Java Client

TABLE OF CONTENTS

1 INTRODUCTION

The JavaClient is a java archive (JAR) library that can be imported into Java programs or any programming environment that supports a Java interface (e.g., R, MATLAB) and provides basic functionality to interface with a Tethys server. A complete list of the functionality of the JavaClient library may be found in the Java application programming interface documentation that is in JavaClient/apidocs.

This manual only documents the basic functionality of the Queries class which can be used to interact with a Tethys server. The apidocs directory within the JavaClient directory documents the entire functionality including many functions that users are unlikely to need or use.

2 CREATING A QUERIES OBJECT

If you are using the Tethys JavaClient from another language, there may be functions provided in that language that will take care of the creation for you (e.g., dbInit in MATLAB). If that is not the case, you will need to ensure that program using the JavaClient has the JAR on its JAR load path.

Once this is done, one should be able to import two classes from dbxml:

```
import dbmxl.JerseyClient;

import dbxml.Queries;
```

The Queries constructor takes a single argument, an instance of a client object. The client object is first created with the URL of the Tethys server. For example, if the server is running on port 9779 (the default port) on machine grampus.local without encryption enabled one would use the following:

```
JerseyClient client = new JerseyClient("http://grampus.local:9779");

/* We'll use q here to make examples shorter, generally we might

 * want to use a more descriptive name, e.g., tethysQueryHandler

Queries q = new Queries(client) ;
```

3 USING THE QUERIES OBJECT

Once the Queries object is created, there are a number of methods that we can use. In these examples, we assume that the Queries object is called q.

3.1 SERVER MAINTENANCE

q.ping() – Returns string alive if the server is reachable.

q.getURL() and q.getURLString() – Return the server http/https address either as a URL object or a string.

3.2 Specifying Species Names

Internally, whenever a species name needs to be stored, Tethys uses the taxonomic serial number (TSN) provided by the integrated taxonomic information system (ITIS). Storing TSNs is helpful for times when common or Latin names are changed such as the recent trend to use goose-beaked whale instead of Cuvier's beaked whale. Serial numbers are not helpful to most humans, and while Tethys can use TSNs for input and output, the default is to map these to Latin names. Users can also define custom abbreviation lists that are named, such as NOAA.NMFS.v3 where specific species are given codings that are useful for an organization such as Tt as shorthand for *Tursiops truncatus*.

The following commands enable one to examine or change the current behavior.

q.getSpeciesIdInput() – Reports the current input system. Will be one of the following:

- Latin
- TSN
- Abbreviation name
- Vernacular name (e.g., English, Spanish, French, etc. Note that the vernacular language name is in English: Spanish, not español and that vernacular entries are not available for all species and if the specified vernacular does not exist, Latin will be used).

q.getSpeciesOutput() – Reports how species will be coded in the results of queries. Will be Latin, TSN, abbreviation or vernacular name.

q.setSpeciesIdInput(type, value) – Indicates how queries will encode species.

- type – Must be one of: TSN, Latin, Vernacular, Abbrev
- value – Specifies the vernacular or abbreviation name and is only required when type is Vernauclar or Abbrev.

q.setSpeciesIdOutput(type, value) – Behaves identically to q.setSpeciesIdInput, but controls how query results will be formatted.

3.3 Querying documents

The queries object provides several different ways to query the server.

3.3.1 XQuery

All of these methods assume that the user has a query written in the XQuery query language. Other methods enable querying without knowing XQuery and are more likely to be useful to users.

q.Query(query) – Given an XQuery string, run the query and return results as a string.

q.QueryReturnDoc(query) – Given an XQuery string, run the query and return results as a document object model (org.w3c.Dom.Document).

q.QueryTethys(query) – Given an XQuery string, prepend the following lines which set a default namespace and enable Tethys XQuery library function calls:

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

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

3.3.2 JSON XQuery

These methods provide the ability to specify collection criteria and elements to be returned as a java script object notation (JSON) string. We do not recommend this interface for casual users, but one can use the drag and drop query generator in the advanced page of the web client to generate JSON query strings if so desired. Details on the JSON specification are given in the Tethys Web Services Interface documentation.

q.QueryJSON(jsonSpec, plan) – jsonSpec must be a JSON specification of the query. plan defaults to zero and is interpreted as follows:

0 – execute the query and return an XML result string

1 – Return the query plan generated by Oracle Berkeley dbXML (not useful for most people)

2 – Return the XQuery that corresponds to the JSON specification.

3.3.3 Simple Declarative Query Language (SDQL)

The simple declarative query language is a simple set of commands that can be used to execute queries without understanding XQuery. In general, one specifies the criteria for selection, and the data that are to be returned. For example, to query deployments that are deeper than 1000 m with sample rates > 200 kHz, one could write the following:

```
DeploymentDetails/ElevationInstrument_m < -1000 and

SampleRate_kHz >= 200
```

Within the set of elements that are used to describe deployments, SampleRate_kHz only occurs once within the deployments schema (although it may be repeated for different channels in a document), consequently we did not need to write the full path: Deployment/SamplingDetails/Channel/Sampling/Regimen/SampleRate_kHz. In contrast, ElevationInstrument_m occurs in both Deployment/DeploymentDetails and Deployment/RecoveryDetails, so we needed to indicate which one we wanted. If we had not, an error would have been generated as SDQL would have been unable to determine which one we wanted.

SDQL also provides a way to specify what should be returned. A return keyword can be added indicating what is desired:

```
q.QuerySimple("DeploymentDetails/ElevationInstrument_m < -1000
and SampleRate_kHz >= 200 return Id,
DeploymentDetails/ElevationInstrument_m", "Deployment")
```

When return is omitted, a default set of return values will be used.

The following methods are supported that all take a query string. Ambiguous element names that match more than one element in the schema may sometimes be resolved based on the type of query. For example, when calling q.getDetections, we assume that users are querying for OnEffort detections unless they specifically use OffEffort in the query.

q.getDetections(query, plan) – Retrieve detections based on the query.

query – Query, e.g. retrieve blue whale detections associated with Project SOCAL

```
Project = "SOCAL" and SpeciesId = "Balaenoptera musculus"
```

plan – Optional, one of the following:

0 – execute the query and return an XML result string (default)

1 – Return the query plan generated by Oracle Berkeley dbXML (not useful for most people)

2 – Return the XQuery that corresponds to the SDQL specification.

3 – Return the JSON query that corresponds to the SDQL specification.

Other methods are similar and have the same interpretation for query and plan. Only the set of default return items varies between these.

q.getDetectionEffort(query, plan) – Retrieve effort that matches query criteria.

q.getDeployments(query, plan) – Retrieve deployments based on criteria.

q.getLocalizations(query, plan) – Retrieve localizations based on criteria.

q.getLocalizationEffort(query, plan) – Retrieve effort based on criteria.

q.getCalibration(query, plan) – Retrieve calibration effort based on criteria.

3.4 REMOVING DOCUMENTS

q.removeDocument(collection, docname) – Remove specified document from collection. Note that while the docname is usually the same as the document Id for documents with Ids, this is not

mandatory. The web client can provide a list of documents for any collection, and it reports the docnames in the list of documents.

3.5 UTILITY FUNCTIONS

Functions exist to translate between the taxonomic serial numbers (TSNs, https://itis.gov/) stored in Tethys and textual represtions of species such as Latin, vernacular names (e.g., blue whale, Ballena azul) or user specified abbreviations (e.g., Lo for *Lagenorhynchus obliquidens* in the SIO.SWAL.v1 species abbreviation map).

Note that ITIS does not have vernacular entries for many obscure species and frequently only has vernacular entries in English. When a vernacular entry is not present, the Latin name is substituted.

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

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

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

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

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

Getter functions provide the ability to retrieve current settings:

- `q.getSpeciesIdInput()`
- `q.getSpeciesIdOutput()`
- `q.getSpeciesIdInputOutput()  (returns a single string suitable for human interpretation)`