needed. Valid kinds are: **LongLat**, **DateTime**, **Integer**, **Number**, and **SpeciesCode**. These are described more fully in details.

- <Default>: Value to use in cases of missing data. If no default is provided, no value is produced. Defaults may be strings, numbers, or the special value {id} which indicates that the document identifier is to be used.
- <Dest>: Name of the output element.

### 3.2.2 Importing documents to Tethys

Data can be added to the database with XML documents that conform to the Tethys schema (Appendix 5) or with documents in a different format that have a source map. Ideally, tools used by researchers will use the Nilus application programming interface to generate XML directly, but many existing tools generate data in other formats. Consequently, Tethys provides data import support for the following data sources:

- Microsoft Excel workbooks
- Comma-separated value lists (text files with commas between entries)
- Microsoft Access
- Most databases

Data import services are provided using industry standard open database connectivity (ODBC), and hence most databases including but not limited to MySQL, Oracle, Visual Fox Pro, PostgreSQL, etc. should function, provided that the database has an ODBC driver and that it has been installed on the Tethys server machine. For data sources such as Excel and comma-separated value lists, it is assumed that the first row contains the names of each field. Filenames can contain numbers, letters, dashes, and periods. Commas and ampersands (&) in filenames are not supported.

See Importing data in the Tethys Data Import manual for complete details.

### 3.2.3 Updating existing documents

Please see the Document Modification section of the Data Import manual for details on how to update or modify existing documents.

### 3.2.4 Viewing attached images in a web browser

Many Detections include audio and/or image files, which are attached when adding the Detections to the Tethys database. The uploader program will upload any image and audio files listed in the image and audio fields. These are expected to be in directories with the same name as the detection document with –image and –audio appended. As an example, if the detection document is *Socal36Odontocetes-SilbidoDetector* the image and audio directories would be *Socal36Odontocetes-SilbidoDetector-image* and *Socal36Odontocetes-SilbidoDetector-audio*. A maximum of 500 files can be attached to any detections document.

One way to view an attached image is using the REST server component of Tethys.

First, the Tethys server needs to be running. In this example, the server is running as localhost with port 9779.

Next, open a web browser such as Firefox. To view an image, type in the address using the server location. Note that the document Id does not have the file type suffix (such as .xls) but that the name of the image file does have the file type.

For attached images you will need the following information:

- Server (e.g., localhost)
- Port (e.g., 9779)
- Collection (e.g., Detections)
- Detections doc Id (e.g., ALEUT02BD\_MF\_MFAOrca\_ajc)
- Image name (e.g., Other-ALEUT2BD-20101211T061447.jpg)

The address we would then type into the web browser would be:

[http://localhost:9779/Attach/Detections/ALEUT02BD\\_MF\\_MFAOrca\\_ajc?Image=Other-ALEUT2BD-20101211T061447.jpg](http://localhost:9779/Attach/Detections/ALEUT02BD_MF_MFAOrca_ajc?Image=Other-ALEUT2BD-20101211T061447.jpg)

which should produce:

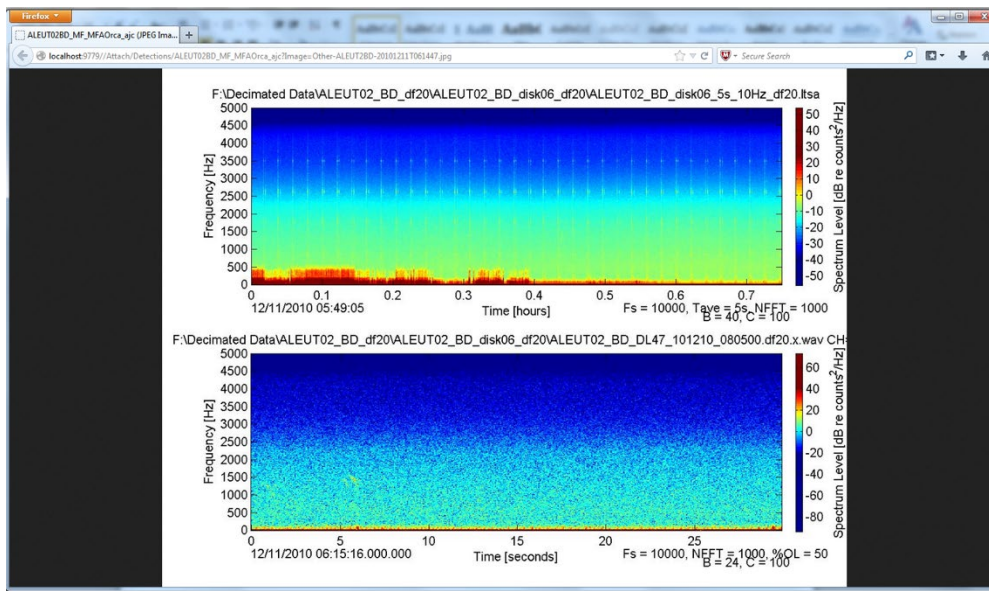


Figure 12 – Detections document attachment viewed in a web browser.

### 3.3 Removing/modifying data

Data removal and modification is described in the Data Import manual.

### 3.4 Query

A query is essentially a request for information that you make to your database. Perhaps you want the start times of all Risso’s dolphin clicks in your database, or to know if there has been any effort to look for fin whales at a specific location—these are each individual queries that you could make to your database.

There are numerous ways you can query your database. You can use the Data Explorer or Web client to make queries from a web browser, you can take advantage of many template queries that are included with the clients, or, if you need to make queries that are more advanced, you can write your own queries in the XQuery language. This section will walk through all of these options.

### 3.4.1 Data Explorer

The Data Explorer is a user-friendly way to investigate the data available in Tethys. As you click around the interface, queries are being executed in the background: the user does not need to know how to formulate a query to use this tool successfully.

Details on using the Data Explorer are provided in the [Tethys-DataExplorerManual](#).

### 3.4.2 Web client

The Web client provides an interface for the user to form database queries from a web browser. While an understanding of the way data is structured in Tethys is helpful for using this tool, no programming experience is required.

The Web client enables the user to form both simple and advanced queries, save the results of a query, and save the query itself (written in XQuery) for use or modification in one of the other clients.

Details on using the Web client are provided in the [Tethys-WebClientManual](#).

### 3.4.3 Java client

The Java client provides basic functionality and consists of two Java classes. The Client class represents client information about the Tethys server, and the Queries class provides an interface to the server.

The Client constructor has a single argument, a URL which will be used to communicate with the Tethys server. This can either be an instance of the `java.net.URL` or `java.lang.String` classes. Once the client is created, one can create a Queries instance which takes an instance of Client as the argument to its constructor:

```
import dbxml.Client;
import dbxml.Queries;

// Initialize connection to Tethys on port 9779 (the default port).
// Substitute an appropriate domain, use https:// for a secure socket layer connection
// The java.net.URL class can also be used.
Client client = new Client("http://tethys.nwfsc.noaa.gov:9779");
Queries queryHandler = new Queries(client);
```

We refer to instances of the Queries class as query handlers. Once a query handler has been constructed, the Client instance is no longer needed. The query handler provides the following methods:

- `boolean ping()` – Attempts to contact the Tethys server and returns true if the server is capable of responding, otherwise false.
- `String Query(String query)` – Executes the XQuery contained in query and returns the result in a string.

- **String QueryTethys(String query)** – Similar to the Query method, except that the following namespace declarations are prepended to the query:

```
declare default element namespace "http://tethys.sdsu.edu/schema/1.0" at "tethys.xsd";
import module namespace lib="http://tethys.sdsu.edu/XQueryFns" at "Tethys.xq";
```

These imports load in the schema and library module for Tethys and are required for many Tethys queries. The primary purpose of this method is to let interactive users perform short queries without the need to declare namespaces. In most production code, the user should use Query or QueryReturnDoc.

- **Document QueryReturnDoc(String query)** – Executes the XQuery contained in query and returns the result as a document object model (DOM) object. DOM is an in-memory graph representation of an XML document and provides a standard interface to access the various elements of the XML document. The Document class is defined in org.w3c.dom.Document. There are numerous tutorials on the DOM interface that can be found on the web and in books on XML.
- **String xmlpp(String xmldoc)** – Given a serialized XML document (one represented as a string), format it to be aesthetically pleasing with proper indentation. Note that this can be slow for large documents. As an example, if xmldoc contains:

```
<Detection><Input_file>H:\SOCAL44N_disk06\SOCAL44N_disk06_5s_100Hz.ltsa</Input_file><Start>2011-08-17T02:58:12.500Z</Start><End>2011-08-17T03:18:02.500Z</End><Event>02/28/1213:05:56</Event><SpeciesId>180444</SpeciesId><Call>Clicks</Call><Parameters><Subtype>A</Subtype></Parameters></Detection>
```

it will be transformed to:

```
<Detection>
  <Input_file>H:\SOCAL44N_disk06\SOCAL44N_disk06_5s_100Hz.ltsa</Input_file>
  <Start>2011-08-17T02:58:12.500Z</Start>
  <End>2011-08-17T03:18:02.500Z</End>
  <Event>02/28/12 13:05:56</Event>
  <SpeciesId>180444</SpeciesId>
  <Call>Clicks</Call>
  <Parameters>
    <Subtype>A</Subtype>
  </Parameters>
</Detection>
```

- **URL getURL()** – Returns the URL to which the query handler is associated.
- **String getURLString()** – Returns a string representation of the URL returned by getURL().
- **String queryJSON(String json)** – Simple method for querying against the database. Allows querying the database without knowing XQuery. The json argument contains information indicating what should be queried. It consists of several sections:
  - **select** – A set of selection criteria indicating what should be selected.
  - **return** – A list of elements to be returned.
  - **species** – An indication of how SpeciesId elements should be interpreted, e.g., using a specific set of abbreviation names, Latin names, etc.

- namespaces – A value of 0 removes XML namespaces in returned results (you might want this if you don't know what an XML namespace is), a value of 1 (default) leaves them intact.
- `enclose` and `enclose_join` – When the query results in results from repeated items, setting the optional `enclose` to 1 results in each group of repeated items being enclosed in an XML document. `Enclose_join` is used to specify an XML element name that wraps sets of documents that have been joined from multiple collections, such as a query that uses both detections and deployments. `Enclose_join` defaults to the name `<Record>`.

Complete details and example queries using the JSON notation can be found in the Tethys Web Services documentation. An optional numerical argument following the `XpathExpr` has the following meaning:

0. Execute the query
  1. Show the internal query plan. This is not useful for most users.
  2. Show the XQuery that was generated from the JSON specification.
- **String XPath(String xmlstr, String XPathExpr)** – Execute an XML path (XPath) expression against an XML string. XPath is a simple language for querying XML documents. XPath enables navigation through nested XML elements by separating them with slashes. As an example, consider the following XML document:

```
<outer>
  <inner>hi there</inner>
  <inner>you</inner>
</outer>
```

If we wanted to extract the text associated with the first inner element, we could use the XPath `/outer/inner`. To retrieve the second inner element, we could append an index in square brackets: `/out/innter[2]`. XPath is described in numerous web tutorials and books. Note that this is a utility function and does not interact with the Tethys server, it is simply used to process XML that may have been retrieved from a Tethys server.

The following methods require a collection name and a document identifier (`docId`). The `docId` is the document identifier that is frequently the same as the `Id` element in some documents, but not necessarily so. A documents identifier can be found by using the `base-uri` XQuery function which returns `dbxml://collection_name/docId`. As an example, this XQuery would list the document names for all Detection documents:

```
for $d in collection("Detections")
  return base-uri($d)
```

You can also see a list of document names in a web browser (e.g., Chrome or Firefox) by typing the collection name after the web address. For example, if the server is running on an unsecure channel, `http://mytethys.dom/Detections` where `mytethys.dom` is replaced with your local server name.



- `String getDocument(String collection, String docid)` – Return specified document from the specified collection.
- `Document getDocumentObjectModel(String collection, String docId)` – Return specified document from the collection. Result is formatted as a document object model (DOM).
- `String removeDocument(String collection, String docid)` – Remove specified document from the specified collection.

### 3.4.4 MATLAB client

The MATLAB client can be used to add data to the database and to query the database. See the Tethys-MATLAB Cookbook for a more detailed description of using the MATLAB client with Tethys. The Cookbook has a list of common MATLAB functions as well as numerous examples for getting started.

There are two subfolders within the *MatlabClient* folder: *db* and *vis*. The functions under *db* are related to accessing the database while the functions in the *vis* directory provide support for visualizing data.

Once MATLAB has started, add the *db* and *vis* directories to your path. This can be done using MATLAB's `pathtool` or `addpath` commands. The `pathtool` command allows you to save the path for the next time you start MATLAB. Alternatively, `addpath` commands can be put in the *startup.m* file which is executed when MATLAB starts. See the MATLAB documentation for details.

Once the path is set, use `dbInit()` to create a database query handler:

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

We will use `query_h` as the name of our query handler throughout this section, but any variable name is fine. The database query handler is the first step in using Tethys in MATLAB and will allow the user to query the Tethys metadata database that has been created on the server and to use the Tethys methods to perform spatial and temporal analyses. While the Tethys interface to MATLAB permits the user to query the Tethys server using XQuery, MATLAB functions for a number of common queries have been written to generate the queries and parse the results into structures that are easily usable within MATLAB.

For any function, one can type “help” or “doc” followed by the function name. As an example, in the MATLAB command window, typing:

```
doc dbInit
```

brings up the help browser with the following text:

```
dbInit(optional_args)
Create a connection to the Tethys database.
With no arguments, a connection is created to the default server
defined within this function.
```

```
Optional args:
```

'Server', NameString - name of server or IP address  
Use 'localhost' if the server is running the  
same machine as where the client is executing.  
'Port', N - port number on which server is running  
'Secure', false|true - make connection over a secure socket

Returns a handle to a query object through which Tethys queries are served.

If you were running your server in unencrypted mode (`secure-socket-layer=false`) on the same computer as your MATLAB client, you would type:

```
query_h = dbInit('Server', 'localhost', 'Secure', false);
```

to obtain the query handler. Many functions in the Tethys MATLAB client take optional arguments that are specified by keyword (e.g., 'Secure') and value (e.g., false) pairs. Again, one can see the optional arguments by using `help` or `doc` followed by the function name in the MATLAB command window.

Once the query handle has been created, it is possible to perform a variety of tasks.

#### **3.4.4.1 Uploading/Removing data**

The MATLAB client provides an interface for submitting documents to the database. This process is described both in the MATLAB Cookbook and the Data Import documentation. Removal of documents i

#### **3.4.4.2 Querying the database**

All MATLAB queries to the database using the Tethys methods require a query handler to be created. While the query handler is capable of querying XML directly using the XQuery language, a number of common queries have been packaged into functions that can be used without any knowledge of XQuery.

Functions to access the database start with the prefix `db`. Most of these functions require the query handler returned from `dbInit` as their first argument. Optional arguments let users specify criteria such as spatial or temporal information, species or call types of interest, etc. Queries can be made using a single value ('Site', 'M') or using a list ('Site', {'M', 'N'}) as desired.

Below are some examples of functions you can use to query Tethys. See the Tethys-MatlabCookbook for a more extensive list and examples.

Deployments:

- `dbGetDeployments()` – Retrieves information about deployments.

Detections:

- `dbGetEffort()` – Retrieves information about effort to detect species.
- `dbGetDetections()` – Retrieves start and end times of detections meeting the specified criteria.

Localizations

- `dbGetLocalizationEffort()` – Retrieves information related to efforts to localize or find the direction of acoustic sources.
- `DbGetLocalizations()` – Retrieves localization information.

Environmental data:

- `dbDiel()` – Provides information about sunrise and sunset.
- `dbGetLunarIllumination()` – Provides information about lunar illumination percentage (without taking into account cloud cover).
- `dbERDDAPSearch()` – Searches NOAA’s Environmental Research Division Data Access Program (ERDDAP) servers to find an appropriate server for various oceanographic data such as ice coverage over a specified spatial-temporal range.
- `dbERDDAP()` – Retrieves oceanographic data from a specific server.

### 3.4.4.3 Visualization

The functions for data visualization will produce a variety of plots. Some of the more commonly used visualizations are:

- `visLocalizations` – Plot localizations on a map, requires MATLAB mapping toolbox.
- `visLunarIllumination` – Given the results of a lunar illumination query, add it to an existing plot.
- `visPresence` – Show presence/absence on a daily plot designed to make diel patterns apparent.
- `visTracks` – Plot WGS84 tracks on a map, requires MATLAB mapping toolbox.
- `visWeeklyEffort` – Show count of hours with detections on a weekly basis.

See the Tethys-MATLAB Cookbook for examples using this and other visualization function.

### 3.4.5 Python client

The Python client is designed primarily for administrative purposes and to provide low-level access (e.g., users writing their own XQueries). The tasks details below can all be invoked from the command line and they all have the following options in common:

- h or --help Show a help message and exit.
- port Specify a port number (defaults to 9779).
- servertype Server transport layer type, do not set.
- server Server address (defaults to the name specified for the server during the installation of clients).

When you open a command console, you will first need to change the directory to the folder where the client is located (e.g., *C:/Program Files/Tethys/PythonClient*). You will then type the path to the *Python39* folder that contains *python.exe* followed by the python file you wish to run. For example, to run *checkpoint.py* you would type:

```
pythonclient_path> python_path\python.exe checkpoint.py
```

where

`pythonclient_path` indicates the path to your *PythonClient* folder

`python_path` refers to the path to the *Python39* folder that contains *python.exe*

### 3.4.5.1 Create a checkpoint

`checkpoint.py` is used for creating a checkpoint in the database. Checkpoints verify that any changes to the database are in a stable state. The database automatically checkpoints itself each time it starts. See section 4.1 for details on checkpoints.

### 3.4.5.2 Import documents

`import.py` provides a mechanism to import documents into a collection. Use of this client is described in section Tethys Data Import manual.

### 3.4.5.3 Remove all documents

`clear_documents.py` removes all documents from a collection. A list of collection names to be cleared are given, e.g.,

```
pythonclient_path> python_path\python.exe clear_documents.py Deployments SpeciesAbbreviations
```

**Use with caution.** The primary use for this command is for clearing out a collection prior to importing from a database. As an example, the Scripps Whale Acoustics Lab stores instrument deployments in a MySQL database. To update the Deployments collection, the collection is first cleared, then `import.py` is used to import all of the deployments.

### 3.4.5.4 Remove a specific document

`remove.py` is used to remove a specific document from a specified collection. Document names are either based on:

- the filename of the submitted data (without the extension) or
- a database name followed by an `_` and a number. As an example, the Scripps Whale Acoustics Lab imports deployments from the database HarpDB and the deployments are named HarpDB\_1, HarpDB\_2, etc.

Example:

```
pythonclient_path> python_path\python.exe remove.py Deployments HarpDB_235
```

### 3.4.5.5 Shutdown the server

`shutdown.py` requests the server to exit. Queries in progress are handled and then the server will stop.

Example:

```
pythonclient_path> python_path\python.exe shutdown.py
Connecting to server: http://127.0.0.1:9779 plain text (UNSECURED)...
<Tethys> exiting </Tethys>
```

### 3.4.5.6 Rebuild a collection

`update.py` is used for rebuilding collections from source documents that have already been submitted. See the Data Import Manual section on adding documents that have been accidentally removed for details.

### 3.4.5.7 XQuery from Python

`client.py` provides an example of how to use an XQuery from Python. Its purpose is to provide an example for users who wish to write XQueries against the server without the need for MATLAB or Java. For more details on XQuery, see section 3.4.6.

## 3.4.6 XQuery

XQuery is a language used to query XML databases. Walmsley's (2006) book on XQuery provides an excellent and complete introduction to XQuery and is recommended reading for people who wish to become experts in XQuery. Many useful queries can be performed using the MATLAB client with no knowledge of XQuery whatsoever. However, for users who wish to create complicated custom queries, investing the time to learn XQuery will be beneficial. Our goal in this section is to provide a gentle and incomplete introduction to XQuery, deferring advanced materials to other sources such as Walmsley's book.

It is helpful to run queries interactively when designing them, and the MATLAB client provides a good way to do this. Let's walk through a few examples of more basic queries using the XQuery methods in the MATLAB environment. The MATLAB client is described in section 3.4.4.

From the MATLAB command window, we first need to define a query handler:

```
query_h = dbInit('Server', 'localhost');
```

Now that a query handler exists, we can query our metadata database. Note that the query handler is an instance of the JavaClient's `dbxml.Queries` class and has a variety of methods (function calls) that are available to it and reported in the JavaClient documentation.

Two methods that are important for querying a Tethys database are `Query` and `QueryTethys`. Let us first consider how we might count the number of ad-hoc detections in a Tethys database. In a database of humpback whale detections, we might have detected other species in addition to the humpbacks even though our main objective was to detect humpbacks. While we might still be interested in recording these other detections, we would record them as off-effort detections as we were not looking for them systematically. In Tethys, we report these detections as `OffEffort` detections.

We will use `query_h.Query()` to execute the XQuery. Let us begin by defining the XQuery. The first step is to declare a query namespace. The namespace will let the query handler know that the data we are querying conforms to the Tethys schemata. The first line of the query is:

```
'declare default element namespace "http://tethys.sdsu.edu/schema/1.0";
```

This is followed by the query itself. In this simple example, we will ask the system to count the number of off-effort (ad-hoc) detections. We use the XQuery function count and a so-called path statement that traces the elements in the Detections collection from the root element Detections down to the off-effort detections:

```
count(collection("Detections")/Detections/OffEffort/Detection')
```

If you wish to explore the Tethys schemata, you may wish to read the advanced query section of the Tethys Web Client documentation. Combining this declaration and the path, we can execute the query in MATLAB as follows:

```
query_h.Query(['declare default element namespace "http://tethys.sdsu.edu/schema/1.0";', ...  
'count(collection("Detections")/Detections/OffEffort/Detection)'])
```

Declaring the default element namespace is a common operation, and using the QueryTethys method will automatically add the declaration for you. The following is an equivalent query:

```
query_h.QueryTethys('count(collection("Detections")/Detections/OffEffort/Detection)')
```

Another example would be if we wanted a list of all the species that have effort in the database. Again, we first need to have a query handler. Next, we create a namespace, and then execute a query that reports the species for which we have effort. This path is enclosed in an XQuery function, distinct-values, that returns each species for which we have effort only once:

```
Query_h.QueryTethys(...  
'distinct-values(collection("Detections")/Detections/Effort/Kind/SpeciesId)')
```

### 3.4.6.1 Advanced queries – Example 1

In addition to the simple queries described above, complex queries are possible using Tethys. In this example, we would like a list of scientific names for all of the species for which we have effort. However, our Detections documents encode species names as an ITIS taxonomic serial number (TSN). Fortunately, the ITIS collection can let us associate the scientific name with the TSN.

We could write a FLOWR<sup>3</sup> expression to match the TSN with the species name, but Tethys has a number of library functions that will let us do this with minimal effort.

First, we will declare our namespace:

```
declare default element namespace "http://tethys.sdsu.edu/schema/1.0";
```

Next, we need to import the software library containing the function that will map from a TSN to a Latin species name, which the ITIS database refers to as a *completename*.

```
import module namespace lib="http://tethys.sdsu.edu/XQueryFns" at "Tethys.xq";
```

---

<sup>3</sup> Due to the keywords for, let, order by, where, and return, queries in the XQuery language are frequently referred to as FLOWRs (pronounced flowers)

In the previous examples, the results of our queries were displayed on the screen. Here we will instead save the query returns as a variable, called `$tsns`.

```
let $tsns := distinct-values(collection("Detections")/Detections/Effort/Kind/SpeciesId)
```

`$tsns` will be the list of each `SpeciesId` for which we have effort. Next, we loop through each of the TSNs stored in `$tsns` and use the function `tsn2completename` to perform the mapping. We prefix `tsn2completename` with the namespace abbreviation `lib` that we defined in our module import.

```
for $tsn in $tsns
  return lib:tsn2completename($tsn)
```

where `$tsn` is a variable consisting of the scientific names, `$tsns` is the output of our previous query, `lib` refers to the software library created containing the scientific names for all species, and `tsn2completename` is the Tethys method to match `SpeciesId` or another species format with scientific names.

The final query is as follows:

```
declare default element namespace "http://tethys.sdsu.edu/schema/1.0";
import module namespace lib="http://tethys.sdsu.edu/XQueryFns" at "Tethys.xq";
let $tsns := distinct-values(collection("Detections")/Detections/Effort/Kind/SpeciesId)
for $tsn in $tsns
  return lib:tsn2completename($tsn)
```

This text can be placed in a string and executed via the query handler. Note that the `QueryTethys` method not only declares the default element namespace, but also the declaration for the library namespace. As such, if the `QueryTethys` method is used instead of `Query`, the first two lines could be omitted.

### ***3.4.6.2 Advanced queries – Example 2***

Let's form a query to find all deployments where effort has been put into finding Pacific white-sided dolphin clicks:

```
declare default element namespace "http://tethys.sdsu.edu/schema/1.0";
```

```
<Result xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
{
for $detections in collection("Detections")/Detections
  (: 180444 is the Pacific white-sided dolphin :)
  (: We'll see how to find this automatically later :)
  where $detections/Effort/Kind/SpeciesId = 180444
  return
    <Effort>
      {$detections/DataSource}
    </Effort>
}
</Result>
```



XQueries return XML documents. One strategy in designing XQueries is to design a document skeleton and have XQuery fill in portions of it. Tethys provides a generic document element called `<Result>` whose schema permits any valid XML, and we note that the XML is bracketed by:

```
<Result xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
...
</Result>
```

The `xsi` namespace declared in `<Result>` is not mandatory, but it will prevent the children of `<Result>` from having the `xsi` namespace in each element. The namespace is present in Tethys documents as they reference a schema, which is part of the `http://www.w3.org/2001/XMLSchema-instance` namespace. To distinguish the XQuery code from the XML document, curly braces `{ }` are used and we see that `{ }` brackets the XQuery code in our query.

Let us turn our attention to the XQuery code itself which we number for convenience:

```
1. declare default element namespace "http://tethys.sdsu.edu/schema/1.0";
2. for $detections in collection("Detections")/Detections
3.   (: 180444 is the Pacific white-sided dolphin :)
4.   (: We'll see how to find this automatically later :)
5.   where $detections/Effort/Kind/SpeciesId = 180444
6.   return
7.   $detections/DataSource
```

Let us begin by examining portions of line 2. The path expression (referred to as an XPath), `collection("Detections")/Detections`, returns a list of documents in the collections detection whose top-level element is `<Detections>`. This should be every document in the Detections collection, but if there were documents in the Detections collection that started with a different element, they would not be included. The for loop will assign the variable `$detections` to each one of these documents at the `<Detections>` level. All XQuery variables start with a dollar sign (`$`).

If we wished to access the Description element for a group of detections, we could do so with `$detections/Description`. The optional where clause allows us to restrict, or filter, the selection of documents based on their contents. In this case, we are looking for Pacific white-sided dolphins, TSN 180444, and will see later how we could write a query for something less obscure, such as using the common name "Pacific White-sided Dolphin" or the scientific name "Lagenorhynchus obliquidens." This is also noted in the comments of lines 3 and 4. Text between a `(: and :)` is interpreted as a comment and is ignored.

To restrict ourselves to detections where analysts or algorithms were searching for any type of call associated with this species, we construct an XPath from `$detections` to the species identifier. The path is based on the structure and meaning of the schema (see section 5). In this case we specify the kind of effort to which we are restricting ourselves with an equality check: `$detections/Effort/Kind/SpeciesId = 180444` (line 5).

For each document that remains after our filter for Pacific white-sided dolphins, we wish to return information about the data source. This is indicated by the return statement on line 6 which is followed

by either an XQuery expression or XML. In the above example, it is followed by an XQuery path expression that states that the `DataSource` element and its children which specify instrument deployment (`DeploymentId`) or instrument deployment group (`EnsembleId`) should be returned. A sample output might look like the following:

```
<DataSource>
  <DeploymentId>SOCAL38-M</DeploymentId>
</DataSource>
any other entries...
```

Alternatively, we could have placed XML in the return statement and the lines following the return could have read:

```
<Effort>
  {$detections/DataSource}
</Effort>
```

When the return contains XML, the curly-brace markers `{` and `}` are used to indicate that items between the markers are processed as query code. Here, the XQuery path `$detections/DataSource` is used, but any XQuery expression is valid. The result would be similar, except each `<DataSource>` element would be enclosed within an `<Effort>` element.

#### 3.4.6.2.1 Let statements and library modules

We extend the previous query with the introduction of other variables and a library function call to eliminate the need to know the TSN for Pacific white-sided dolphins. We also wrap the query in an element called `Result`. The `Result` element has a namespace attribute (`xmlns:xsi`) that indicates that the results from a schema definition. This is optional and does not affect correctness, but it will make the results more readable.

As the XQuery is nested inside the XML element `Result`, curly braces are used to indicate that the text between the `Result` and `</Result>` is XQuery to be executed and not XML data. Variables can be introduced with the `let` statement. Here, we use a library function to look up the TSN from the Latin species name and assign it to a variable:

```
1. declare default element namespace "http://tethys.sdsu.edu/schema/1.0";
2. import module namespace lib="http://tethys.sdsu.edu/XQueryFns" at "Tethys.xq";

3. <Result xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
4. {
5. (: Find TSN from species name :)
6. let $id := lib:completename2tsn("Lagenorhynchus obliquidens")
7. for $detections in collection("Detections")/Detections
8. where $detections/Effort/Kind/SpeciesId = $id
9. return
10. <Effort>
11.   {$detections/DataSource}
12. </Effort>
13. }
14. </Result>
```

This XQuery returns the same values as the previous ones but adds an additional import to access the library module in the namespace `http://tethys.sdsu.edu/XQueryFns` (line 2) which is given the abbreviated name `lib`. The lookup is performed by function `completename2tsn` in line 6 which will map the species name to a TSN if it is in the ITIS collection. The value is then substituted into the equality of line 8. The results from this query will be identical to the previous one.

A description of the other functions in the module can be found in section 7. Most of the functions are useful for translating back and forth between TSNs and species names, common names, or local abbreviations.

#### 3.4.6.2.2 Nested loops and conditional statements

We continue this with a slightly more complicated query which finds the detections themselves in documents where we know that we are looking for Pacific white-sided echolocation clicks. Note that we could have looked for detections in *every* Detection document, but it is more efficient to restrict our search to places where detection effort has been made as there should not be any on-effort detections in any document where no effort has been placed in finding them.

```
1. declare default element namespace "http://tethys.sdsu.edu/schema/1.0";
2. import module namespace lib="http://tethys.sdsu.edu/XQueryFns" at "Tethys.xq";

3. <Result xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
4. {
5.   let $id := lib:completename2tsn("Lagenorhynchus obliquidens")
6.   for $detections in collection("Detections")/ty:Detections
7.   where $detections/Effort/Kind/SpeciesId = $id
8.   return
9.     for $d in $detections/OnEffort/Detection
10.    where $d/SpeciesId = $id and $d/Call = "Clicks"
11.    return if ($d/Parameters/Subtype)
12.              then $d
13.              else ()
12. }
13. </Result>
```

This example begins in a similar manner to the first one but begins to differ in what is returned starting at line 8. The return value is actually a nested XQuery. Once we have identified a document where there was effort to find Pacific white-sided dolphins, we look through the on-effort detections (line 9) and then retain calls from Pacific white-sided dolphin echolocation clicks with a where clause (line 10). As this is a nested XQuery, it also needs a return value. Rather than returning every detection, we wish to only return echolocation clicks that have a subtype associated with them. In Soldevilla et al. (2008), we identified two types of echolocation clicks which we called subtypes A and B. We store this distinction in Tethys by adding a `<Subtype>` element as a child of `<Parameters>`.

The if statement on line 11 is true if there is a `Parameters/Subtype` element relative to the current detection, `$d`. When this is true, the detection is returned; otherwise, the empty sequence is returned, which does not change the output. The following shows a sample output from this query:

```
<Result xmlns="http://tethys.sdsu.edu/schema/1.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  ... many detections omitted ...
```

```

<Detection>
  <Input_file>H:\SOCAL44N_disk06\SOCAL44N_disk06_5s_100Hz.ltsa</Input_file>
  <Start>2011-08-17T02:58:12.500Z</Start>
  <End>2011-08-17T03:18:02.500Z</End>
  <Event>02/28/12 13:05:56</Event>
  <SpeciesId>180444</SpeciesId>
  <Call>Clicks</Call>
  <Parameters>
    <Subtype>A</Subtype>
  </Parameters>
</Detection>
<Detection>
  <Input_file>H:\SOCAL44N_disk07\SOCAL44N_disk07_5s_100Hz.ltsa</Input_file>
  <Start>2011-09-15T02:49:27.500Z</Start>
  <End>2011-09-15T03:18:17.500Z</End>
  <Event>03/02/12 10:58:35</Event>
  <SpeciesId>180444</SpeciesId>
  <Call>Clicks</Call>
  <Parameters>
    <Subtype>A</Subtype>
    <UserDefined>
      <peak_1_low>21700.0</peak_1_low>
      <peak_2>27700.0</peak_2>
      <peak_3_high>38400.0</peak_3_high>
    </UserDefined>
  </Parameters>
</Detection>
</Result>

```

Although not shown here, the order by clause can be used to sort the results and is placed just before the return clause.

### 3.4.7 Case study: Northeast Fisheries Science Center Minke Boing analysis

NOAA's NEFSC conducted an analysis of Minke boing calls where received levels were estimated for individual pulses near the beginning, middle, and end of each call (Table 2). The **Detection** element in the Detections schema (set of rules indicating valid syntax) has methods to support recording nonstandard data. In the **Parameters** section, there is an element called **UserDefined**. This is an example of a user definable field that can have arbitrary data including nested XML elements that can be queried like any other element. All of the methods to import data into Tethys support arbitrary children being added to user defined elements. See the Data Import Manual for examples of user definable fields.

This is where XML can shine by allowing both standard and non-standard elements to be mixed. The schema for detections indicates that any set of elements are permitted under **UserDefined**, permitting extensibility where it is needed while preserving consistency elsewhere.

The source material contains multiple lines, and some of the information is associated with the entire call (average received level) while other parts are pulse specific. This can be represented via XML with the following structure which describes the first Boing in the table:

```

<Detection>
  <Start> ... [field values represented by ...] </Start>
  <End> ... </End>
  <SpeciesId> ... </SpeciesId>
  <Call> Boing </Call>

```

```

<Parameters>
  <ReceivedLevel_dB> 101.841103727 </ReceivedLevel_dB>
  <MinFreq_Hz> 45.6 </MinFreq_Hz>
  <MaxFreq_Hz> 169.1 </MaxFreq_Hz>
  <UserDefined>
    <pulse>
      <number> 1.0 </number> <!-- Call number -->
      <signal>
        <Selection> 5.0 </Selection>
        <end_s> 36988.673 </end_s>
        <rms_amp_dBre1uPa> 103.684142949 </rms_amp_dBre1uPa>
        <f_rms_amp_u> 139.1 </f_rms_amp_u>
        <tag> pt1_A </tag>
        <start_s> 36988.374 </start_s>
        <Channel> 2.0
      </Channel>
    </pulse>
    <ambient>
      <low_Hz> 47.6 </low_Hz>
      <Selection> 2.0 </Selection>
      <end_s> 37010.763 </end_s>
      <rms_amp_dBre1uPa> 99.0726664225 </rms_amp_dBre1uPa>
      <f_rms_amp_u> 81.8 </f_rms_amp_u>
      <tag> pt1_A </tag>
      <high_Hz> 171.1 </high_Hz>
      <start_s> 37010.463 </start_s>
      <Channel> 2.0 </Channel>
    </ambient>
  </UserDefined>
</Parameters>
</Detection>

```

The **UserDefined** element allows embedding arbitrary information such as whistle contour tracks, etc.

Selection	Original/ Ambient	Channel	Begin Time (s)	End Time (s)	Low Freq (Hz)	High Freq (Hz)	Tag	F- RMS Amp (u)	RMS Amplitude (dB re 1 μPa)	Subtracted RMS Amp (dB re 1 μPa)	Pulse train	Average RL
2	A	2	37010.463	37010.763	47.6	171.1	pt1_A	81.8	99.1	101.8	1	98.7
5	O	2	36988.374	36988.673	45.6	169.1	pt1_A	139.1	103.7			
4	A	2	37011.004	37011.293	46.7	195.4	pt1_B	87.4	99.6	97.1		
6	O	2	36998.247	36998.535	47	195.8	pt1_B	109.2	101.6			
6	A	2	37011.471	37011.737	52.5	164.7	pt1_C	78.7	98.7	97.1		
7	O	2	37009.577	37009.843	51.2	163.5	pt1_C	102.1	101.0			
8	A	5	37294.504	37294.748	50	171.7	pt2_A	70.3	97.8	96.5	2	96.4
8	O	5	37276.193	37276.437	49.7	171.3	pt2_A	93	100.2			
10	A	5	37294.914	37295.18	47.7	168	pt2_B	57.8	96.1	95.4		
9	O	5	37283.932	37284.198	46.9	167.1	pt2_B	78.8	98.7			
11	A	5	37293.887	37294.142	45.7	153.4	pt2_C	65.3	97.1	97.4		
10	O	5	37293.255	37293.51	45.5	153.1	pt2_C	93.8	100.3			

Table 2 - NOAA NEFSC Minke boing source level information. Information from pulse trains are measured near the beginning, middle, and end of each boing. Alternating lines show measurements of energy during the pulse (original) and ambient background. The estimated received levels are given along with the average received level.

## 4 Maintenance

### 4.1 Checkpoints

All database operations are made as transactions, which means that if the database dies in the middle of an operation, it should not corrupt itself. A series of transaction log files are written to the databases's *db* directory as operations are conducted. Each time the Tethys server is started, it audits the log files to verify that the database is in a consistent state (a checkpoint operation, that can also be requested manually, see the PythonClient `checkpoint.py`). If anything is amiss, it uses the log files to restore the database to a stable state with the last modification instructions either omitted or applied successfully. After verification, the database is checkpointed, and the log files are moved into the *db/verified\_logs* directory. The verified logs can be periodically deleted to save disk space.

The database itself is stored in a set of files with the same names as the collections and a set of files that start with `__db.` followed by a number. The `__db` files are a relational database decomposition of the XML. This is handled automatically.

### 4.2 Backups

Your database should be backed up from time to time. The location of the database files is reported when the server first starts. Here is an example of the first few lines of output from the server:

```
[03/Apr/2023:21:29:36] Welcome to Tethys, 3.0
Examining logs in C:\Users\TethysUser\Tethys\Databases\demodb\db to verify that database is
correct.
This may require a long time (e.g. 1 h) when the logs have not been verified recently
  or a large changes have been made to the database
Log processing started at 2023-04-03 21:29:36.484990
Cache size set to 1.00 GB
Log processing complete. started at: 2023-04-03 21:29:36.484990 elapsed 0 days 00:00:00.442662
Checkpointing database C:\Users\TethysUser\Tethys\Databases\demodb\db... checkpoint complete
Cache size set to 1.00 GB
BSDDB environment initialized
Starting DB XML in transactional mode
Enabling 128 GB query cache at C:\Users\TethysUser\Tethys\Databases\demodb\diskcache
[03/Apr/2023:21:29:41] ENGINE Bus STARTING
[03/Apr/2023:21:29:41] ENGINE Started monitor thread 'Autoreloader'.
[03/Apr/2023:21:29:41] ENGINE Serving on http://0.0.0.0:9779
[03/Apr/2023:21:29:41] ENGINE Bus STARTED
[03/Apr/2023:21:29:41] Web interface at http://tethys.sdsu.edu:9779/Client
```

In addition to backing up the database itself (the *db* folder), the source material used to construct the database should be backed up. This consists of spreadsheets, XML documents, and any media files that are referenced from the XML. Tethys keeps a copy of everything that you have submitted that is not from a database in the source-docs folder contained in the resource directory. It is assumed that other databases have their own backups. **Backing up source material is critical.** Should the database ever become completely corrupted or undergo major revisions, this will allow you to regenerate it with very little effort.



## 4.3 Help! My database has fallen and cannot get up!

### 4.3.1 Server will not start / Server window disappears

Double-clicking on the Tethys batch file should start the server. However, if the server fails, the window will disappear. The first step in troubleshooting is to open a command window and change the directory to the folder where the server was started; for example:

```
cd c:\Users\Tethys\metadata
```

Then start Tethys using the batch file in the directory:

```
tethys.bat
```

This will let you see the error. See below how to handle “database corrupted” messages. If there are permission problems, the account executing Tethys may not have write privileges for the folders containing the database.

### 4.3.2 Database is not responsive

If you are running the database from a Windows command line terminal, and you press a key in the window, all input/output to the database may be paused, effectively blocking all operations. Making the window active by clicking on it and pressing escape will remedy this situation. When you run the database as a service this is not an issue.

Very large queries to external services take time to process. There are three components that drive how fast local data queries are processed:

1. Whether or not the data have been indexed. Most database elements that are frequently used to select data are indexed.
2. The speed and congestion of the network over which they are being transported.
3. The amount of time needed to parse the XML into a usable format at the database client. This amount of time can be greatly reduced by having queries only return the information that is needed rather than everything in a record.

### 4.3.3 Database is corrupted

**IMPORTANT:** If for some reason you start a second instance of the database on the same port, you will receive a message that the database is corrupted even though everything is fine. In this case, simply kill both processes and restart.

In many cases, restarting the database will allow the automatic recovery of the database. However, if things should become damaged beyond repair (we have yet to see this), you can use the dbxml recovery program `db_recover` which is in the *Tethys/dbxml-6.1.4/bin* folder. Open a console window and `cd` to the appropriate Tethys directory, then run it (you may need to change this):

```
cd C:\Users\Tethys\metadata\db
"C:\path to Tethys\Tethys\dbxml-6.1.4\bin\db_recover"
```

The `db_recover` program can also be run with a `-c` option for “catastrophic” database recovery. Finally, should all else fail, remember that Tethys retains the source documents submitted to the database in the `source-docs` directory and the `update_documents.py` Python script described in the data import

manual can be used to reinsert them into a blank database. Documents derived from an external database such as MySQL or Access are not stored; it is assumed that you can use any of the document submission methods described in the data import manual to reinsert these.

## 5 Appendix: XML Schema Diagrams

The following sections of this manual provide an overview of the types of data that appear in each collection. Frequent use is made of XML schema diagrams which show the structure of the XML data. While most of the details of the XML schema need not be understood by the casual reader, a few notations are worth describing:



denotes a sequence of elements that must appear in the order that they appear,



is used to represent a choice, only one of the child elements (or groups of elements) can be used.

Mandatory elements are denoted by dark lines whereas optional elements have light lines. Elements that may be repeated are indicated by labels indicating how many times they can be repeated. As an example, 1 ... ∞, indicates that an element must occur at least one time.

The full Tethys schema can be accessed at <https://tethys.sdsu.edu/documentation/>.

### 5.1 Calibration

Calibration documents record information about the calibration of an instrument (Figure 14). These records can be linked to instruments used during specific deployments. Every Calibration document must include an identifier of the instrument being calibrated (**Id**), the date and time the calibration occurred (**TimeStamp**), the **Type** of calibration (hydrophone, preamplifier, or end-to-end), details on the calibration quality (**QualityAssurance**), a reference intensity for the measurements (**IntensityReference\_μPa**), a list of corresponding frequencies (**Hz**) and responses (**dB; FrequencyResponse**), and details on who is responsible for calibration metadata (**MetadataInfo**).

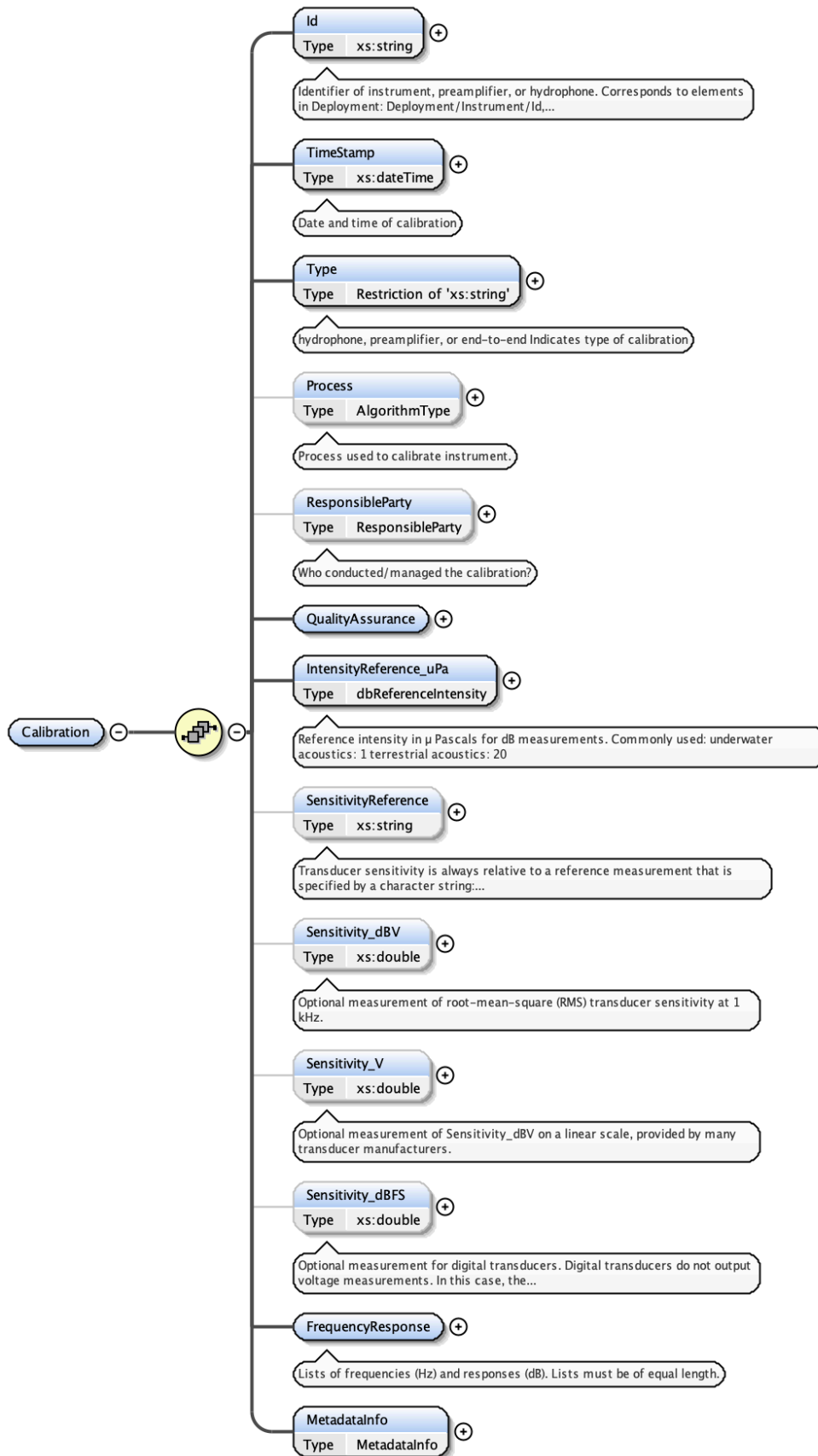


Figure 13 – Calibration schema for recording information about instrument calibration. Dark lines indicate required elements; light lines indicate optional elements.

The **MetadataInfo** element (Figure 15) requires details on the individual responsible for the calibration metadata (**Contact**), the **Date** the metadata was last updated, and information about how often the data are updated (**UpdateFrequency**).

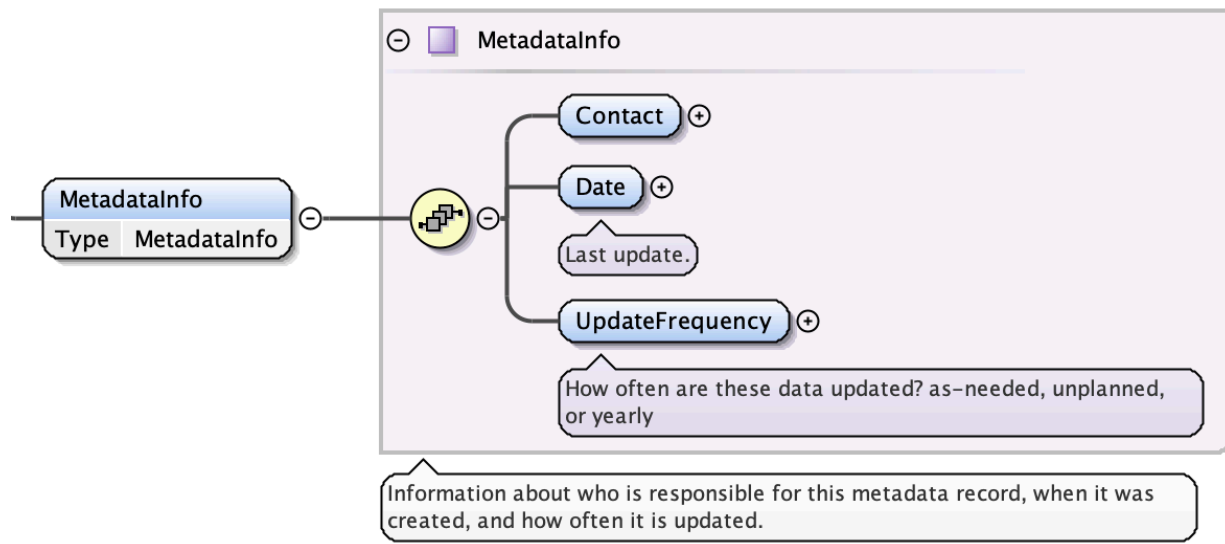


Figure 14 – MetadataInfo is a required element in the Calibration schema to provide details on who is responsible for the calibration record.

The **QualityAssurance** element (Figure 16) requires details about the **Quality** of the calibration (e.g., whether it is unverified, valid, or invalid) and a **Comment** on the **Quality**.

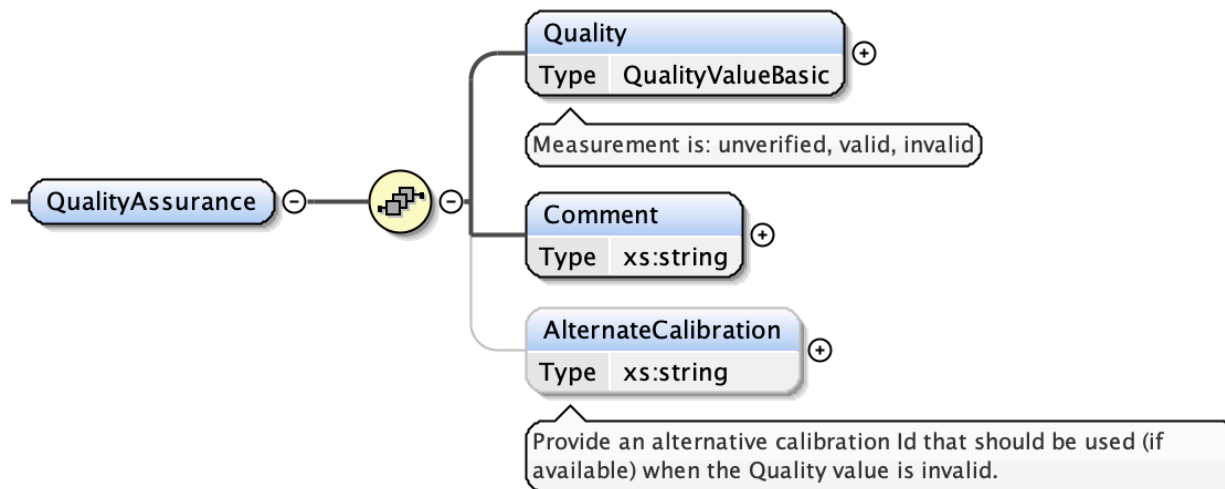


Figure 15 – QualityAssurance is a required element in the Calibration schema to provide details on the quality of the calibration.

Although not required, **Process** allows for additional details about the calibration process used (Figure 17). Required elements include the name of the **Software** used and the **Parameters** used to execute the algorithm. A description of the method, the software version, and any supporting software can also be included.

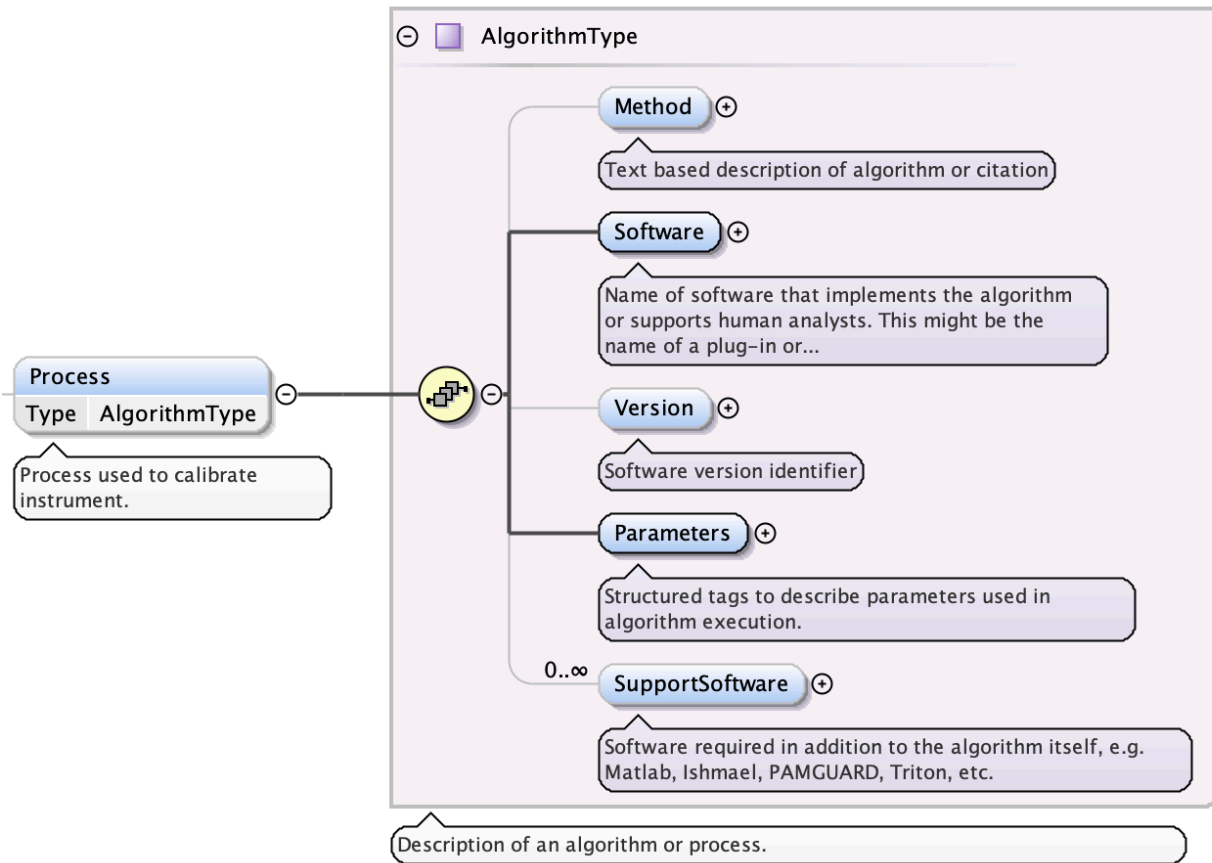


Figure 16 – Process is an optional element in the Calibration schema to provide details of the calibration process used.

Additionally, information about the responsible party (**ResponsibleParty**) can be provided, including names and contact information (Figure 18), as well as information about transducer sensitivity (**Sensitivity\_dBV**, **Sensitivity\_V**, **Sensitivity\_dBFS**) and a sensitivity reference (**SensitivityReference**).

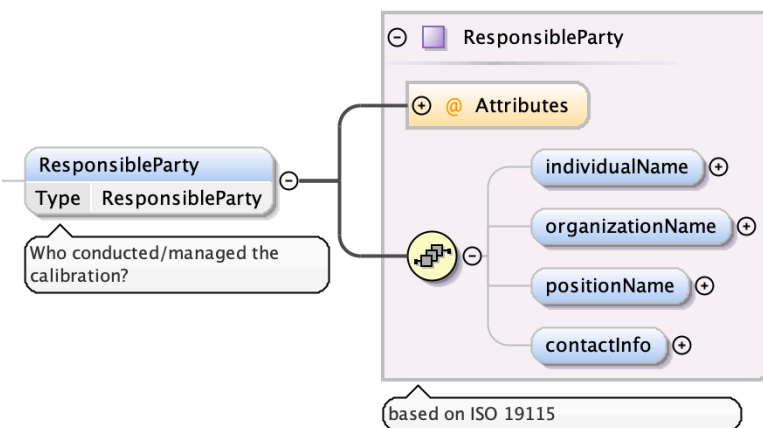


Figure 17 – ResponsibleParty is an optional element in the Calibration schema to provide details on who performed the calibration.

## 5.2 Deployment

Deployment records are used to track how an instrument is configured during the time that it is deployed. If the instrument is not fixed to a single point, a trackline is added showing the instrument's

position at various points during the deployment. Note that while deployments describe the current configuration of the equipment, the deployment record is not designed to describe the instrument itself. It does however provide sufficient information to identify the instrument, whose characteristics may be stored in a separate database.

Every deployment document *must* include the following elements (Figure 19):

- **Id** – a unique identifier for the deployment
- **Project** – a project name associated with the deployment, often related to a geographic region. As an example, at the Scripps Whale Acoustics Lab, we use SOCAL for our high frequency acoustic recording package (HARP) deployments in Southern California.
- **DeploymentId** – a number used to identify the  $n^{\text{th}}$  deployment with respect to a project (or if you prefer, with respect to a site or cruise).
- **Platform** – the type of platform on which the instrument is deployed (e.g., mooring or tag).
- **Instrument** – the **Type** of instrument used in the deployment (e.g., HARP, EAR, D-Tag) as well as its unique identifier (**InstrumentId**)
- **SamplingDetails** – information about the recordings on each channel
- **Data** – information about where the acoustic data are stored. Can also contain data collected during the deployment such as longitudes and latitudes, pitch, roll, etc.
- **DeploymentDetails** – information about the deployment location and time
- **Sensors** – the types of sensors on the instruments (i.e., audio, depth, or other)

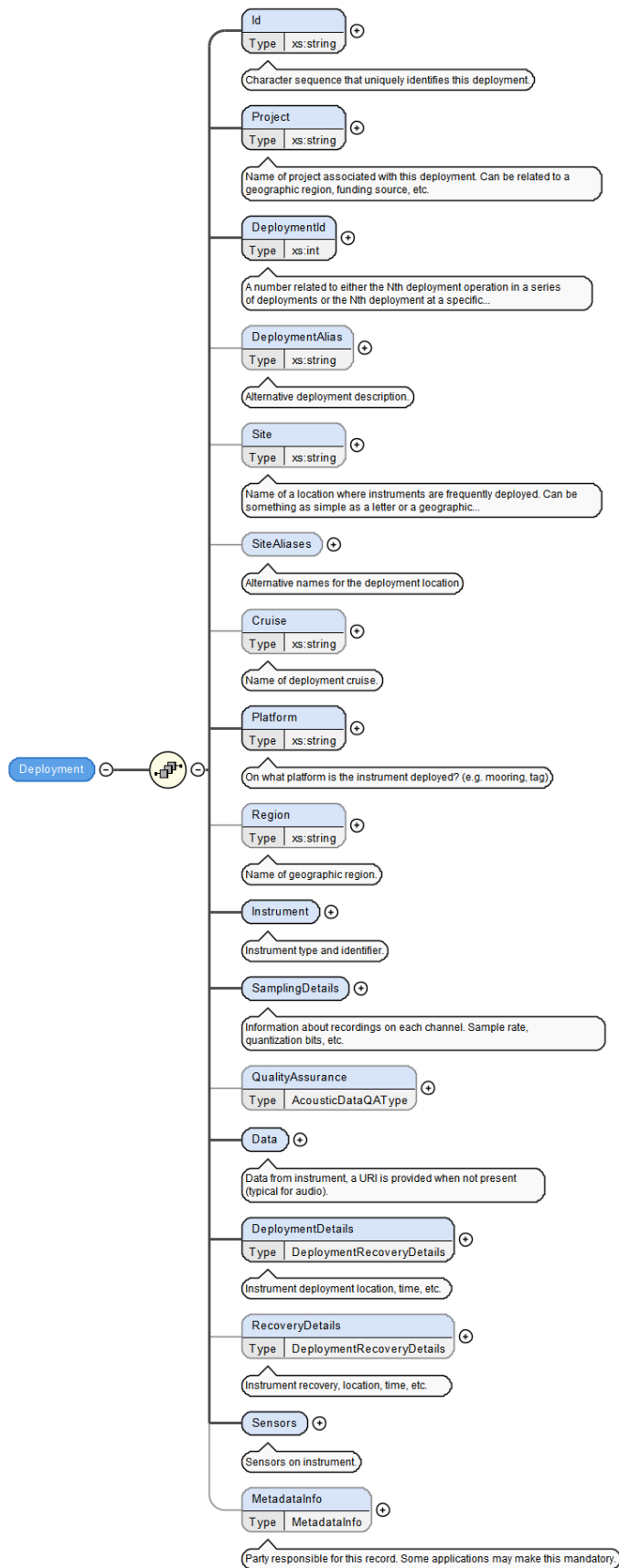


Figure 18 – Deployment schema for recording information about instrument deployments. Dark lines indicate required elements; light lines indicate optional elements.



The **SamplingDetails** element (Figure 20) requires information about the recordings on each channel including the **ChannelNumber** and **SensorNumber**, the **Start** and **End** time of the recording, and the **Sampling** rate. There is also the option to include information about **Gain** and **DutyCycle**.

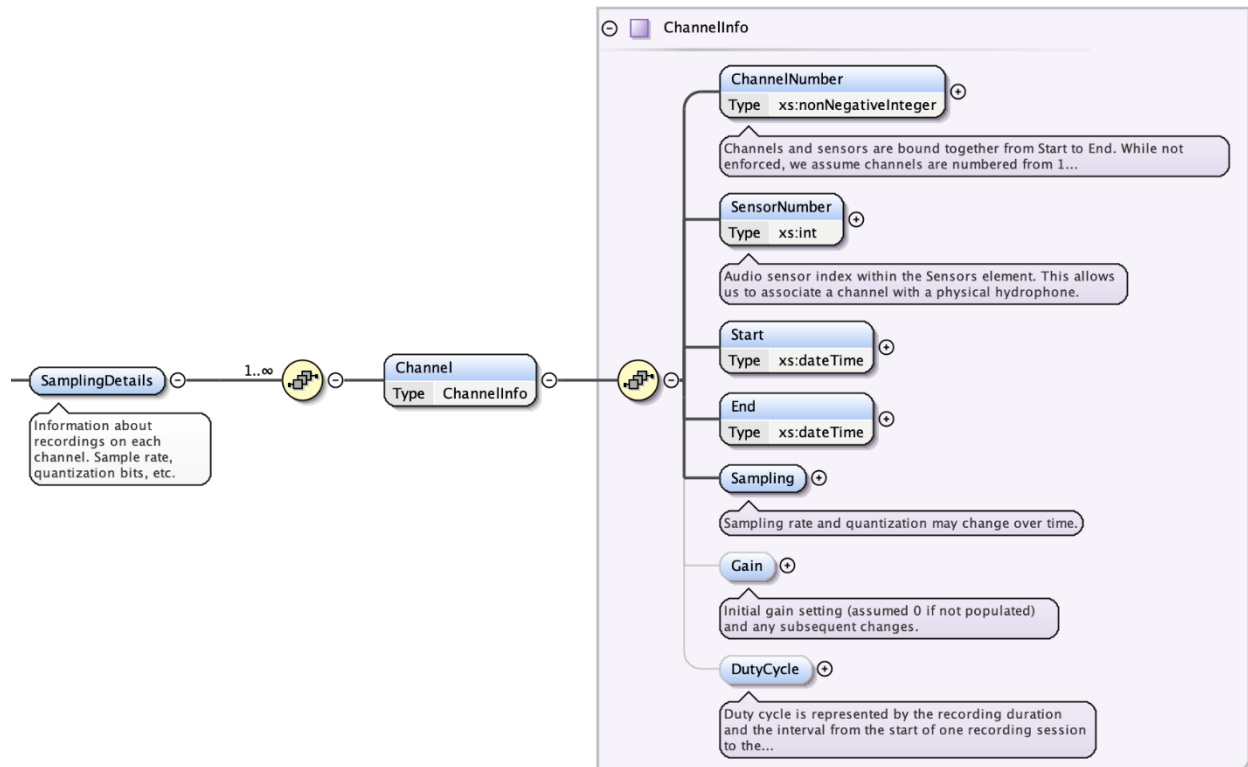


Figure 19 – The **SamplingDetails** element within the Deployment schema contain information about the recordings on each channel during the deployment.

The **Data** element (Figure 21) references where to find the data itself. This element requires an **Audio** Uniform Resource Indicator (**URI**), which can be a formal identifier of a data set, such as a digital object identifier (DOI). When formal URIs are not available, this should be any sort of string indicator as to where the physical data resides, such as a serial number or filing code. The locations of **Processed** and **Raw** data can also be included but are not required. There is also the option to include information about a track line (**Tracks**) if necessary.

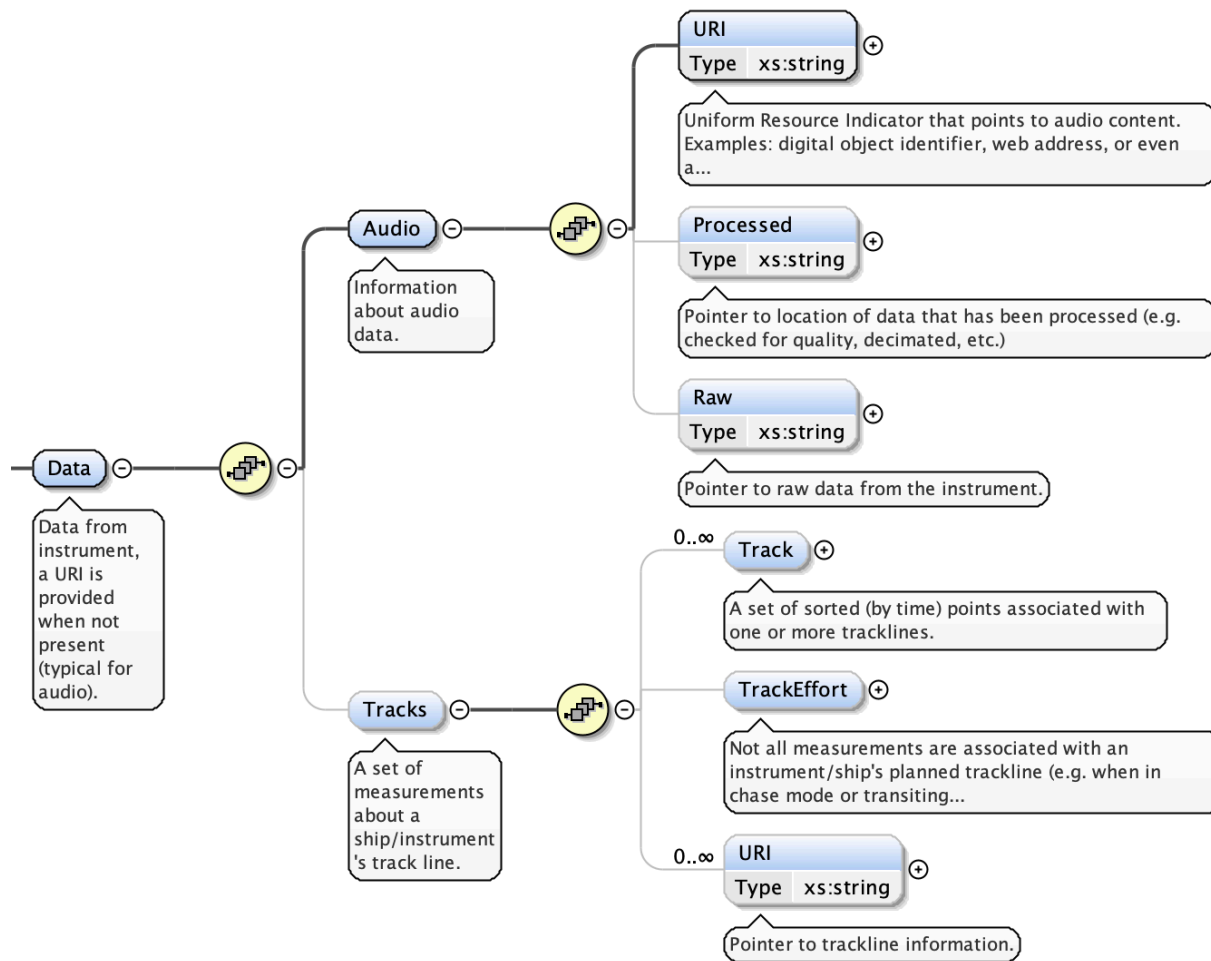


Figure 20 – The Data element within the Deployment schema contains information about the location of the data from the deployment.

The **DeploymentDetails** element (Figure 22) requires information about the location of the deployment (Longitude and Latitude), the time the instrument was deployed (**TimeStamp**) and the time the recording started or stopped (**AudioTimeStamp**). Optional details include the instrument elevation (**ElevationInstrument\_m**) and depth (**DepthInstrument\_m**), the elevation at the deployment location (**Elevation\_m**), the **Vessel** from which the deployment was made, and contact information for the person or party responsible for the deployment (**ContactGroup**). There is an optional **RecoveryDetails** element that contains the same information as **DeploymentDetails** but for instrument recovery. When recovery details are available, it is highly recommended to include them as database users may use these fields for computing deployment duration.

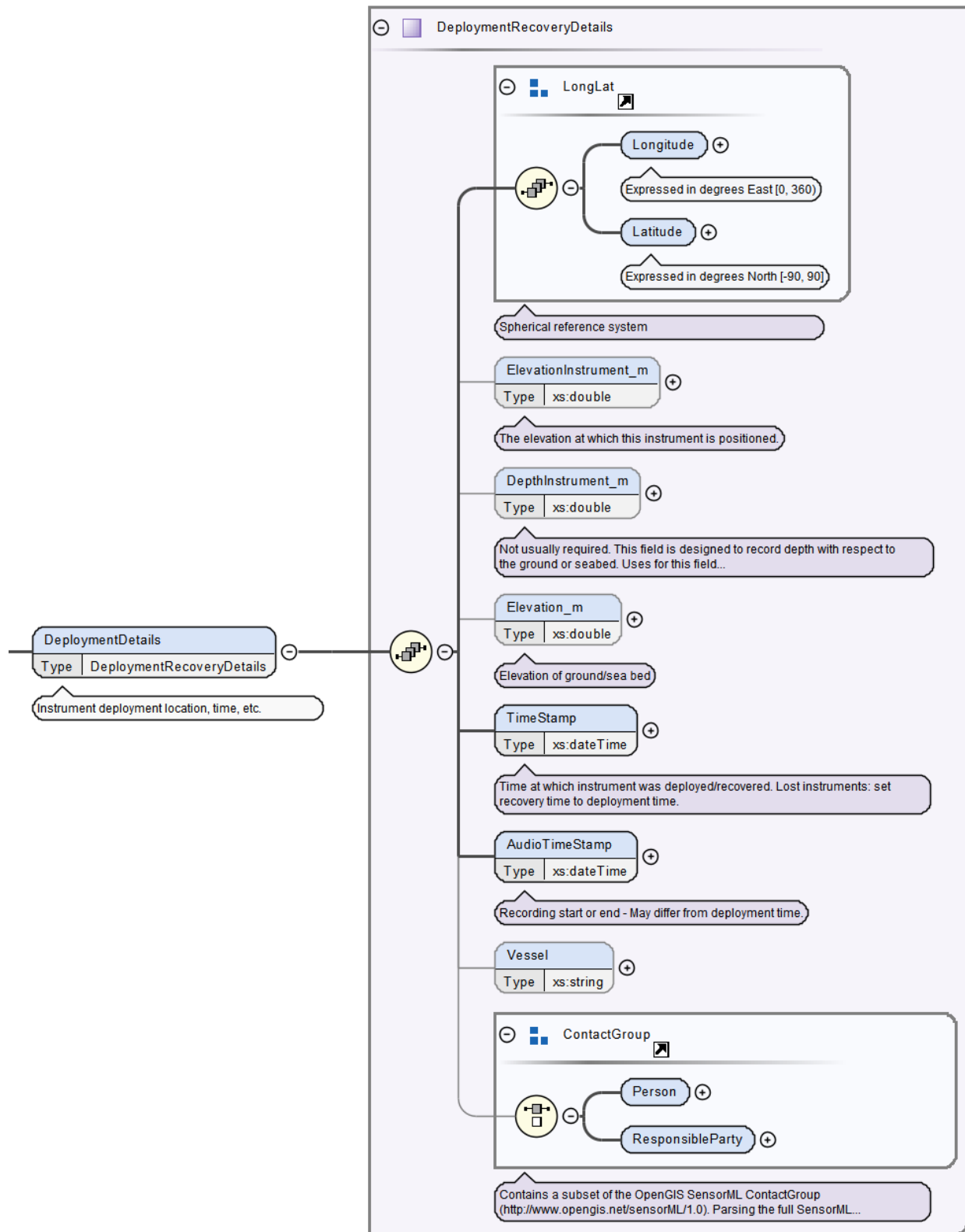


Figure 21 – The DeploymentDetails element within the Deployment schema contains information about where and when a deployment occurred.

The **Sensors** element (Figure 23) describes information about the types of sensors that are associated with the deployment. There are currently elements to describe **Audio** and **Depth** sensors as well as an extendable generic **Sensor** to accommodate other sensor types. Each sensor has a geometry relative to the platform, a name and description, and sensor identifiers that can be used to identify specific pieces of equipment. This information can be used to retrieve information such as calibration data for equipment from a separate instrument database which would be exterior to Tethys.

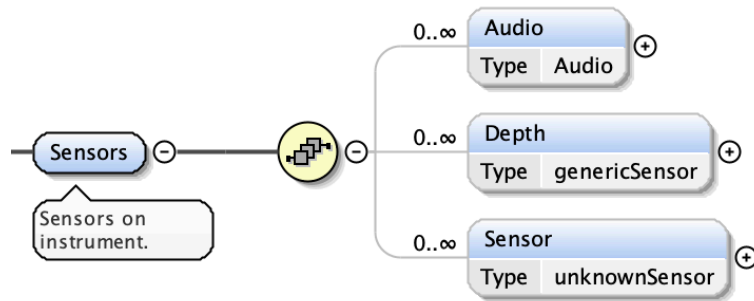


Figure 22 – The Sensors element within the Deployment schema describes the types of sensors associated with a deployment.

Although not required, further information about the deployment can be included such as an alternative description for the deployment (**DeploymentAlias**), a **Site** name, which permits a name to be associated with the area in which the instrument is deployed, an alternative name for the deployment location (**SiteAlias**), the name of the **Cruise** the deployment was a part of, the geographic **Region** in which the deployment occurred, and information about the party responsible for the deployment record (**MetadataInfo**; see Figure 15). **QualityAssurance** allows for the specification of data quality (Figure 24). In most cases, all data retrieved should begin with a single **Quality** element. The **Category** should be "unverified" until the data has been examined, and the **Start** and **End** elements would specify the entire deployment. Later, once the data has been inspected, quality assurance can be specified for different time and/or frequency ranges, marked with their respective categories: good, compromised, or unusable.

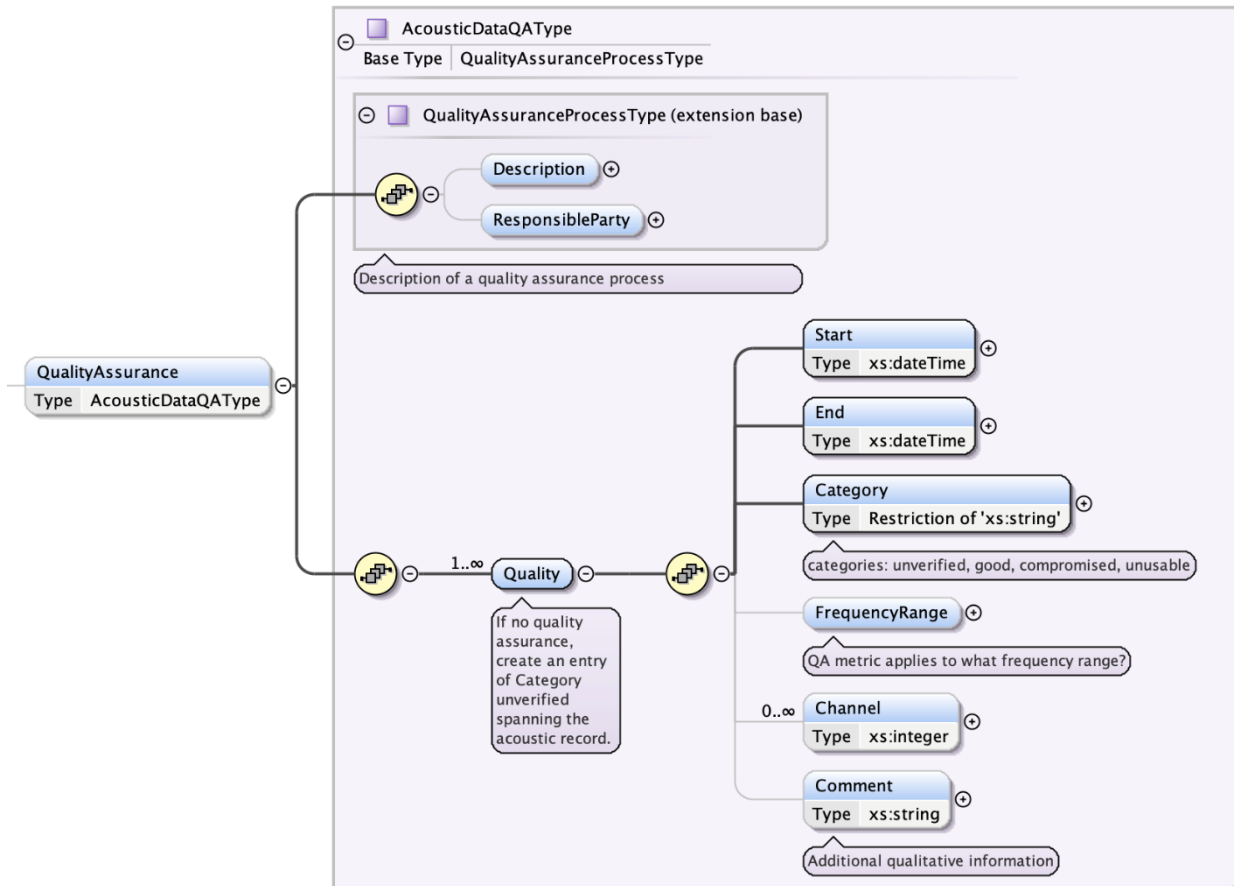


Figure 23 – The optional QualityAssurance element within the Deployment schema allows for the inclusion of details about the quality assurance process.

### 5.3 Detections

The Detections schema contains three primary types of information: information describing the detection process and data on which the process was performed, a specification of the effort, and the detections themselves.

Required elements in the Detections schema include the following (Figure 25):

- **Id** – a unique identifier for the detection document
- **DataSource** – an identifier for a specific deployment (**DeploymentId**) or a set of deployments that have been grouped together in a logical manner, which is referred to as an ensemble (**EnsembleId**).
- **Algorithm** – details on the algorithm that was used to perform the detections. This element contains the same information as the **Process** element in the Calibration schema (Figure 17). It is important that this element is filled out accurately to be able to compare or reproduce results.
- **UserId** – identification of the user that submitted the detection document
- **Effort** – the span of time over which events were searched for in the specified deployment or ensemble and the kinds of events of interest.
- **OnEffort** – a collection of individual detections

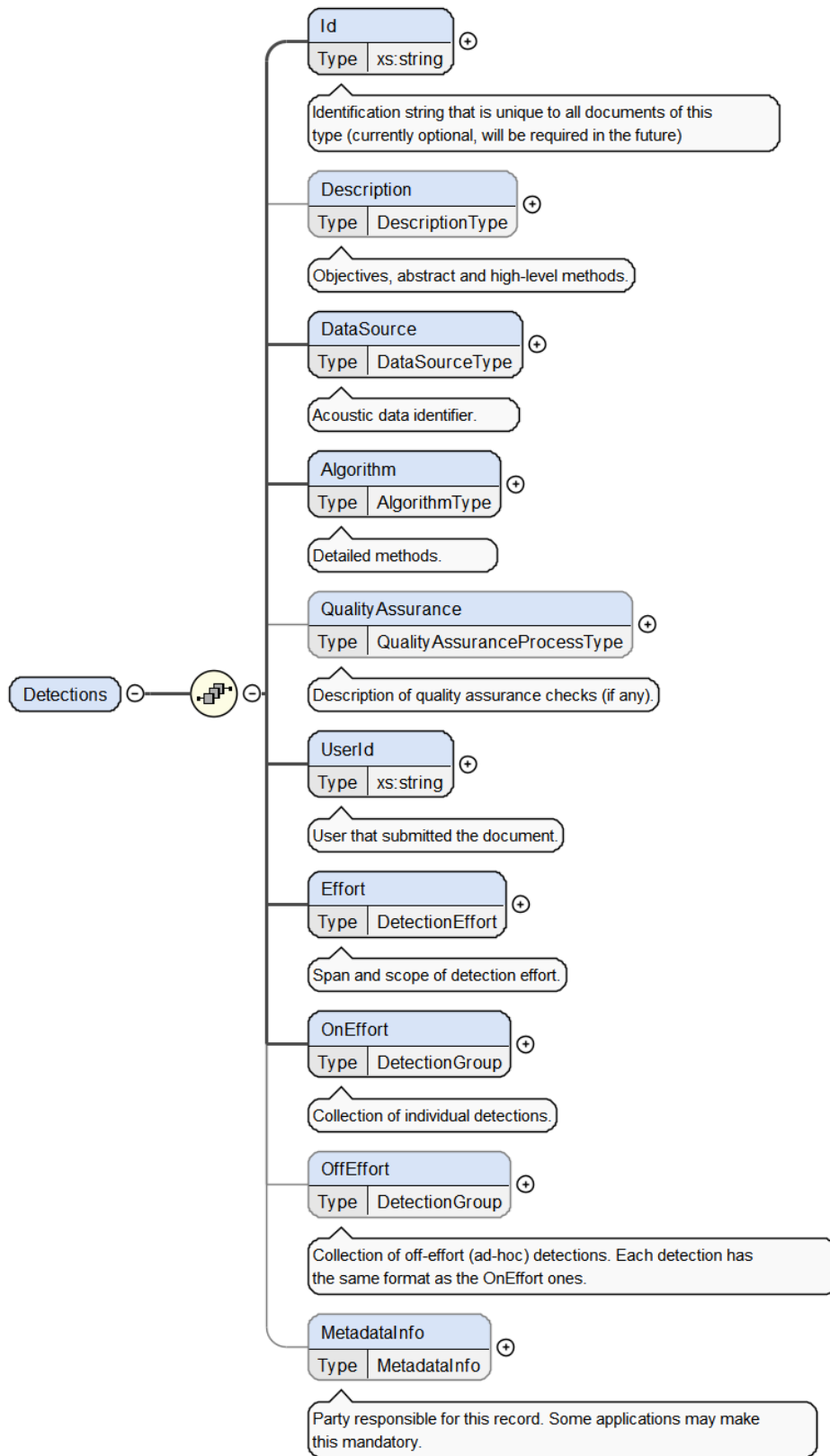


Figure 24 – Detection schema for recording information about detections. Dark lines indicate required elements; light lines indicate optional elements.

The **Effort** element describes the span of time over which events were searched for in the specified deployment or ensemble and what kinds of events are of interest (Figure 26). Required elements include the **Start** and **End** times of the effort and the **Kind** of effort, which includes a **SpeciesId** and **Granularity**. **SpeciesId** is taken from the Integrated Taxonomic Information System (ITIS; [www.itis.gov](http://www.itis.gov)). For anthropogenic events such as ship noise, we typically attribute the species as *Homo sapiens*. To denote calls when the species is not well known, a higher-order label can be used. As an example, if an echolocation click could not be contributed to a specific species, one could record the SpeciesId using the order label of *Odontoceti*. **Granularity** is used to indicate how often detections are recorded. Valid parameters are call, encounter, and binned, which represent the annotation of individual calls, the beginning and end of a set of calls, and the presence of calls within a given time interval, respectively. When binned is selected, the attribute **BinSize\_min** is used to indicate how often (minutes) the presence is reported. In the unlikely chance that the first bin does not start at the beginning of the analysis effort, attribute **FirstBin\_Start** indicates the timestamp of the first presence/absence bin. When **Granularity** is encounter, attribute **EncounterGap\_min** indicates the gap between calls in minutes before subsequent detections are recorded as a new detection. The **Effort** element can optionally include details on any gaps in analysis effort (**AnalysisGaps**) and a reference for any dB measurements (**dBReferenceIntensity\_uPa**).

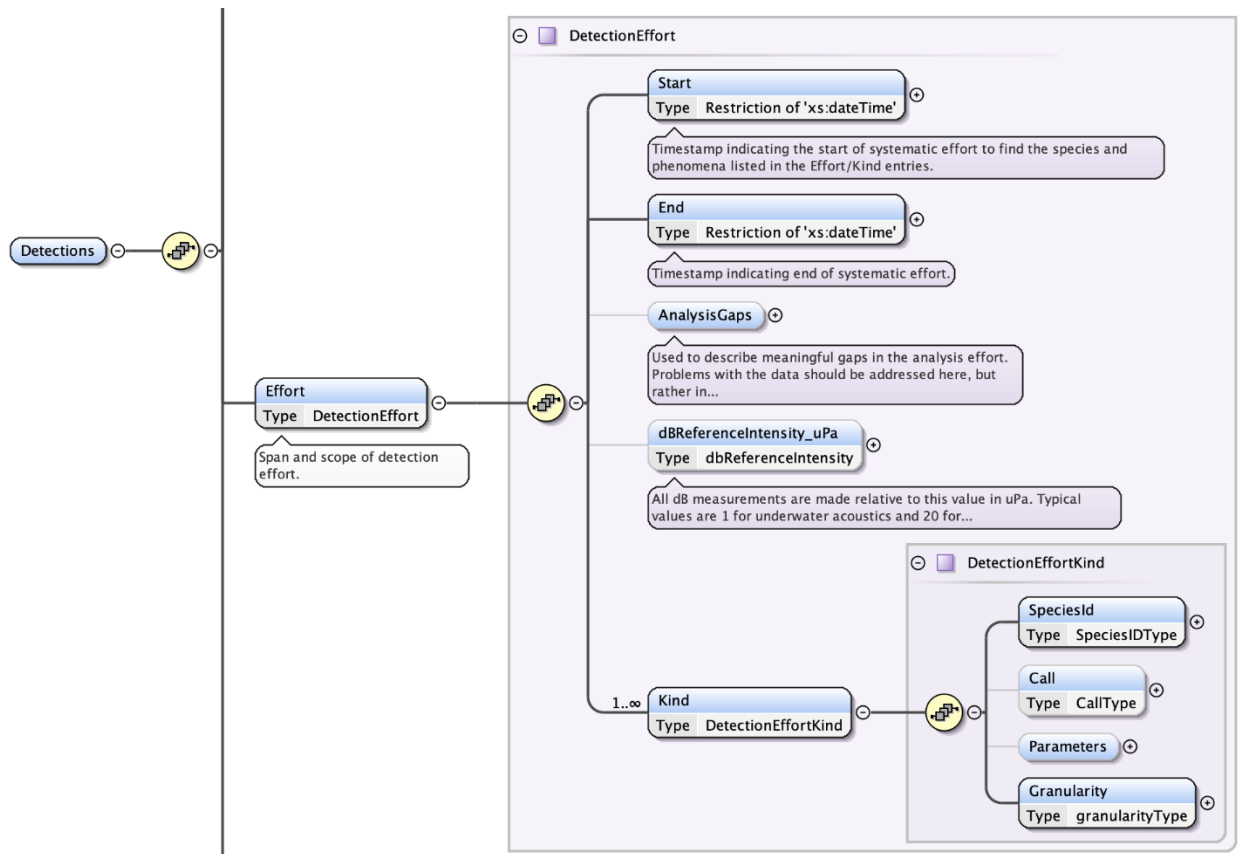


Figure 25 – The Effort element within the Detection schema captures the timespan and types of events that were investigated.

The **OnEffort** and **OffEffort** elements describe the detections themselves. Detections recorded in the **OnEffort** section represent systematic detections corresponding to the **Algorithm** and **Effort** specifications (Figure 27). Ad-hoc detections from non-systematic effort must be placed in the **OffEffort** element. The separation of these two types of detections permits valid inference. A small number of ad-hoc detections do not imply that a species was rarely present, but rather that we happened to notice it on a few occasions. Within the **OnEffort** and **OffEffort** elements, a sequence of 0 or more **Detection** elements are used to specify events.

For any **Detection** element, a **Start** and **SpeciesId** must be included to denote the time at which the event started and the species that was detected, respectively. There are many optional elements that are well described by the comments in the schema and we will discuss only a few in further detail. When the effort is binned, the **Start** time should ideally be the start time of a call within the bin. The **End** time of calls reported as binned detections is optional. The **Parameters** element can be used to describe characteristics of the call. A number of common parameters are included, but Tethys users can define their own parameters as well in a **UserDefined** section. The **Image** and **Audio** elements permit the user to store an image or audio sample of the event. These samples are submitted with the Detections document and can be retrieved from the database.



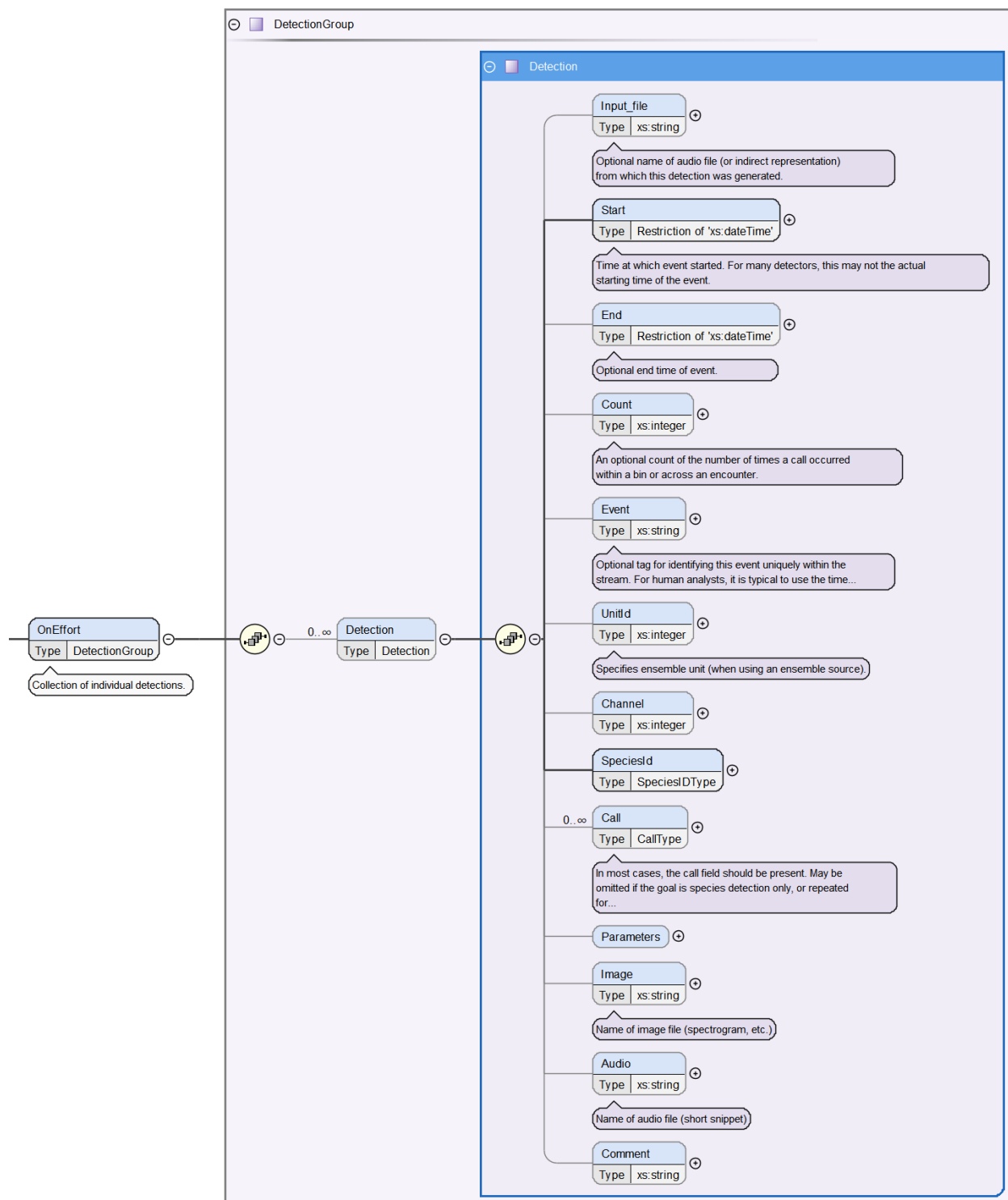


Figure 26 – The OnEffort element within the Detection schema is where individual detections are recorded.

Optional elements in the Detection schema include a **Description** (a high-level textual overview of the detection process, consisting of **Objectives** (e.g., find every call produced by a rare species), **Abstract**, and **Method** elements, which can be text or URLs; Figure 28), a description of any quality assurance checks and the responsible party (**QualityAssurance**), detections that were discovered **OffEffort** (i.e.,

they were not actively being looked for when they were found), and details about the party responsible for the detection record (**MetadataInfo**; see Figure 15).

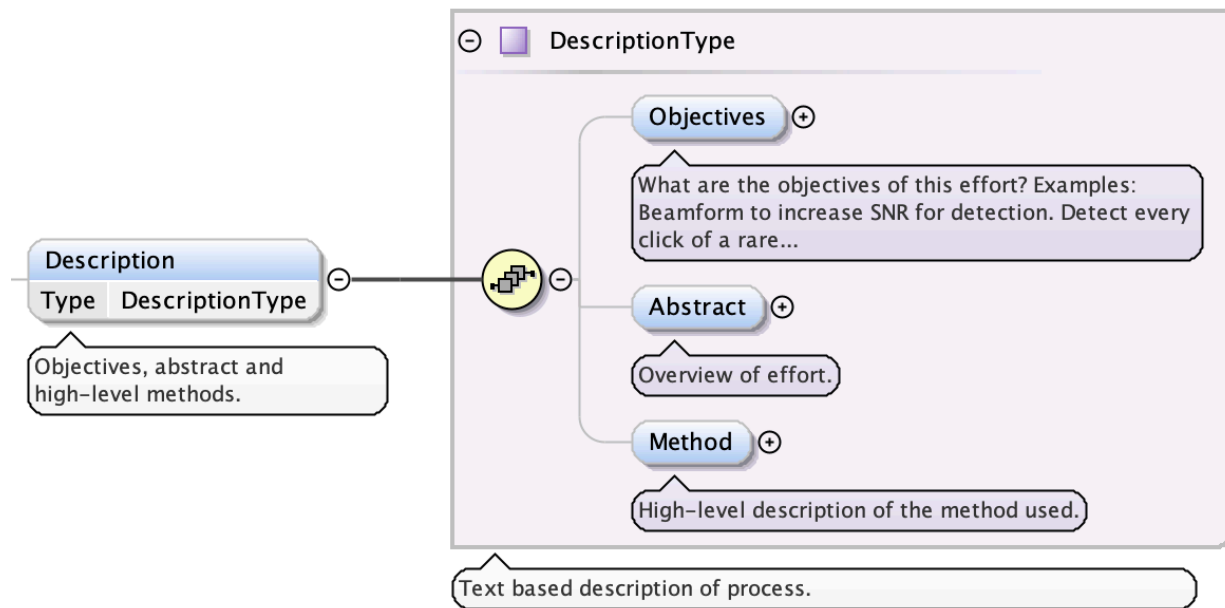


Figure 27 – The Description element with the Detection schema allows for details on the detection objectives and methods.

## 5.4 Ensemble

Ensembles provide a logical grouping of instrument deployments. This is most useful for large aperture localization where separate instruments may be deployed individually, but the data within them are to be treated as if they originated from a single instrument. Ensemble records consist of an ensemble name (**Id**) that is a unique identifier for the ensemble and a sequence of **Unit** elements that describe each instrument in the ensemble (Figure 29). Each **Unit** consists of a unit number (**UnitId**) that is unique within the ensemble and a reference to a deployment document Id field (**DeploymentId**) to determine the deployment associated with the unit.

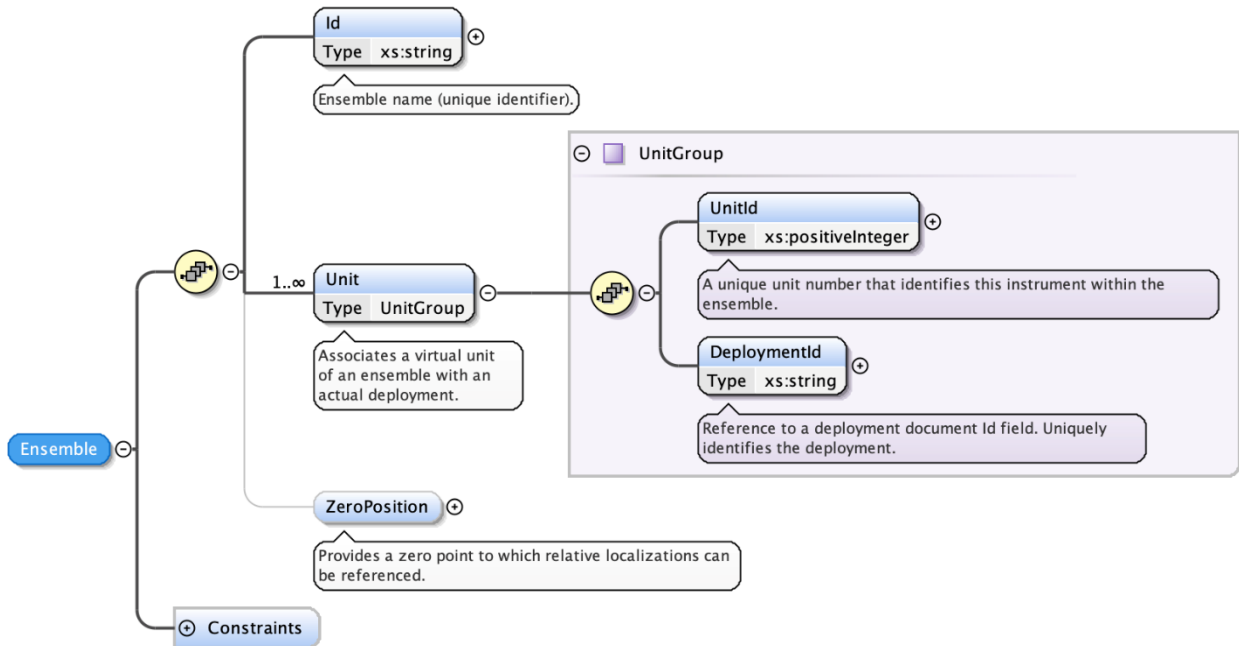


Figure 28 – Ensemble schema used to create logical groupings of instrument deployments. Dark lines indicate required elements; light lines indicate optional elements.

While not a required element, **ZeroPosition** allows for the identification of a point to which relative localizations can be referenced (Figure 30). This element requires values for **Longitude** and **Latitude** and, optionally, instrument depth (**DepthInstrument\_m**).

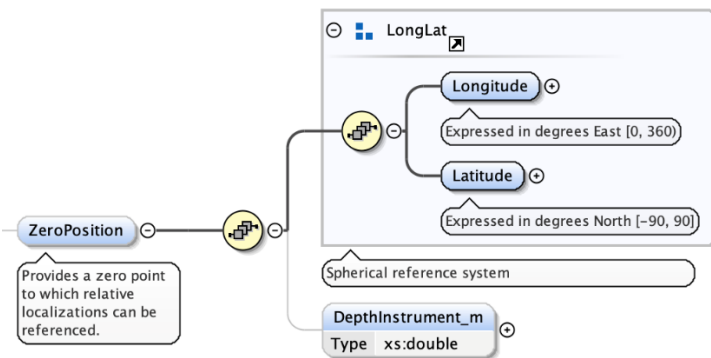


Figure 29 – The ZeroPosition element in the Ensemble schema describes a point that localizations can reference.

## 5.5 Event

Event documents record phenomena or events that are derived from other knowledge sources. Examples include planned Naval exercises, whale watching cruises, pile driving, oil exploration, earthquakes, etc. Information about the type of event and the time period during which it took place are described here. Every Event (Figure 31) should include the name of the event (**Name**), a **Description** that includes the **Type** of event and a contact person or group (**ContactGroup**), the event start time (**Start**), and the event end time (**End**).

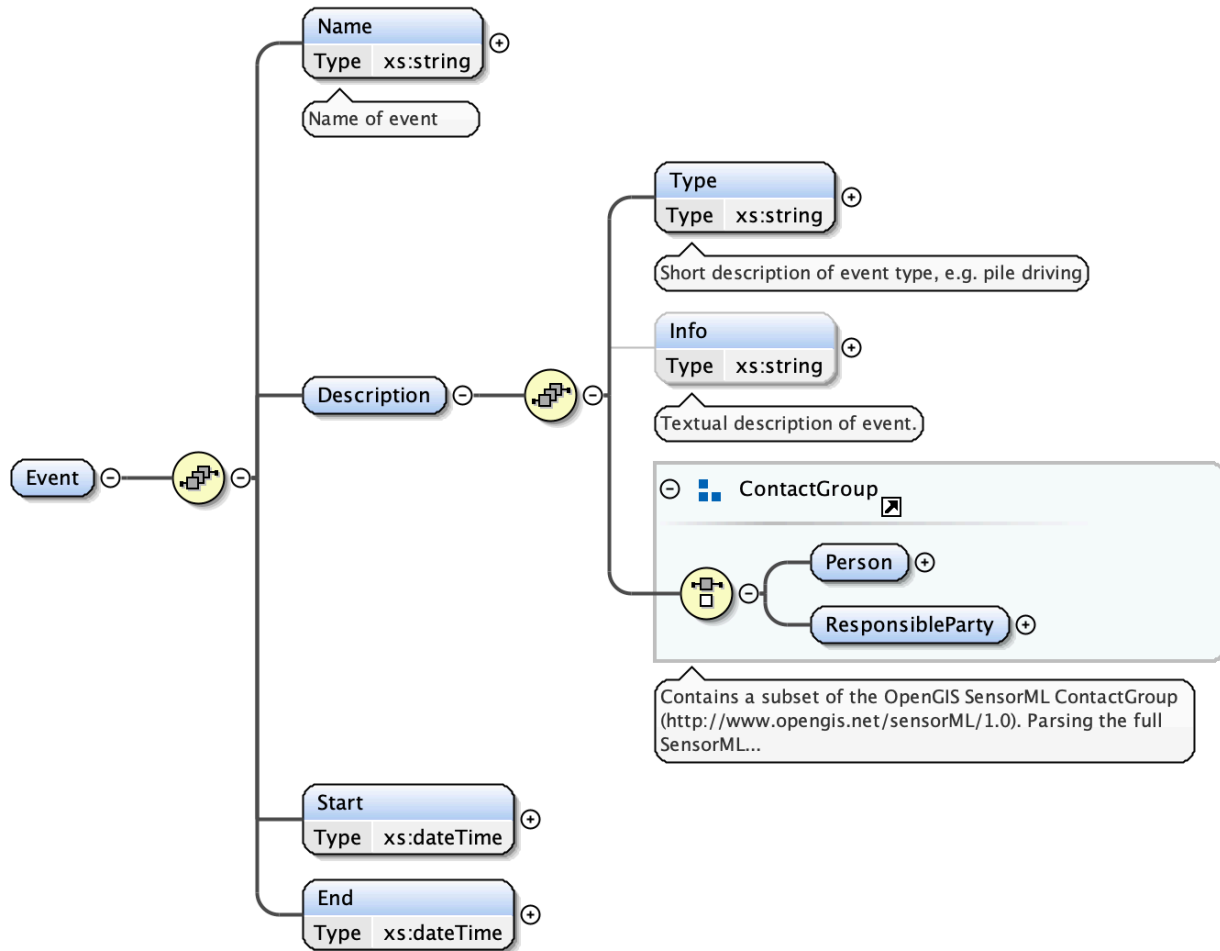


Figure 30 – Event schema used to record phenomena derived from other knowledge sources. Dark lines indicate required elements; light lines indicate optional elements.

## 5.6 Localization

The Localization collection is designed to organize information about localizations derived from multiple hydrophones, whether from a single deployment or an ensemble of multiple instrument deployments.

Every localization document (Figure 32) must include a unique identifier for the document (**Id**), the deployment or ensemble from which localizations were obtained (**DataSource**), a description of the detection algorithm (**Algorithm**), information about the person or organization that generated the metadata (**ResponsibleParty**), the **UserId** of the user that submitted the document, information about the type of localization and when it occurred (**Effort**), and information about individual **Localizations**.

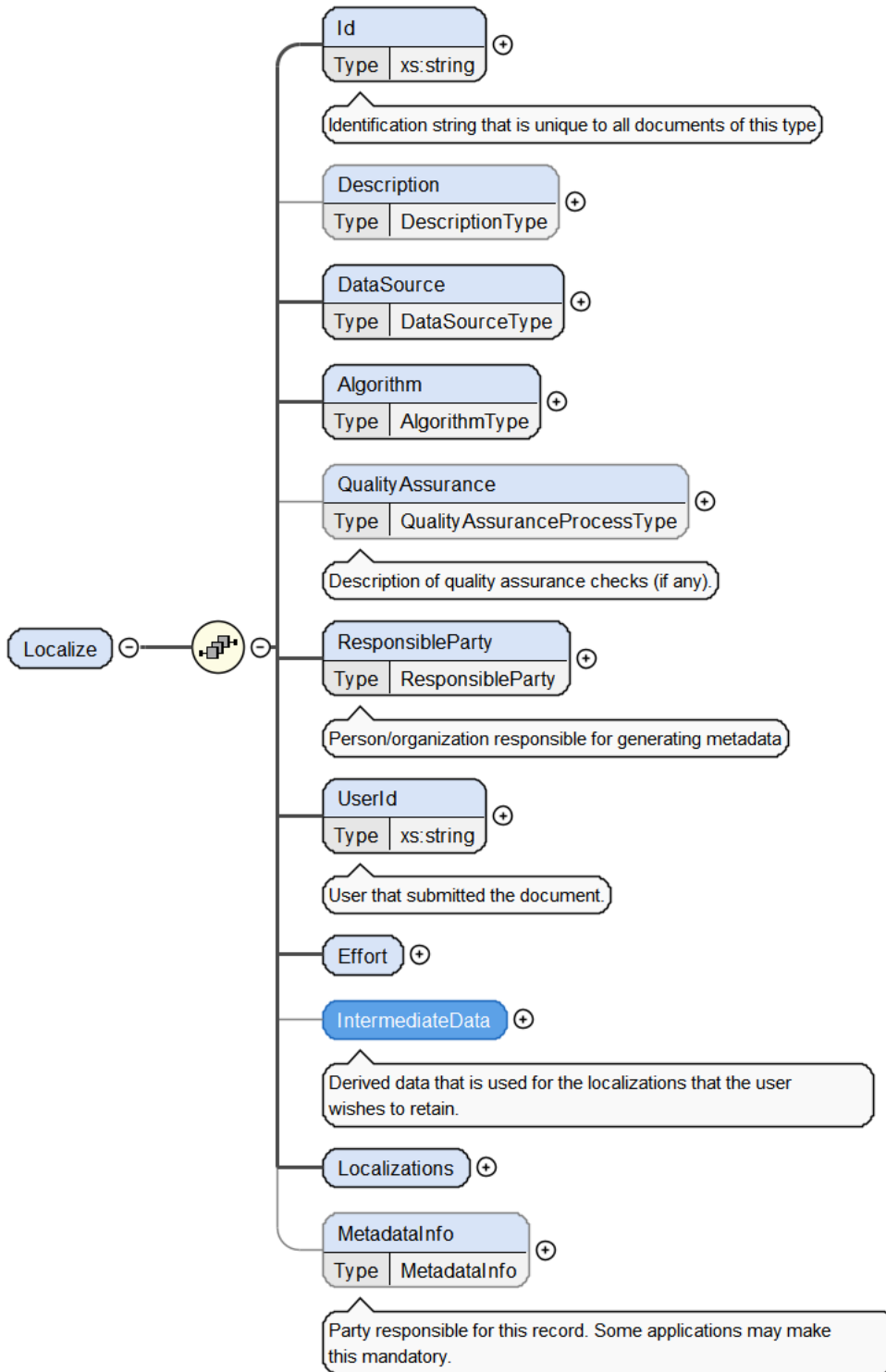


Figure 31 – The Localization schema is used to record localizations of sources from multiple instruments. Dark lines indicate required elements; light lines indicate optional elements.

The **Algorithm** and **ResponsibleParty** elements match those described for the Calibration schema (Figures 18 and 19). **Effort** (Figure 33) includes information about when location information was looked for (**Start** and **End**), the type of localization information produced (**CoordinateSystem**), and the type of

localization (**LocalizationType**), which must be one of: **Bearing**, **Ranging**, **WGS84**, **Cartesian**, or **Track**. **ReferencedDocuments** consists of a list of data products that were used in determining the localizations. For example, if positions are obtained by looking for intersecting bearings from a fast-moving ship, one would reference the bearing localization document. One would need to note that the referenced document was of **Type** Localizations, the **Id** of the document containing the bearings, and an arbitrary **Index** number. When referring back to the individual bearings that produced location information, this **Index** would be used along with **Event** identifiers for each bearing.

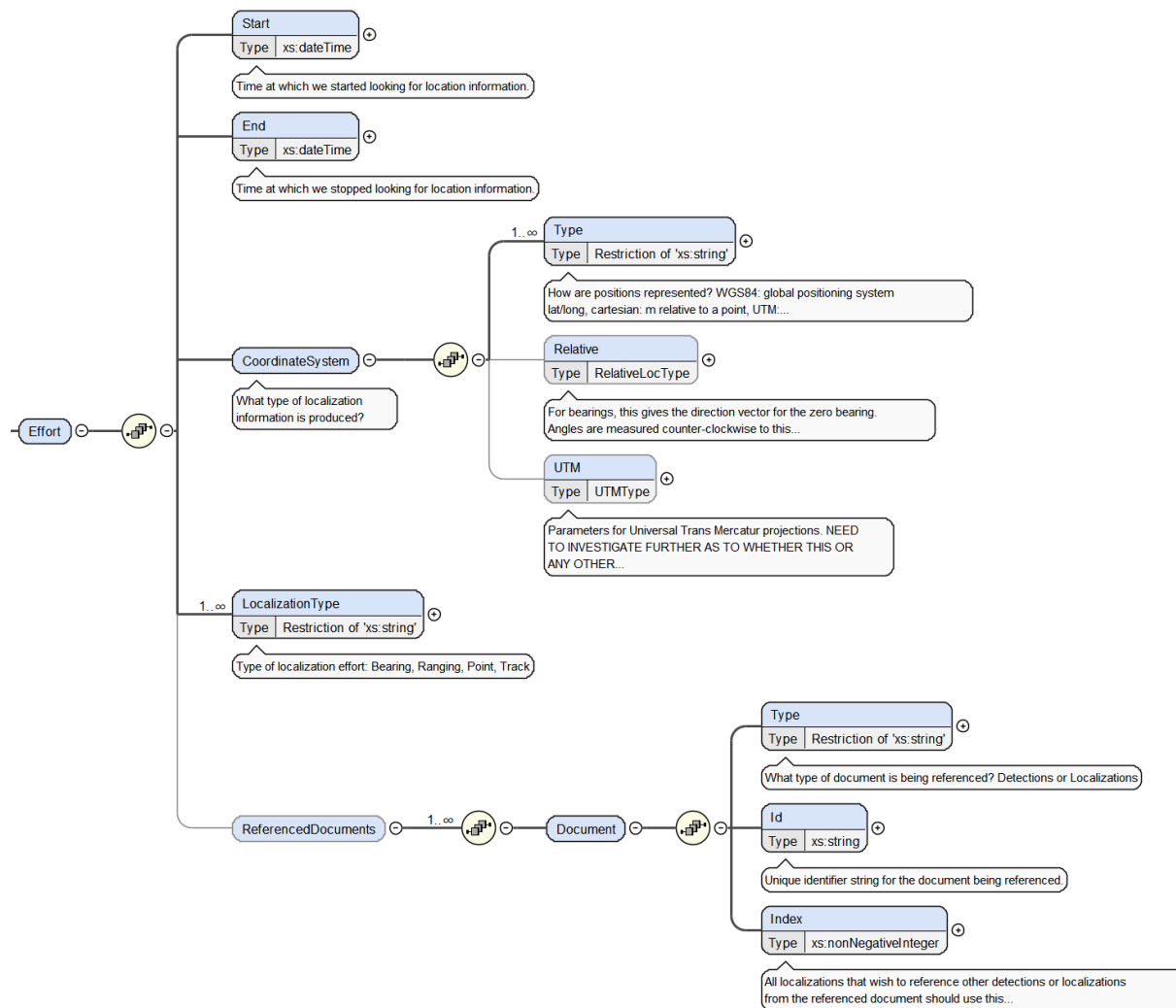


Figure 32 – The Effort element in the Localization schema includes information about when the localization occurred.

The **Localizations** element consists of a series of **Localization** elements (Figure 34). Each **Localization** contains a **Timestamp** indicating when the localization occurred and location information. Location information consists of one of the following elements:

- **Bearing** – A horizontal and optional vertical angle, specified in degrees, as well as left right horizontal ambiguity.

- **Ranging** – Consists of the same information as a bearing but adds a distance to the localized source.
- **WGS84** – Contains a world geodetic system 1984 longitude and latitude reference, along with optional elevation information.
- **Cartesian** – x\_m, y\_m, and optional z\_m distances, specified in meters relative to the zero location.
- **Track** – A series of associated positions and timestamps.

For all types of localizations, standard error may be specified in the same units. The localization can also be optionally associated with an **Event**, a **SpeciesId**, **References** and **QualityAssurance** information can also be provided. When **References** is present, a series of **Reference** elements contain the referenced document **Index** (see Effort above) and an **EventRef** field that specifies a detection or localization **Event** within the referenced document.

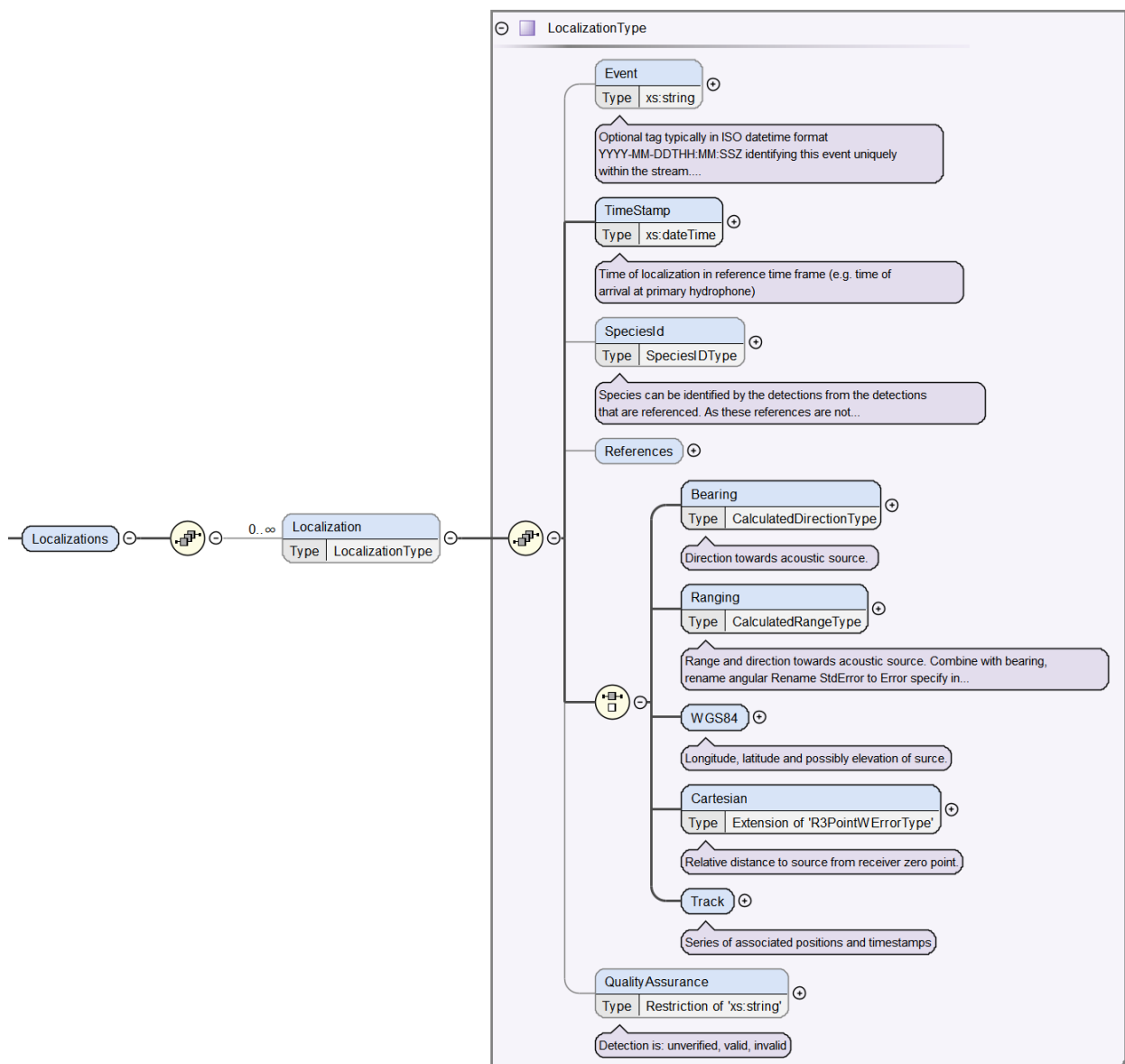


Figure 33 – The Localization element within the Localization schema provides details about individual localizations.

Although not required, the **Description** and **MetadataInfo** elements include information about the **Objectives**, **Abstract**, and **Methods**, and the creation of the localization record, respectively, as described in the Detection and Deployment schemas (Figure 28 and Figure 15). The **QualityAssurance** element allows for a **Description** and **ResponsibleParty**. The **IntermediateData** element can be used to record information about the localizations that the user wishes to retain (Figure 35).

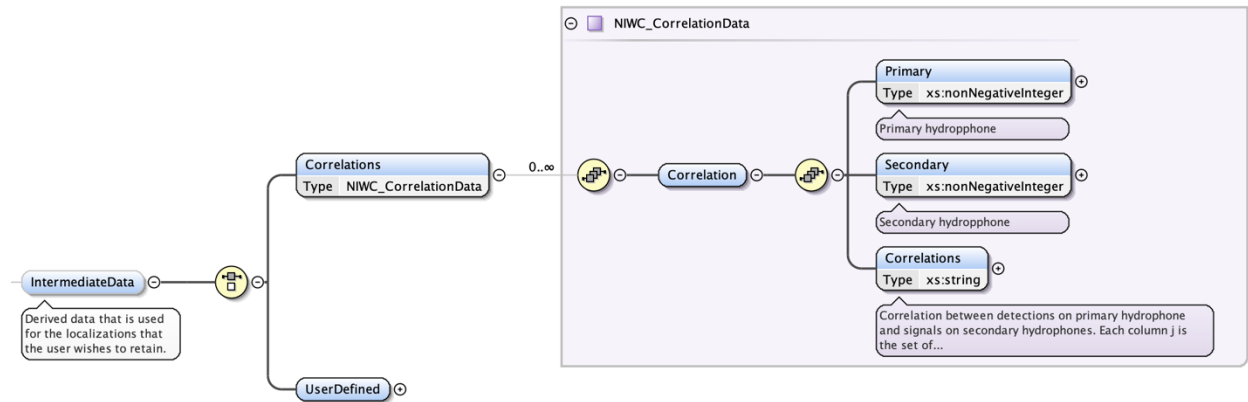


Figure 34 – The **IntermediateData** element within the Localization schema can be used to record information about the localizations.

## 6 Appendix – BatchLogs

Operations that take a long time (e.g., rebuilding part of the database) may be performed as batch operations. When this occurs, a message is given providing a batch log url that can be used to access the log. To see any of the logs that are currently in the system, use a web browser (e.g., Chrome, Internet Explorer) or another tool to visit the Tethys URL with the path BatchLogs. For example, if the server is running on Tethys.me.edu port 9779 (the default port), use <http://Tethys.me.edu:9779/BatchLogs?html=true> . The ?html=true is optional, but tells the browser to format the result for human reading.

For example, one might see the following:

```
<BatchLogs>
  <InProgress>None</InProgress>
  <Complete>
    <Log>2018-05-10T01.06.13Z-a5794435.xml</Log>
    <Log>2018-05-10T01.06.40Z-fd6ffd54.xml</Log>
    <Log>2018-05-10T01.08.11Z-3042c318.xml</Log>
  </Complete>
</BatchLogs>
```

This states that all tasks are complete (InProgress – None) and lists three different batch logs. To see the last one, we visit <http://Tethys.me.edu:9779/BatchLogs/2018-05-10T01.08.11Z-3042c318.xml?html=true>:

```
<UpdateDocuments>
  <Collection>Detections</Collection>
  <ClearBeforeUpdate>False</ClearBeforeUpdate>
  <ReplaceExistingDocuments>False</ReplaceExistingDocuments>
```



```
<SummaryReport>
</SummaryReport>
</UpdateDocuments>
```

This tells us that the batch job updated all Detections, existing documents were not removed first, and any existing documents were not replaced. The `SummaryReport` shows what documents were added and if any documents that we attempted to add failed.

## 7 Appendix: Tethys.xq Module Functions

XQuery allows one to define subroutines that can be called from within an XQuery. Tethys has provided a set of such functions in a module called `Tethys.xq`. Currently, the main purpose of this module is to provide conversion between ITIS taxonomic serial numbers, Latin species names, common species names, and user-defined abbreviations. When converting to vernacular names, a language must be specified. Currently, there are ITIS entries for the languages Afrikaans, Arabic, Chinese, Djuka, Dutch, English, French, Galibi, German, Greek, Hausa, Hawaiian, Hindi, Icelandic, Japanese, Portuguese, and Spanish. However, not all entries are supported equally, and English is by far the most complete with some languages having only a few entries. When a vernacular name does not exist, the Latin name is returned. In rare instances, ITIS contains multiple vernacular names for the same species. In such cases, the first name is returned. As an example, *Orcinus orca* has both Killer Whale and orca as English vernacular names, but the functions described here will always return Killer Whale as the English vernacular name.

In order to use these functions, an XQuery must include the following line of code prior to FLWR query:

```
declare default element namespace "http://tethys.sdsu.edu/XQueryFns";
```

Note that we do not describe all functions in *Tethys.xq*, but rather the ones that Tethys users are most likely to find helpful. Many of the client queries wrap the results of queries in one of these functions to change TSNs into strings.

**AbbreviationMapExists(abbrevmap)** – Determines whether a specified abbreviation map exists.

e.g., `lib:AbbreviationMapExists("NOAA.NMFS.v1")` → true

**abbrev2completename(abbrev, abbrevmap)** – Converts an abbreviation to a Latin name.

The function accepts two strings, an abbreviation and the abbreviation map that is to be used.

e.g., `lib:abbrev2completename("Oo", "NOAA.NMFS.v1")` → *Orcinus orca*

**abbrev2group(abbrev, abbrevmap)** – Finds the group associated with a specific abbreviation.

Group is an attribute that is sometimes used with species to denote additional information. As an example, there are currently a number of echolocation clicks that we believe are produced by beaked whales, but it is unclear which species of beaked whale produced them. In the NOAA.NMFS.v1 abbreviation map, different beaked whale encodings all map to the taxon Hyperoodontidae and are distinguished by a Group attribute in the SpeciesId. Using `abbrev2tsn` for abbreviation "BWC" will map to the TSN for Hyperoodontidae, but `lib:abbrev2group("BWC", "NOAA.NMFS.v1")` will map to a group name. In this case, the group is also called BWC any string would have been possible.

**abbrev2tsn(abbrev, abbrevmap)** – Converts an abbreviation to a (taxonomic serial number) TSN. The function accepts two strings, an abbreviation and the abbreviation map name.  
e.g., `lib:abbrev2tsn("Oo", "NOAA.NMFS.v1")` → 180469.

**completename2tsn(name)** – Translates a Latin name to a TSN.  
e.g., `completename2tsn("Orcinus orca")` → 180469.

**tsn2completename(tsnnodes)** – Converts a TSN to a “complete” (Latin) name.  
tsnnodes must be a list of tsn nodes that each have an integer value, e.g. a SpeciesId from a detection document.

**tsn2vernacular(tsnnodes, language)** – Converts a TSN to a vernacular name.  
tsnnodes must be a list of tsn nodes that each have an integer value, e.g. a SpeciesId from a detection document. The argument language must be a string for a language that is supported by ITIS.

**tsn2abbrev(tsnnodes, abbrevmap)** – Converts a TSN to a user-defined abbreviation.  
tsnnodes must be a list of tsn nodes that each have an integer value, e.g. a SpeciesId from a detection document. Parameter abbrevmap must be the name of an abbreviation map that specifies the set of abbreviations to be used. Like tsn2vernacular, the Latin name is returned if no abbreviation is found.

**vernacular2tsn(CommonName, Language)** – Converts a common name to a TSN.  
e.g., `lib:vernacular2tsn("Killer Whale", "English")` → 180469.

The following functions are intended to be called on result documents and can rewrite an entire XML document. They are used by the client libraries to reformat results in a human-readable format.

**SpeciesIDtsn2name(element)** – Converts XML document from TSN species identifiers to Latin names.

**SpeciesIDtsn2abbrev(element, abbrevmap)** – Converts XML document from TSN species to the specified abbreviation map.

**SpeciesIDtsn2vernacular(element, language)** – Converts XML document from TSN species to vernacular names for the specified ITIS-supported language.

**get-descendants(tsn)** – Given a TSN rank, return XML representing all inferior ranks. Note that the rank itself is not included. e.g., return all blue-whale (180528) subspecies, `lib:get-descendants(180528)`

```
<ranks xmlns="http://tethys.sdsu.edu/schema/1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <rank>
    <rankname>Subspecies</rankname>
    <tsn>612603</tsn>
    <completename>Balaenoptera musculus musculus</completename>
    <vernacular>
      <name language="English">Northern Blue Whale</name>
    </vernacular>
  </rank>
  <rank>
    <rankname>Subspecies</rankname>
    <tsn>612604</tsn>
    <completename>Balaenoptera musculus indica</completename>
    <vernacular>
      <name language="English">Great Indian Rorqual</name>
      <name language="English">Northern Indian Ocean blue whale</name>
    </vernacular>
  </rank>
</ranks>
```

```

    </vernacular>
  </rank>
  <rank>
    <rankname>Subspecies</rankname>
    <tsn>612605</tsn>
    <completename>Balaenoptera musculus brevicauda</completename>
    <vernacular>
      <name language="English">Pygmy Blue Whale</name>
    </vernacular>
  </rank>
  <rank>
    <rankname>Subspecies</rankname>
    <tsn>612606</tsn>
    <completename>Balaenoptera musculus intermedia</completename>
    <vernacular>
      <name language="English">Antarctic blue whale</name>
      <name language="English">Southern Blue Whale</name>
    </vernacular>
  </rank>
</ranks>

```

**get-descendant-tsn-list(tsn)** – Like `get-descendants`, but returns a simple list of TSNs. This function is useful for queries where we wish to query a taxonomic set such as any type of delphinid.

**get-ancestors(tsn)** – Given a TSN rank, return ranks describing the TSN and its chain of taxonomic ancestors. When used with `get-descendants`, a complete set of paths from the root of the taxonomic tree through the specified `tsn` and to all of its descendants can be found.

**get-ancestor-tsn-list(tsn)** – Like `get-ancesotrs`, but returns a simple list of TSNs.

## 8 References

- Giorgini, J. D., Yeomans, D. K., Chamberlin, A. B., Chodas, P. W., Jacobson, R. A., Keeseey, M. S., Lieske, J. H., Ostro, S. J., Standish, E. M. and Wimberly, R. N. (1996). JPL's On-Line Solar System Data Service. *B. Am. Astron. Soc.* 28, 1158.
- Hoffman, C. (2012). How to Create Advanced Firewall Rules in the Windows Firewall. *Advanced Firewall*, vol. 2012.
- Joint W3C/IEFT URI Planning Interest Group. (2002). Uniform Resource Identifiers (URIs), URLs, and Uniform Resource Names (URNs): Clarifications and Recommendations. In *Request for Comments series*, eds. M. Mealling and R. Dennenberg), pp. 11: Internet Engineering Task Force.
- Roch, M.A., Batchelor, H., Baumann-Pickering, S., Berchok, C.L., Cholewiak, D., Fujioka, E., Garland, E.C., Herbert, S., Hildebrand, J.A., Oleson, E.M., Van Parijs, S.M., Risch, D., Sirovic, A. (2016). Management of acoustic metadata for bioacoustics. *Ecol. Inform.* 31, 122-136.
- Soldevilla, M. S., Henderson, E. E., Campbell, G. S., Wiggins, S. M., Hildebrand, J. A. and Roch, M. A. (2008). Classification of Risso's and Pacific white-sided dolphins using spectral properties of echolocation clicks. *J. Acous. Soc. Am.* 124, 609-624.
- Walmsley, P. (2006). XQuery. Farnham, UK: O'Reilly.

## 9 Licenses

Tethys uses components from the following vendors:

## 9.1 Python

The Python programming language is used to bind various server components and is also used in the Python client. Several Python libraries that are used also fall under this license: Python for Windows extensions (pywin32),

1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or Organization ("Licensee") accessing and otherwise using Python 3.9.13 software in source or binary form and its associated documentation.
2. Subject to the terms and conditions of this License Agreement, PSF hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python 3.9.13 alone or in any derivative version, provided, however, that PSF's License Agreement and PSF's notice of copyright, i.e., "Copyright © 2001-2022 Python Software Foundation; All Rights Reserved" are retained in Python 3.9.13 alone or in any derivative version prepared by Licensee.
3. In the event Licensee prepares a derivative work that is based on or incorporates Python 3.9.13 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python 3.9.13.
4. PSF is making Python 3.9.13 available to Licensee on an "AS IS" basis. PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 3.9.13 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 3.9.13 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 3.9.13, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.
6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.
7. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between PSF and Licensee. This License Agreement does not grant permission to use PSF trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.
8. By copying, installing or otherwise using Python 3.9.13, Licensee agrees to be bound by the terms and conditions of this License Agreement

## 9.2 Berkeley DBXML

The server's database store is implemented using the freely redistributable BerkeleyDB XML which is subject to the following terms:

The following is the license that applies to this copy of the Berkeley DB XML software. For a license to use the Berkeley DB XML software under conditions other than those described here, or to purchase support for this software, please contact [berkeleydb-info\\_us@oracle.com](mailto:berkeleydb-info_us@oracle.com).

If you were looking for the license that applies to Berkeley DB, click here.

<http://www.oracle.com/technetwork/products/berkeleydb/downloads/oslicense-093458.html>

If you were looking for the license that applies to Berkeley DB Java Edition, click here.

<http://www.oracle.com/technetwork/products/berkeleydb/downloads/jeoslicense-086837.html>

Berkeley DB XML makes use of Berkeley DB for storage of XML data, indexes, and other information. It also depends on two other 3rd party software packages. These are all included as part of each Berkeley DB XML download package for convenience sake. The first 3rd party package is XQilla (<http://xqilla.sourceforge.net/>). This XQuery implementation was donated to the open source community by Oracle ([http://www.oracle.com/corporate/press/2008\\_mar/xquilla.html](http://www.oracle.com/corporate/press/2008_mar/xquilla.html)) in 2008. It is licensed under the Apache 2.0 (<http://www.apache.org/licenses/LICENSE-2.0.txt>) license. The second 3rd party library used in Berkeley DB XML is Xerces-C (<http://xerces.apache.org/xerces-c/>), it too is licensed under the Apache 2.0 (<http://www.apache.org/licenses/LICENSE-2.0.txt>) license.

The license below applies to the Berkeley DB XML code itself.

The following is the license that applies to this copy of the Berkeley DB XML software. For a license to use the Berkeley DB XML software under conditions other than those described here, or to purchase support for this software, please contact Oracle at [berkeleydb-info\\_us@oracle.com](mailto:berkeleydb-info_us@oracle.com).

=====

GNU AFFERO GENERAL PUBLIC LICENSE  
Version 3, 19 November 2007

Copyright (C) 2007 Free Software Foundation, Inc. <<http://fsf.org/>>  
Everyone is permitted to copy and distribute verbatim copies  
of this license document, but changing it is not allowed.

Preamble

The GNU Affero General Public License is a free, copyleft license for software and other kinds of works, specifically designed to ensure

cooperation with the community in the case of network server software.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, our General Public Licenses are intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

Developers that use our General Public Licenses protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License which gives you legal permission to copy, distribute and/or modify the software.

A secondary benefit of defending all users' freedom is that improvements made in alternate versions of the program, if they receive widespread use, become available for other developers to incorporate. Many developers of free software are heartened and encouraged by the resulting cooperation. However, in the case of software used on network servers, this result may fail to come about. The GNU General Public License permits making a modified version and letting the public access it on a server without ever releasing its source code to the public.

The GNU Affero General Public License is designed specifically to ensure that, in such cases, the modified source code becomes available to the community. It requires the operator of a network server to provide the source code of the modified version running there to the users of that server. Therefore, public use of a modified version, on a publicly accessible server, gives the public access to the source code of the modified version.

An older license, called the Affero General Public License and published by Affero, was designed to accomplish similar goals. This is a different license, not a version of the Affero GPL, but Affero has released a new version of the Affero GPL which permits relicensing under this license.

The precise terms and conditions for copying, distribution and modification follow.

## TERMS AND CONDITIONS

## 0. Definitions.

"This License" refers to version 3 of the GNU Affero General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

## 1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of

interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

## 2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you



with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

### 3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

### 4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

### 5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

## 6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical

medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install

and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

## 7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

- a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
- c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
- d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
- e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

## 8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

#### 9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

#### 10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered

work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

#### 11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so

available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

## 12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.



### 13. Remote Network Interaction; Use with the GNU General Public License.

Notwithstanding any other provision of this License, if you modify the Program, your modified version must prominently offer all users interacting with it remotely through a computer network (if your version supports such interaction) an opportunity to receive the Corresponding Source of your version by providing access to the Corresponding Source from a network server at no charge, through some standard or customary means of facilitating copying of software. This Corresponding Source shall include the Corresponding Source for any work covered by version 3 of the GNU General Public License that is incorporated pursuant to the following paragraph.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the work with which it is combined will remain governed by version 3 of the GNU General Public License.

### 14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU Affero General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU Affero General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU Affero General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU Affero General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

### 15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

#### 16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

#### 17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

### END OF TERMS AND CONDITIONS

#### How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>  
Copyright (C) <year> <name of author>
```

```
This program is free software: you can redistribute it and/or modify
```

it under the terms of the GNU Affero General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License along with this program. If not, see <<http://www.gnu.org/licenses/>>.

Also add information on how to contact you by electronic and paper mail.

If your software can interact with users remotely through a computer network, you should also make sure that it provides a way for users to get its source. For example, if your program is a web application, its interface could display a "Source" link that leads users to an archive of the code. There are many ways you could offer source, and different solutions will be better for different programs; see section 13 for the specific requirements.

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU AGPL, see <<http://www.gnu.org/licenses/>>.

#### ADDITIONAL THIRD PARTY NOTICES:

-----

Zlib Data Compression Library 1.2.11

Copyright (C) 1995-2017 Jean-loup Gailly and Mark Adler

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.

3. This notice may not be removed or altered from any source distribution.

Jean-loup Gailly      Mark Adler  
jloup@gzip.org      madler@alumni.caltech.edu

=====  
xerces-C++ 3.1.4

This product includes software developed by  
the Apache Software Foundation (<http://www.apache.org>)

Portions of this software were originally based on the following:  
- software copyright (c) 1999, IBM Corporation., <http://www.ibm.com>.

=====  
Xqilla 2.3.3

This product includes software developed by  
the Apache Software Foundation (<http://www.apache.org>)

=====

The following applies to all products licensed under the Apache 2.0 License:

You may not use the identified files except in compliance with the Apache License, Version 2.0 (the "License.")

You may obtain a copy of the License at  
<http://www.apache.org/licenses/LICENSE-2.0>. A copy of the license is also reproduced below.

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and limitations under the License.

Apache License  
Version 2.0, January 2004  
<http://www.apache.org/licenses/>

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

## 1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent

to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.
4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
  - (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and
  - (b) You must cause any modified files to carry prominent notices stating that You changed the files; and
  - (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work,

excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any

risks associated with Your exercise of permissions under this License.

8. **Limitation of Liability.** In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
9. **Accepting Warranty or Additional Liability.** While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

#### END OF TERMS AND CONDITIONS

#### APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");  
you may not use this file except in compliance with the License.  
You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software  
distributed under the License is distributed on an "AS IS" BASIS,



WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
See the License for the specific language governing permissions and  
limitations under the License.

### 9.3 CherryPy Object oriented web framework

[www.cherrypy.org](http://www.cherrypy.org) - The Tethys server uses CherryPy to implement its transport layer.

Copyright © 2004-2019, CherryPy Team ([team@cherrypy.dev](mailto:team@cherrypy.dev))

All rights reserved.

---

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of CherryPy nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

## 9.4 Libraries using the MIT License

The following libraries use the MIT License shown below:

1. py-dom-xdom: XML XPath queries for Python (c) 2009, used in Python client:  
<https://code.google.com/p/py-dom-xpath/>
2. pyodbc – Open database connectivity library (c) 2012, used in server:  
<https://code.google.com/p/pyodbc/>

The MIT License (MIT)

Copyright (c) <year> <copyright holders>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.