TextGridRep: Manual Architecture, Installation, Expandability (R 1.2.2)

Version 28.11.2012
Work Package 1
Responsible Partner SUB Göttingen

TextGrid

Virtual Research Environment for the Humanities

Ein Projekt im D-GRID

GEFÖRDERT VOM Bundesministerium für Bildung und Forschung

Project: TextGrid – Virtual research Environment for the Humanities
Funded by the German Federal Ministry of Education and Research (BMBF) by Agreement: 01UG0901A
Project Duration: June 2009 - May 2012

Document Status: Final
Distribution: Public
Authors:

Stefan E. Funk (DAASI International)
Martin Haase (DAASI International)
Sibylle Söring (SUB Göttingen)
Ubbo Veentjer (SUB Göttingen)
Thorsten Vitt (Universität Würzburg)

Contents:

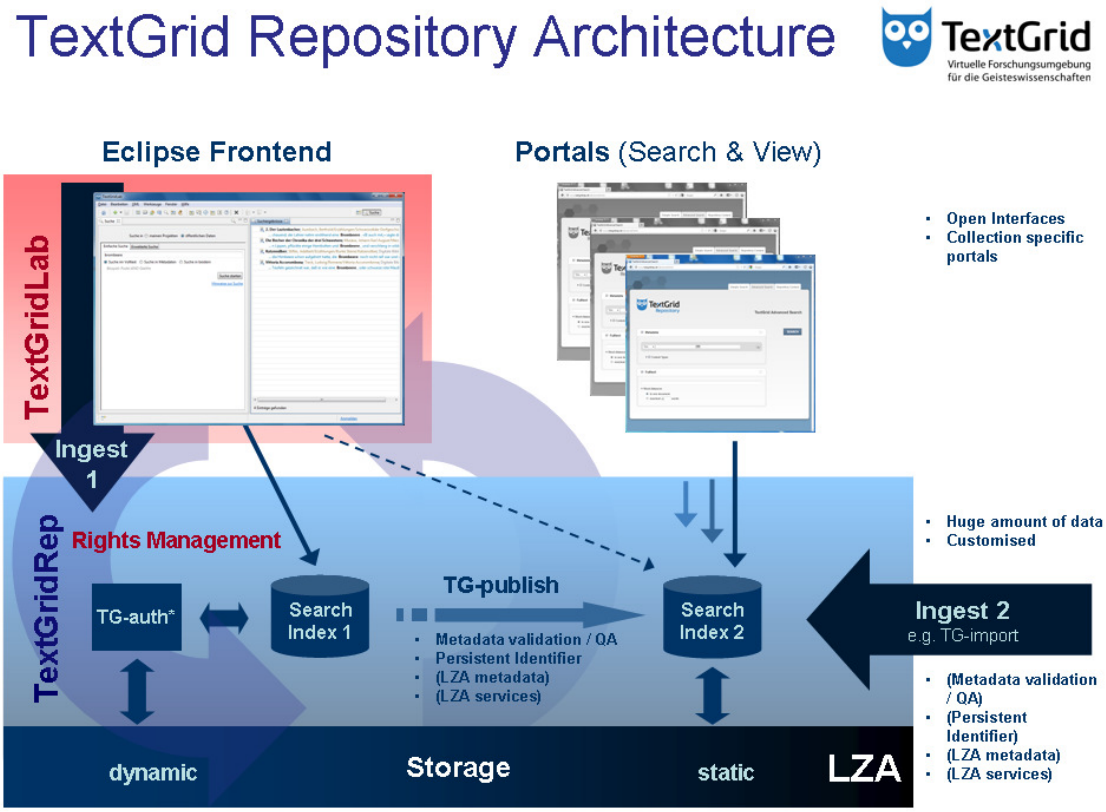
1. Introduction

The TextGrid Repository figures as one of the two core components of the Virtual Research Environment (VRE) TextGrid. In addition to the TextGrid Laboratory (TextGridLab) consisting of the various tools and services, the TextGrid Repository (TextGridRep) provides an academic long-term archive for research data of the text-based humanities, the long-term availability and accessibility of which as well as their optimal internal and external integration need to be guaranteed. Moreover, humanistic data or content archived in the TextGridRep is both quotable and searchable for the public. As such, the TextGridRep serves as an archive that enables the long-term storage and re-use of research data: A Persistent Identifier (PID) service assigns a PID, a unique code consisting of a sequence of numbers and letters, to any item published in the Repository, guaranteeing quotability.

This report describes the architecture, installation and some aspects of the expandibility of the TextGridRep, based on its main middleware components TG-auth*, TG-search and TG-crud.

2. The TextGrid Architecture and Installation

The TextGrid Repository is built on a reliable and powerful infrastructure, providing a middleware consisting of the three utilities TG-auth*, TG-search and TG-crud, implemented as web services.



The TextGrid Repository Architecture.

2.1. TG-auth*

TG-auth* covers two aspects. With "N" replacing the asterisk, it provides for autheNtication of users in the TextGrid environment. With "Z", it serves as an authoriZation engine. As already mentioned, for authorization, TextGrid uses a role-based access control solution called OpenRBAC where permissions are stored in an LDAP database. With logging in to TextGrid, a session ID is generated, which is passed around between the utilities and services, to be used to check permissions with TG-auth*.

The TG-auth* system consists of two main components:

- **openRBAC**, a system to maintain, modify, and enforce authorisation policies using the Role-Based Access Control framework. See http://www.openrbac.de/; however, the basic software has been extensively customised for use with TextGrid

- **WebAuthN**, a system offering authentication functionalities, both direct using a community-managed user directory and the Shibboleth-based DFN-AAI. WebAuthN is embedded in TextGridLab offering a Login Screen and registers the user in RBAC.

Some minor components interact with tg-auth*:

- **PWchange**, a Web application allowing for setting a new password given the initial password is still known
- **PWreset**, a Web application allowing for setting a new password given the initial password is no longer known

2.1.1. Technical Information

- **openRBAC**
    - Implementation: PHP, consisting of
        - openRBAC core: RBAC implementation backed up by an LDAP directory, e.g. openLDAP
        - openRBAC Web Service layer: for accessing openRBAC functions via SOAP
        - tgextra (also a SOAP Web Service): additional functions implemented for TextGrid needs, either aggregating basic RBAC functions or introducing unrelated functions that leverage the underlying LDAP server as storage
    - Storage: an OpenLDAP server
        - two additional schemas: for RBAC core and for TextGrid-specific attributes
        - Branches:
            - ou=people for users
            - ou=roles for the roles users can activate. TextGrid projects are treated like roles, with sub-roles for the actual roles visible in the TextGridLab. e.g. Administrator or Editor
            - ou=resources for the TextGridObjects and their role-right assignments
            - ou=sessions for the Session IDs that users have in the TextGridLab and the roles they activated in their sessions

- **WebAuthN**
    - Implementation: PHP
        - Dual Login on the first page:
            - direct authentication in the community LDAP server or via
            - Shibboleth Login with DFN-AAI-Basic
        - Both Login methods populate the Server variable $REMOTE_USER
        - In Login Mode, the following steps will be effected:

                - authentication
                - registration of a user session with activation of all available roles in RBAC
                - check if user has filled out all required personal information and accepted the Terms of use
                - exposure of the newly assigned Session ID for use in further activities with the TextGridLab and the TG-Utilities
            - In User Details mode (no authentication, just read and modify user's attributes), only c.) happens
            - One WebAuthN installation with one community LDAP server can interact with multiple RBAC instances
            - HTTP GET or POST arguments for TextGrid-WebAuth.php:
                - authZinstance - string identifying the RBAC instance to be used. Always mandatory.
                - loginname and password - for authentication at community LDAP. Only in Login mode and with HTTP POST.
                - Sid - Session ID known from some earlier authentication. Required for User Details mode
                - ePPN - User ID of the user. Required in User Details mode.
        - TextGrid-WebAuth.php is being called from WebAuthN2.php, which presents both the community login form and the Shibboleth Login Button
        - For Shibboleth login, the Shibboleth Service Provider (Apache module) guarantees the provision of a correct User ID delivered from some home organisation.

- **PWchange**
    - PHP Web application
    - Authenticates and changes passwords against an LDAP direcory (community LDAP server)
    - Source currently not in SVN, but available upon request

- **PWreset**
    - Perl Web application
    - provides links for verification of a user's email adress
    - to be used with the system's Web browser, not within TextGridLab, as cookies are used to remember a user

2.1.2. URLs

Subversion Repository

openRBAC: https://develop.sub.uni-goettingen.de/repos/textgrid/trunk/middleware/tgauth/info.textgrid.middleware.tgauth.rbac

WebAuthN: https://develop.sub.uni-goettingen.de/repos/textgrid/trunk/middleware/tgauth/info.textgrid.middleware.tgauth.webauth

PWreset: https://develop.sub.uni-goettingen.de/repos/textgrid/trunk/middleware/tgauth/info.textgrid.middleware.tgauth.passwordReset

WSDL

OpenRBAC SOAP WSDL locations on the productive TextGridRep TG-auth* server:

- Most relevant for Lab/User interaction:
  https://textgridlab.org/1.0/tgauth/wsdl/tgextra.wsdl
- Relevant for Server access: https://textgridlab.org/1.0/tgauth/wsdl/tgextra-crud.wsdl
- Administrative functions:
    - https://textgridlab.org/1.0/tgauth/wsdl/tgadministration.wsdl
    - https://textgridlab.org/1.0/tgauth/wsdl/tgreview.wsdl
    - https://textgridlab.org/1.0/tgauth/wsdl/tgsystem.wsdl

Web Applications

End points of the productive TextGridRep:

WebAuthN (Login mode):
https://textgridlab.org/1.0/WebAuthN/WebAuthN2.php?authZinstance=textgrid-ws3.sub.uni-goettingen.de

WebAuthN (User Details mode):

https://textgridlab.org/1.0/WebAuthN/TextGrid-WebAuth.php?authZinstance=textgrid-ws3.sub.uni-goettingen.de (append "&Sid=XXXX&ePPN=YYY@ZZZ", see above)

PWchange: https://textgridlab.org/1.0/PWchange/index.php

PWreset: https://textgridlab.org/1.0/pwReset.pl

2.2. TG-crud

TG-crud is a web service to create, retrieve, update and delete TextGrid objects. It is the in- terface to storing information to the grid environment, the search indices and the role based access control system (RBAC) using TG-auth* (see above). TG-crud also checks access permissions and ensures that the TextGrid repository stays consistent. Furthermore, it uses the Adaptor Manager to convert XML documents into the TextGrid baseline encoding, which also are stored in the XML database for efficient structural search. Additionally, the Adaptor Manager is responsible for extracting relation information from metadata, TEI files and the generated baseline-encoded files (such as contained links to other TextGrid objects and XML schema references) and storing them to the RDF triple store (a Sesame database). As may be easily comprehensible, TG-crud bundles most of the application logic of TextGrid and its middleware. So the TG-crud service is responsible for creating, retrieving, updating, and deleting TextGrid resources, i.e. TextGrid objects including TextGrid metadata. A TextGrid object or resource consists of a data file, e.g. a TEI XML file or an image, and a metadata file conforming to the TextGrid Metadata Schema. Furthermore, it generates TextGrid URIs (see also *TG-noid*), and can be used to *lock* and *unlock* objects.

Available methods are described below. TG-crud provides its methods via SOAP and via REST.

2.2.1. Technical Information

The class *TGCrudServiceImpl* implements the service interface created by the CXF webservice implementation. This service is used to ingest, access, and delete TextGrid objects – which actually have two components: the metadata file and the data file. It is meant to be used by the TextGridLab software and by other TextGrid Services that are used by the TextGridLab. All service methods can be used with the Java client classes provided in the *middleware.tgcrud.clients.tgcrudclient* folder, using the JAXB data binding, or using every client to be built to serve the TG-crud's WSDL file, or using the *tgcrud-client* Maven module provided with the parent TG-crud module.

2.2.2. Subversion Repository

Productive TextGrid 2.0 tag:
https://develop.sub.uni-goettingen.de/repos/textgrid/tags/tgcrud/middleware.tgcrud-2012-05-16-v3.0.0-SNAPSHOT-TG2.0/

TG-crud SVN repository (trunk), containing the current develop version that mainly is operable: https://develop.sub.uni-goettingen.de/repos/textgrid/trunk/middleware/tgcrud/

2.2.3. Version

This documentation is valid for TG-crud Service Version

3.0.0-SNAPSHOT-2012-04-27T11:42-'TextGrid 2.0'

Check the current productive TG-crud 2.0 version:
http://textgridlab.org/1.0/tgcrud/rest/version

Latest development version: http://textgridlab.org/dev/tgcrud/rest/version

2.2.4. Further Documentation

Aditional information is available from the current productive TG-crud 2.0 WSDL file:
https://textgridlab.org/1.0/tgcrud/TGCrudService?wsdl
alternatively:
/middleware.tgcrud-2012-05-16-v3.0.0-SNAPSHOT-TG2.0/services/tgcrudservice/tgcrud-base/src/main/webapp/WEB-INF/TGCrudService.wsdl (SVN file)

2.2.5. Installation

Getting and deploying the WAR file

Building from SVN

TG-crud may be checked from the SVN (trunk or tagged V2.0 version); then use Maven to build the TG-crud service WAR file. First go to your favorite home directory, and create a folder for the code:

```
mkdir src/
cd src
```

Then checkout the TG-crud service code:

```
svn co https://develop.sub.uni-goettingen.de/repos/textgrid/tags/tgcrud/
middleware.tgcrud-2012-05-16-v3.0.0-SNAPSHOT-TG2.0/services/tgcrudservice/
```

Build the package:

```
cd tgcrudservice
mvn package
```

You will get a TG-crud WAR file in the folder *src/tgcrudservice/tgcrud-base/target*.

Using Pre-Build WAR File

Alternatively, use the WAR file from our Archiva repository server: /3.0.0-SNAPSHOT/tgcrud-base-3.0.0-20120516.114410-9.war

For simplification, it may be renamed as *tgcrud.war*.

Deploying the TG-crud Service

The WAR file *tgcrud.war* is deployed into your favorite Application Server, e.g. Apache Tomcat.

2.2.6. Configuration

Config File Location

To create the default config folder, the owner should be user *root*:

```
mkdir /etc/textgrid/tgcrud/conf
chmod 755 /etc/textgrid/tgcrud/conf
```

Move the two config files *tgcrud.log4j* and *tgcrud.properties* to this folder. For more information on the config files please consult the documentation inside, and edit both files accordingly.

Log File Location

Create the log file location:

```
mkdir /var/log/textgrid/tgcrud
sudo chown -R textgrid /var/log/textgrid/tgcrud/
```

The user should be set to the Tomcat's user, so it can log its information.

CXF Spring Configuration

CXF calls the service's constructor just once at service start time. We call *init()* with every method call. The configuration file location is set in the *beans.xml* configuration file as constructor argument, as well as the service's endpoint location in the WSDL file. The TG-crud's config files (*tgcrud.properties* and *tgcrud.log4j*) will be instantly used after editing with the next TG-crud service call.

2.2.7. SOAP and REST API

Almost all TG-crud calls require an RBAC Session ID and an (optional) log parameter. For TextGrid or Shibboleth account owners, this ID is available from the TG-auth* Web-Auth service. All of the methods can also be called via REST (please see below).

Parameter Description

- **sessionId**
  The RBAC session ID is needed with nearly every method call; it is used to authenticate the user to the TG-crud and its underlying databases.
- **logParameter**
  If a log parameter is given, the TG-crud will log to the TextGrid log service shown from within the TG-lab. All logs up to log level INFO are logged.
- **uri**
  If no URI is given, the TG-crud will create a new one. If an URI is provided, it needs to be one the TG-crud created using the #GETURI method, and not hve been in use earlier. This parameter will mostly be used if objects shall be automatically imported and prepared, e.g. using internal references, link rewriting, etc.
  For other methods such as reading and deleting, the URI is mandatory.
- **createRevision**
  Set to TRUE if a new revision of the given object shall be created (baseUri is mandatory then), the revision number will be increased, whereas the base URI remains the same. Set to FALSE (default) if a new object shall be created using a new URI.
- **projectId**
  Provide the Project ID of the project the new object shall be created in.
- **tgObjectMetadata**
  The TextGrid metadata as XML object, see WSDL file and XML metadata schema.
- **tgObjectData**
  The data object.
- **howMany**
  An integer value.

#GETVERSION

Just returns the current version of the TG-crud.

Parameters

- NONE

RESTful access

- HTTP GET http://textgridlab.org/1.0/tgcrud/rest/version
- Response: 200 OK, version string delivered in body (text/plain)

#CREATE

The **#CREATE** method of the TG-crud service is used to create TextGrid objects and store them **TO THE GRID**. The following steps will be performed:

- Check if **publish** access is granted to the given RBAC session ID (ONLY if the DIRECTLY PUBLISH option is set to TRUE).
- Check if **create** access is granted to the project resource using the given RBAC session ID.
- Compute the revision number to use.
- Create new TextGrid URI if neded, check URI if given.
- Generate the generated metadata type.
- Get some public data out of the metadata (ONLY if the DIRECTLY PUBLISH option is set to TRUE)
- STORE AGREGATION, EDITION, or COLLECTION DATA to the XML DATABASE.
- STORE ORIGINAL XML DATA to the XML DATABASE.
- Call the AdaptorManager, process data if needed.
    - If an aggregation object is ingested: Add the subject's URI to the aggregation ORE file.
    - If an adaptor URI is existing in the metadata: Read the (baseline) adaptor XSLT file and put the baseline encoding into the XML database.
    - Put the namespaces of XSD files into the RDF database.
    - Put the relations extracted with the AdaptorManager into the RDF database.
    - Add warnings to the metadata, if existing.
- STORE METADATA and DATA TO THE GRID.
- STORE RELATIONS to the RDF DATABASE.
- STORE METADATA to the XML DATABASE.
- REGISTER RESOURCE to the TG-AUTH.
- Set the isPublic flag in TG-AUTH (ONLY if the DIRECTLY PUBLISH option is set to TRUE)
- Add permissions to the metadata.
- Return the complete metadata element.

Parameters

- sessionID (mandatory, String)
- logParameter (optional or empty, String)
- uri (optional or null, URI)
- createRevision (mandatory, Boolean)
- projectId (mandatory, String)
- tgObjectMetadata (mandatory, MetadataContainerType)
- tgObjectData (mandatory, DataHandler)

RESTful access

- HTTP POST http://textgridlab.org/dev/tgcrud/rest/create
- Parameters as stated above
- tgObjectMetadata and tgObjectData as Multipart
- Special header information provided
    - Location: TextGrid URI
    - Last-Modified date
- Response: 200 OK, MetadataContainerType delivered in body (text/xml)
- Errors
    - MetadataParseFault: 400 BAD REQUEST
    - WebApplicationException: 500 INTERNAL SERVER ERROR
    - ObjectNotFoundFault: 404 NOT FOUND
    - AuthFault: 401 UNAUTHORIZED

#CREATEMETADATA (Not implemented yet)

To be used to create TextGrid objects that are holding the object's metadata and an HTTP reference only to the object's data. TG-crud will get the data via HTTP and deliver it via TG-crud#READ.

Parameters

- sessionID (mandatory, String)
- logParameter (optional or empty, String)
- uri (optional or null, URI)
- projectId (mandatory, String)
- externalReference (mandatory, String)
- tgObjectMetadata (mandatory, MetadataContainerType)

#READ

The **#READ** method of the TG-crud service reads TextGrid objects (including metadata) **FROM THE GRID**, and will perform the following steps:

- Check if **read** access is granted to the given URI using the given RBAC session ID.
- Read the metadata and data files FROM THE GRID.
- Fill *dataContributor* and *permissions* tags with the information provided by the checkAccess query.
- Return the complete data and metadata elements.

Parameters

- sessionID (optional or empty for public resources, String)
- logParameter (optional or empty, String)
- uri (mandatory, URI)
- tgObjectMetadata (mandatory for SOAP requests for delivering the metadata, MetadataContainerType)
- tgObjectData (mandatory for SOAP requests for delivering the data, DataHandler)

RESTful access

- HTTP GET http://textgridlab.org/1.0/tgcrud/rest/textgrid:vqmz.0/data
- Special header information provided: Last-Modified
- Response: 200 OK, Object delivered in body (mimetype depending on object type)
- Errors
    - ObjectNotFoundFault: 404 NOT FOUND
    - MetadataParseFault: 400 BAD REQUEST
    - IoFault: 500 INTERNAL SERVER ERROR
    - ProtocolNotImplementedFault: 400 BAD REQUEST
    - AuthFault: 401 UNAUTHORIZED

#READMETADATA

The **#READMETADATA** method of the TG-crud service reads the metadata of a TextGrid object (metadata only) **FROM THE GRID**, and does the following in the given order:

- Check if **read** access is granted to the given URI using the given RBAC session ID.
- Read the metadata file FROM THE GRID.
- Fill *dataContributor* and *permissions* tags with the information provided by the checkAccess query.
- Return the complete metadata element.

Parameters

- sessionID (optional or empty for public resources, String)
- logParameter (optional or empty, String)
- uri (mandatory, URI)

RESTful access

- HTTP GET http://textgridlab.org/1.0/tgcrud/rest/textgrid:vqmz.0/metadata
- Response: 200 OK, MetadataContainerType delivered in body (text/xml)

#UPDATE

The **#UPDATE** method of the TG-crud service updates a TextGrid object including metadata and data. Moreover, it performs the following in the given order:

- Retrieve the metadata FROM THE GRID.
- Store aggregation, edition, and collection data to the XML database.

- Store the original data (if XML) to the XML database.
- Delete the relations for the given object from the RDF database.
- Call the Adaptor Manager.
- Store the relations again to the RDF database.
- Store metadata and data TO THE GRID.
- Store the metadata to the XML database.
- Return the updated metadata element.

User locking is involved here (please see #LOCK and #UNLOCK).

Parameters

- sessionId (mandatory, String)
- logParameter (optional or empty, String)
- uri (mandatory for RESTful access! Not used for SOAP access, the URI is included in the metadata object involved!)
- tgObjectMetadata (mandatory, MetadataContainerType)
- tgObjectData (mandatory, DataHandler)

RESTful access

- HTTP POST http://textgridlab.org/dev/tgcrud/rest/textgrid:1234/update
- Parameters as stated above
- tgObjectMetadata and tgObjectData as Multipart
- Special header information provided
  - Location: TextGrid URI
  - Last-Modified date
- Response: 200 OK, MetadataContainerType delivered in body (text/xml)
- Errors
  - MetadataParseFault: 400 BAD REQUEST
  - IoFault: 500 INTERNAL SERVER ERROR
  - ObjectNotFoundFault: 404 NOT FOUND
  - AuthFault: 401 UNAUTHORIZED
  - UpdateConflictFaull: 409 CONFLICT

#UPDATEMETADATA

The **#UPDATEMETADATA** method of the TG-crud service updates the metadata of a TextGrid object. Moreover, it performs the following in the given order:

- Retrieve the metadata FROM THE GRID.
- Retrieve Adaptor data FROM THE GRID.
- Delete the relations for the given object from the RDF database.
- Store the relations again to the RDF database.
- Store metadata TO THE GRID.
- Store the metadata to the XML database.
- Return the updated metadata element.

User locking is involved here (please see #LOCK and #UNLOCK).

Parameters

- sessionId (mandatory, String)
- logParameter (optional or empty, String)

- uri (mandatory for RESTful access! Not used for SOAP access, the URI is included in the metadata object involved!)
- tgObjectMetadata (mandatory, MetadataContainerType)

RESTful access

- HTTP POST http://textgridlab.org/dev/tgcrud/rest/textgrid:1234/updateMetadata
- Parameters as stated above
- tgObjectMetadata as Multipart
- Special header information provided
    - Location: TextGrid URI
    - Last-Modified date
    - Response: 200 OK, MetadataContainerType delivered in body (text/xml)
    - Errors
    - MetadataParseFault: 400 BAD REQUEST
    - IoFault: 500 INTERNAL SERVER ERROR
    - ObjectNotFoundFault: 404 NOT FOUND
    - AuthFault: 401 UNAUTHORIZED
    - UpdateConflictFaull: 409 CONFLICT

#DELETE

The **#DELETE** method of the TG-crud service performs the following in the given order:

- Delete metadata, original, aggregation, and baseline data from the XML database.
- Unregister object from the TG-auth*.
- Add a *deleted* relation to the RDF database.
- Delete data and metadata FROM THE GRID.

Parameters

- sessionID (mandatory, String)
- logParameter (optional or empty, String)
- uri (mandatory, URI)

RESTful access

- HTTP GET http://textgridlab.org/1.0/tgcrud/rest/textgrid:vqmz.0/delete
- Response: 200 OK

#GETURI

A valid RBAC Session ID given, the method **#GETURI** generates the requested amount of TextGrid URIs, e.g. to prepare and then import a bunch of files via the Import Tool External (koLibRI), or the copy workflow of the TG-publish Service.

Parameters

- sessionID (mandatory, String)
- logParameter (optional or empty, String)
- howMany (mandatory, Integer)

RESTful access

- HTTP GET http://textgridlab.org/1.0/tgcrud/rest/getUri
- Response 200 OK, TextGrid URI list separated by newline (text/plain)

- Errors
  - ObjectNotFoundFault: 404 NOT FOUND
  - IoFault: 500 INTERNAL SERVER ERROR
  - AuthFault: 401 UNAUTHORIZED

#LOCK

The implementation using the NOID (see TG-noid) locks TextGrid objects (that is, their URIs) depending on the user ID (write access needed) and an "automagic unlocking time" currently set to 30 minutes. If an URI is not yet locked, any user owning writing access is able to lock this URI. Only a user who has locked the object in the first place can **(a)** update the object and metadata (save object and save metadata), and **(b)** re-lock to keep the object locked, and should do so before the 30 minutes have expired. If the unlocking time has exceeded, any other user with writing access can lock the object again. Updating an object will perform no re-locking, the lock just stays alive.
No REST call is provided yet.

Parameters

- sessionId (mandatory, String)
- logParameter (optional or empt, String)
- baseUri (mandatory, URI)

Returns a boolean value that states if the locking succeeded or not: FALSE in case of an error only, TRUE if locking succeeded, and it throws an IoFault in case another user already holds a lock. The user ID of that user is included as exception message.

Notes

- There is no method for locking (and unlocking) more than one TextGrid URI at once, as the NOID is not able to address more than one URI at once either.
- Published objects can not be locked or unlocked.
- If a user works with both the TG-lab 1.0(.2) and the TG-lab 2.0, updating will still be possible using the TG-lab 1.0(.2), although the object is locked with the 2.0 TG-lab. Hence, combining the lab versions is not recommended. If different users with writing access to the same document are working with an old TG-lab and a new one, updating the same document is not possible with the old TG-lab.

RESTful access

- HTTP GET http://textgridlab.org/1.0/tgcrud/rest/textgrid:1234.0/lock
- Response 200 OK, boolean response in body (text/plain)

#UNLOCK

The implementation using the NOID (see TG-noid) unlocks TextGrid objects. Unlocking is permitted for the user who has locked the object, and for any user (with writing access), if the automatic unlocking time has elapsed.

Parameters

- sessionId (mandatory)
- logParameter (optional or empty string)
- baseUri (mandatory)

Returns a boolean value that states if the unlocking succeeded or not: FALSE in case of an error only, TRUE if unlocking succeeded, and it throws an IoFault in case another user already holds a lock. The user ID of that user is included as exception message.

RESTful access

- HTTP GET http://textgridlab.org/1.0/tgcrud/rest/textgrid:1234.0/unlock
- Response 200 OK, boolean response in body (text/plain)

#MOVEPUBLIC

Moves data from the non-public repository storage location to the public repository storage location. Needs special authentication and is used from other services as TG-publish only.
No RESTful access is provided.

2.2.8. The TG-crud Client

A simple way to work with the TG-crud service is to just include the *tgcrud-client* Maven module to your own Maven modules as dependency:

```
<groupId>info.textgrid.middleware</groupId>
<artifactId>tgcrud-client</artifactId>
<version>2.3.4-SNAPSHOT</version>
```

Subsequently, you may use the method *TGCrudClientUtilities.getTgcrud()*, provide a TG-crud service endpoint and simply call this tgcrud client's methods. For example:

```
// Create TG-crud service client, use MTOM.
TGCrudService tgcrud =
TGCrudClientUtilities.
getTgcrud("http://textgridlab.org/1.0/tgcrud/TGCrudService?wsdl", true);

// Get TG-crud's version.
System.out.println("TG-crud version is " + tgcrud.getVersion());

// Read the metadata of a TextGrid object from the Digitale Bibliothek
// (No sessionId is needed because the object is public).
MetadataContainerType metadata = tgcrud.readMetadata("", "", "textgrid:vqmw");

// Print out the JaxB metadata object.
JAXB.marshal(metadata, System.out);
```

2.2.9. Online JUnit Tests

The productive TextGrid 2.0 JUnit tests tag can be found here: /middleware.tgcrud-2012-05-16-v3.0.0-SNAPSHOT-TG2.0/clients/tgcrudclient/

The TG-crud client SVN repository (trunk) containing the current develop version that is (most of the time) runnable is available from:
https://develop.sub.uni-goettingen.de/repos/textgrid/trunk/middleware/tgcrud/clients/tgcrudclient/

2.3. TG-noid

TG-noid is a service that mints (generates) the TextGrid URIs for each TextGrid object. It is used by TG-crud only. The TextGrid URIs have the prefix ***textgrid:*** followed by a generated character sequence consisting of the NOID enhanced character set. Starting with ***textgrid:00*** the URIs can be increased infinitly. TG-noid is a specially configured NOID (Nice Opaque Identifier) service that mainly consists of a Perl script and some HTTP server configurations.

2.3.1. Further Documentation

The NOID is a minter and name resolver used as a microservice by the UC Curation Center of the California Digital Library. Further information regarding the NOID and related issues: https://wiki.ucop.edu/display/Curation/NOID

The Perl implementation used by TextGrid: Noid@CPAN

Direct links to the NOID Perl module documentation: http://search.cpan.org/~jak/Noid-0.424/noid (HTML), http://www.cdlib.org/inside/diglib/ark/noid.pdf (PDF)

2.3.2. Installation

Download and unpack http://search.cpan.org/CPAN/authors/id/J/JA/JAK/Noid-0.424.tar.gz

to a folder of your choice. Follow the installing instructions of the NOID.

Quick Installation Guide

Install the ***libberkeleydb-perl*** as Perl module ***BerkeleyDB***, and compile the NOID as follows. Insert the following to your command line (in the main NOID folder):

```
perl Makefile
make
make test
make install
```

2.3.3. Configuration

Copy the binary from **/usr/local/bin/noid** to ***~/htdocs/nd/noid***, to ***noidr_textgrid***, and ***noidu_textgrid*** (last two names depend on your minter to be created). Call

```
noid dbcreate textgrid:.zee
```

Add the noid apache configuration to ***/etc/apache2/sites-available/default:***

```
# ------------------------------------------------------------------------
# All the NOID configuration things following here for minting TextGrid URIs
# ------------------------------------------------------------------------

# ScriptAlias /cgi-bin/ /home/textgrid-noid/htdocs/nd/
<Directory "/home/textgrid-noid/htdocs/nd/">
    AuthType Basic
           AuthName "The TextGrid URI NOID Service"
           AuthUserFile /etc/apache2/tgnoid.htpasswd
           Require valid-user
           AllowOverride None
           Options +ExecCGI -Includes
           Order allow,deny
           Allow from all
</Directory>

# Make the server recognize links to htdocs/nd
ScriptAliasMatch ^/nd/noidr(.*) "/home/textgrid-noid/htdocs/nd/noidr$1"
ScriptAliasMatch ^/nd/noidu(.*) "/home/textgrid-noid/htdocs/nd/noidu$1"

# Define all the rewrite maps, start every program once on server start
RewriteMap rslv_textgrid prg:/home/textgrid-noid/htdocs/nd/noidr_textgrid
```

Change group of NOID folder to www-data, set write permissions for group, both recursively (must be ***NOID 775*** and ***NOID/* 664***).

Authentication

If you want to secure the minter, simply use HTTP authentication protocols (e.g. basic auth):
Set user/password authentication in ***/etc/apache2/sites-available/default*** (as shown above).
Create password file for the NOID and add user tgcrud, for TG-crud is the only one to use the TG-noid and create URIs:

```
htpasswd -c tgnoid.htpasswd tgcrud
```

Change permissions to 600.
Add ***/etc/apache2/tgnoid.passwd*** to NOID config.
Set user/passwd in TG-crud config file (as shown above).

Known Problems

Error: No "Env" object (Permission denied): Set correct access permissions to folder NOID (775 and 664).

2.4. TG-conf

2.4.1. Installation

First, download https://wiki.sub.uni-goettingen.de/textgrid/images/5/56/Confserv.zip, then unzip to /var/www, you should now have /var/www/confserv/getAllJ.php. Rename the file to

```
mv /var/www/confserv/getAllJ.php /var/www/confserv/getAllJ
```

and add the following line to /etc/apache2/sites-enabled/000-default:

```
<Location /confserv/getAllJ>
        SetHandler php5-script
</Location>
```

2.5. TG-search

2.5.1. Prerequisites

Sesame

To install openrdf sesame 2.6.4 please follow the steps below:

- download openrdf-sesame-2.6.4-sdk.tar.gz
  from http://sourceforge.net/projects/sesame/files/Sesame%202/2.6.4/
- untar
- copy wars to /var/lib/tomcat6/webapps
- mkdir /usr/share/tomcat6/.aduna
- chown tomcat6:tomcat6 /usr/share/tomcat6/.aduna
- create textgrid repository (the dot at the end of console commands is important)
- cd openrdf-sesame-2.6.4
- bin/console.sh
- connect http://localhost:8180/openrdf-sesame.
- create native.
  - insert the ID "textgrid" and a title "textgrid native store", set indexing to spoc,posc,opsc,sopc
  - /var/lib/tomcat/webapps/sesame/WEB-INF/web.xml, you should see that there is a security constraint for "sesame-admin".
- To match this, create a "sesame-admin" role in tomcat-users.xml, and create a sesame user with the sesame admin role.
- Create the user as user=workbench, password=textvre1

eXist 1.4.1

Installation

Download jar from http://sourceforge.net/projects/exist. At the time of writing these notes, 1.4.1 was available at

http://sourceforge.net/projects/exist/files/Stable/1.4.1/eXist-setup-1.4.1-rev15155.jar/download

```
mkdir /usr/local/exist
java -jar eXist-setup-1.4.1-rev15155.jar -p /usr/local/exist
```

By default eXist will install with a jetty server on port 8080. This is fine and does not need to be changed. Start the server using the following command:

```
./bin/startup.sh &
```

Set username/password by browsing to http://localhost:8080/exist and use management interface to set admin username and password ("admin", "the_password"). To integrate exist in the startup sequence do

```
ln -s /usr/local/exist/tools/wrapper/bin/exist.sh /etc/init.d/exist
update-rc.d exist defaults
```

To configure Memory settings look into /usr/local/exist/tools/wrapper/conf/wrapper.conf. Using wrapper the logs go to /usr/local/exist/tools/wrapper/logs, you may add a symlink to a more convenient place, e.g.:

```
ln -s /usr/local/exist/tools/wrapper/logs /usr/local/exist/logs
```

2.5.2. Installing TG-search

TG-search non-public

Copy
/tgsearch/tgsearch-webapp/src/main/webapp/WEB-INF/tgsearch-nonpublic.properties.tmpl
to
/tgsearch/tgsearch-webapp/src/main/webapp/WEB-INF/tgsearch-nonpublic.properties, afterwards edit the following properties in tgsearch-nonpublic.properties:

```
tgauth.endpoint = http://localhost/rbac/tgextra.php
sesame.endpoint = http://localhost:8180/openrdf-
sesame/repositories/textgrid
exist.endpoint = http://localhost:8080/exist/rest/
exist.user = tgsearch
exist.pw = the_password
sesame.user = workbench
sesame.pw = the_password
```

Now build the WAR package:

```
cd tgsearch/
mvn package -Pnonpublic
```

The file tgsearch.war in tgsearch/tgsearch-webapp/target/tgsearch.war is created. Copy this file to the webapps area oft he tomcat server.

There is a set of scripts that need to be added to the exist installation. To do this, start the exist client as above and log into the exist instance. There is an exist sub directory in the tgsearch directory. Replicate the directory structure in exist by Tools > Create Collection and work down through the structure. Once this is complete, import all the files at the appropriate point in the structure. Note that the following files need to be edited before adding to eclipse:

- exist/db/tgsearch/2.0.1/modules/agghelp.xqm
- exist/db/tgsearch/2.0.2/modules/agghelp.xqm
- exist/db/tgsearch/2.0.3/modules/agghelp.xqm

Each of these files contains references to an openrdf-sesame repository. These references must be changed to match your local installation, for example:

```
"http:/localhost:8180/openrdf-sesame/repositories/textgrid?query="
```

TG-search public

Install eXist into /usr/local/exist-public like described here: eXist 1.4.1, modify /usr/local/exist-public/tools/jetty/etc/jetty.xml to use port 8999, you may set symlinks and add to startup sequence accordingly > eXist 1.4.1.

Saxon XSLT needs to be used:

```
cd /usr/local/exist-public/lib/user/
wget -U NoSuchBrowser/1.0
http://repo1.maven.org/maven2/net/sourceforge/saxon/saxon/9.1.0.8/saxo
n-9.1.0.8.jar
wget -U NoSuchBrowser/1.0
http://repo1.maven.org/maven2/net/sourceforge/saxon/saxon/9.1.0.8/saxo
n-9.1.0.8-xpath.jar
wget -U NoSuchBrowser/1.0
http://repo1.maven.org/maven2/net/sourceforge/saxon/saxon/9.1.0.8/saxo
n-9.1.0.8-dom.jar
```

Now edit /usr/local/exist-public/conf.xml to use saxon: Find the line

```
<transformer  class="org.apache.xalan.processor.TransformerFactoryImpl"
caching="yes"/>
```

and replace it with

```
<transformer class="net.sf.saxon.TransformerFactoryImpl">
    <attribute name="http://saxon.sf.net/feature/version-warning"
value="false" type="boolean"/>
</transformer>
```

Start up exist and insert contents from
https://develop.sub.uni-goettingen.de/repos/textgrid/trunk/middleware/tgsearch/exist/db/
into the database.

Now add a repository textgrid-public to sesame, and same with TG-search non-public, build with "mvn package" (not -Pnonpublic) and install the tgsearch-public.war to tomcat.

2.5.3. Setting up the TextGrid Repository Browser

Set up TextGrid Repository website

Perform the following steps:

```
cd /var/www
svn co https://develop.sub.uni-
goettingen.de/repos/textgrid/trunk/middleware/tgsearch/textgridrep-website/
tgrep
```

Now edit tgrep/js/conf.js, and set sandbox to false.

Add proxy config to /etc/apache/sites-enabled/000-default:

```
<Location /services/tgcrud-public >
    ProxyPass http://localhost:8180/tgcrud retry=0
    ProxyPassReverse http://localhost:8180/tgcrud
    Allow from all
</Location>

<Location /services/exist-public/xslt >
    ProxyPass http://localhost:8999/exist/rest/xslt retry=0
    ProxyPassReverse http://localhost:8999/exist/rest/xslt
    Allow from all
</Location>

<Location /services/exist-public/xql >
    ProxyPass http://localhost:8999/exist/rest/tgsearch/2.0.1/query retry=0
    ProxyPassReverse http://localhost:8999/exist/rest/tgsearch/2.0.1/query
    Allow from all
</Location>

<Location /services/tgsearch-public >
    ProxyPass http://localhost:8280/tgsearch-public retry=0
    ProxyPassReverse http://localhost:8280/tgsearch-public
    Allow from all
</Location>

<Location /services/sesame >
    ProxyPass http://localhost:8180/openrdf-sesame retry=0
    ProxyPassReverse http://localhost:8180/openrdf-sesame
    Allow from all
    <LimitExcept GET>
      Require valid-user
    </LimitExcept>
</Location>

<Location /services/tgpublish >
    ProxyPass http://localhost:8280/tgpublish retry=0
    ProxyPassReverse http://localhost:8280/tgpublish
    Allow from all
</Location>

<Location /services/tgsearch >
    ProxyPass http://localhost:8180/tgsearch retry=0
    ProxyPassReverse http://localhost:8180/tgsearch
    Allow from all
```

```
</Location>

<Location /services/tgcrud >
    ProxyPass http://localhost:8180/tgcrud retry=0
    ProxyPassReverse http://localhost:8180/tgcrud
    Allow from all
</Location>

<Location /fedora >
    ProxyPass http://localhost:8081/fedora retry=0
    ProxyPassReverse http://localhost:8081/fedora
    Allow from all
</Location>

Alias /PWchange /var/www/PWchange/htdocs
<Directory "/var/www/PWchange/htdocs/*">
    Options +FollowSymLinks
    Order Allow,Deny
    Allow from all
</Directory>
```

Now you should be able to browse published content on http://localhost/tgrep/.

2.6. TG-publish

The TG-publish service is used to publish documents and/or TextGrid editions/collections created within the TextGridLab to the public TextGrid Repository. Documents that are published in the TextGrid Repository

- can be publicly searched from within the TextGridLab
- are publicly available in the TextGrid Repository Browser.
- have got a Handle persistent identifier to be able to stay accessible.
- are stored in a secure and robust repository environment that provides retrieval for the long term.

Three different options allow for publishing data:

1. Valid TextGrid Editions or Collections created in the TextGridLab can be published. To learn more about the mandatory TextGrid Metadata, please have a look at the online documentation.
2. Some technical data from the TextGridLab can be published as WorldReadable objects. These objects and their metadata are tested differently for validity; they are not added to the public search index, and no PIDs will be assigned.
3. Objects that shall be checked for validity after importing can first be published into the TextGrid Repository Sandbox. After importing, objects may be verified in the TextGrid Repository Sandbox Browser, then be published using a special policy of TG-publish, or again deleted if not valid/complete/errorful. As of today, only the final publishing process is covered by TG-publish; hence, access to the RDF database is required. Publishing and deletion only are possible via the Import Tool External (koLibRI) (see below). As yet, the Sandbox is not used from within the TextGridLab either.

In addition, the TG-publish service may be used to copy TextGrid objects from (a) the TextGridLab and (b) the TextGridRep to own TextGrid projects in the TG-lab. TG-publish uses the workflow library koLibRI (kopal Library of Retrieval and Ingest).

2.6.1. Subversion Repository

The SVN repository can be found at: https://develop.sub.uni-goettingen.de/repos/kolibri/

Related tags of the TG-publish documentation:

https://develop.sub.uni-goettingen.de/repos/kolibri/tags/2012-04-28_kolibri-tgpublish-service_TG2.0/
https://develop.sub.uni-goettingen.de/repos/kolibri/tags/2012-04-28_kolibri-tgpublish-client_TG2.0/
https://develop.sub.uni-goettingen.de/repos/kolibri/tags/2012-04-28_kolibri-tgpublish-api_TG2.0/

2.6.2. Version and Documentation

This page is valid for TG-publish Service Version

```
2.0.0-SNAPSHOT-2012-04-28T00:04-'TextGrid 2.0'
```

Currently used TG-publish version: http://textgridlab.org/1.0/tgpublish/version.

For a more detailed understanding of the koLibRI workflow tool and its modules, see the koLibRI documentation: koLibRI Documentation Version 2.0 (work in progress)

2.6.3. Installation

Generally, a complete TG-publish Service installaltion consists of (or needs):

1. The TG-publish service itself.
2. A second TG-crud instance for public data (TG-crud public).
3. A second TG-search instance for public data including a second eXist and Sesame instance (TG-search public).
4. The public TextGrid Repository Browser (including the Repository Sandbox)

Downloading TG-publish

TG-publish can be downloaded from the TextGrid Archiva Repository as a WAR file from

kolibri-tgpublish-service-2.0.0-SNAPSHOT.war

and be renamed to *tgpublish.war* (just for simplicity) and directly be deployed into a Tomcat's *webapp* folder. A newer version may of course be used if existing.

If anything needs to be added to TG-publish (e.g. more koLibRI modules extending the functionality), the current sources of the koLibRI are available from:

https://develop.sub.uni-goettingen.de/repos/kolibri/trunk/

Subsequently, perform

```
mvn package
```

in */kolibri/kolibri-tgpublish-service/*. All modules for building TG-publish might require a verification, as any of its modules begin with *kolibri-tgpublish*. As TG-lab depends on it, the AP and client should not be changed. To test the TG-publish service (locally, but with non-local services) you might run

```
mvn tomcat:run
```

2.6.4. Configuration

There are mainly two configuration files coming with the tg-publish koLibRI module that need to be taken care of: *config.xml* and *policies.xml*. The latter is used to define the TG-publish workflow and leads the publishing process through the various ActionModules that are normally processed one by one. These modules can share information using a custom data object. Detailed information is available from the work-in-progress version 2.0 referred to above.

TG-publish is pre-configured to put its configuration files into */etc/textgrid/tgpublish/* and log to */var/log/textgrid/tgpublish/* using [dateTime].log as a filename.

Please create the appropriate folders, then copy the config files from

https://develop.sub.uni-goettingen.de/repos/kolibri/trunk/kolibri-tgpublish-service/config/

into the config folder. Remember to set the permissions and owner settings so that Tomcat can write to it.

More config files may be needed from

https://develop.sub.uni-goettingen.de/repos/kolibri/trunk/config/

In case file-not-found-errors should occur, add the following:

```
dias_formatregistry.xml
jhove.conf
```

policies.xml

There are currently four policies to be used with the TextGridLab:

1. *TGPublish*
2. *TGPublishWorldReadable*
3. *TGPublishSandboxData*, and
4. *TGCopy*

These workflows (or policies) are described in the *policies.xml* file and define the order of processing koLibRI ActionModules. Each of the three workflows is started as a ProcessStarter with the current configuration (see below). *TGPublish* is used from within the TG-lab using the Publish Perspective, *TGPublishWorldReadable* is used from within the TG-lab as well, but only applies to single technical files as e.g. XML Schema documents, XSLT stylesheets, TextGrid workflow documents, etc. The variety of files to be able to publish worldReadable can be checked by requesting the worldReadable List. *TGPublishSandboxData* is used to finally publish objects that were imported to the TextGrid Repository Sandbox and is used from e.g. the Import Tool External (koLibRI), and last but not least *TGCopy* used from within the TG-copy workflow to copy TextGrid objects from either the public repository or the non-public repository to own projects for further processing. Rewriting URIs and other items are included here. The respective three policies are described in detail hereafter.

TGPublish

```
<?xml version="1.0" encoding="UTF-8"?>
<policy name="TGPublish">
  <step class="textgrid.PublishStart">
    <step class="textgrid.PublishCheckEdition">
      <step class="textgrid.CheckIsPublic">
        <step class="textgrid.CheckReferences">
          <step class="textgrid.GetPids">
            <step class="textgrid.ModifyAndUpdate">
              <step class="textgrid.CopySearchIndex">
                <step class="textgrid.CopyRelationData">
                  <step class="textgrid.MoveToStaticGridStorage">
                    <step class="textgrid.UpdateTgauth">
                      <step class="textgrid.PublishComplete" />
                    </step>
                  </step>
                </step>
              </step>
            </step>
          </step>
        </step>
      </step>
    </step>
  </step>
</policy>
```

PublishStart

Just marks the publish process started.

PublishCheckEdition

Checks for correct Edition/Collection Metadata.

CheckIsPublic

Checks for already published objects.

CheckReferences

Checks if some objects that are referred to, are NOT contained in the current Edition/Collection to publish.

GetPids

Fetches PIDs for every object's TextGrid URI using the GWDG Handle Service.

ModifyAndUpdate

Performs rewriting of several URIs to PIDs, modifies all necesarry object metadata and/or data, and finally updates everything calling TG-crud#UPDATEMETADATA or TG-crud#UPDATE.

CopySearchIndex

Copies the XML search index to the public XML database.

CopyRelationData

Copies the RDF relation data to the public RDF database.

MoveToStaticGridStorage

Moves all metadata and data to the public storage location.

UpdateTgauth

Updates the TG-auth calling the method TG-auth#PUBLISH

PublishComplete

PublishComplete is called just to ensure the operation was finished successfully, and to report to logfiles, etc.

TGPublishWorldReadable

```
<?xml version="1.0" encoding="UTF-8"?>
<policy name="TGPublishWorldReadable">
  <step class="textgrid.PublishStart">
    <step class="textgrid.PublishCheckWorldReadable">
      <step class="textgrid.ModifyAndUpdate">
        <step class="textgrid.MoveToStaticGridStorage">
          <step class="textgrid.UpdateTgauth">
            <step class="textgrid.PublishComplete" />
          </step>
        </step>
      </step>
    </step>
  </step>
</policy>
```

PublishStart

See above.

PublishCheckWorldReadable

Checks for correct WorldReadable Metadata.

ModifyAndUpdate

Performs rewriting of several URIs to PIDs, modifies all necesarry object metadata and/or data, and finally updates everything calling TG-crud#UPDATEMETADATA or TG-crud#UPDATE.

MoveToStaticGridStorage

Moves all metadata and data to the public storage location.

UpdateTgauth

See above.

PublishComplete

See above.

TGPublishSandboxData

```
<?xml version="1.0" encoding="UTF-8"?>
<policy name="TGPublishSandboxData">
  <step class="textgrid.PublishStart">
    <step class="textgrid.UpdateTgauth">
      <step class="textgrid.ReleaseNearlyPublishedRelation">
        <step class="textgrid.PublishComplete" />
      </step>
    </step>
  </step>
</policy>
```

PublishStart

See above.

UpdateTgauth

See above.

ReleaseNearlyPublishedRelation

Releases the nearlyPublished relation in the TG-rep's Sesame triple store so the object is viewable and searchable in the public TextGrid repository browser and TextGridLab search GUI.

PublishComplete

See above.

TGCopy

```
<?xml version="1.0" encoding="UTF-8"?>
<policy name="TGCopy">
  <step class="textgrid.StartCopy">
    <step class="textgrid.GatherObjectUris">
      <step class="textgrid.ModifyAndCreate">
        <step class="textgrid.CopyComplete" />
      </step>
    </step>
  </step>
</policy>
```

CopyStart

Just marks the copy process started.

GatherObjectUris

Retrieves all referenced URIs from the objects out of the given URI list (out of all aggregations/editions/collections recursively), and adds every URI to the PublishResponse object list.

ModifyAndCreate

Retrieves every URI from the PublishResponse object list from the TG-crud, rewrites aggregation lists and other URIs includes, and creates a new TextGrid object in the project given.

CopyComplete

CopyComplete is called just to ensure the operation was finished successfully, and to report to logfiles, etc.

config.xml

config.xml or in this case *config__tgpublish.xml* is the main koLibRI configuration file. The processStarters are configured as well as all the ActionModules, also global settings can be configured here. The TextGrid specific ProcessStarters and ActionModules are described inside the config file (see *description* tags):

2012-04-28_kolibri-tgpublish-service_TG2.0/config/config__tgpublish.xml

XML config file tags not documented are not used by the TG-publish (and not needed); their meaning is described in the koLibRI documentation, or the main koLibRI configuration file:

koLibRI-2.0-SNAPSHOT-TG1.0-P1/config/config.xml

Logging

Currently, the koLibRI logs to stdout (see e.g. the Tomcat's *catalina.out* log) as well as to a logfile located at the configured logfile location (see *config.xml*). The logfile's name contains a timestamp, and a new file is created every time the koLibRI Workflow Tool is started.

2.6.5. TextGridLab GUI

A description of how the TG-publish and TG-copy is used from within the TextGridLab is available from TG-publish GUI and TG-copy GUI (Context Menu of the Navigator View).

2.6.6. API Overview

Method	Request Parameters	Response	Examples
`getStatus`	TextGrid URI or TG-copy UUID	publishResponse XML	`/tgpublish/textgrid:1234/status`
`getVersion`	none	Version String	`/tgpublish/version`
`listWorldReadables`	none	worldReadableMimetypes XML	`/tgpublish/listWorldReadables`
`Publish`	sid, log, ignoreWarnings, dryRun	HTTP 201 OK	`/tgpublish/textgrid:1234/publish?sid=SID&log=&ignoreWarnings=TRUE&dryRun=FALSE`
`publishSandboxData`	sid, log, ignoreWarnings, dryRun	HTTP 201 OK	`/tgpublish/textgrid:1234/publishSandboxData?sid=SID&log=&ignoreWarnings=TRUE&dryRun=FALSE`
`publishWorldReadable`	sid, log, ignoreWarnings, dryRun	HTTP 201 OK	`/tgpublish/textgrid:1234/publishWorldReadable?sid=SID&log=&ignoreWarnings=TRUE&dryRun=FALSE`
`copy`	sid, log, uri (repeatable), projectId, newRevision	TG-copy UUID	`/tgpublish/copy?sid=SID&log=&uri=textgrid:1234&uri=textgrid:2345&projectId=TGPR&newRevision=FALSE`

getStatus()

Retrieves the status of a certain publish process using the specific TextGrid URI or the TG-copy UUID, e.g.

http://textgridlab.org/1.0/tgpublish/textgrid:1234/status

Optional parameters are

- the RBAC Session ID, and
- the log service parameter.

The TextGrid URI or the TG-copy UUID is provided in the RESTful URL path, see example above. The publish response will be provided as an XML file, which is described below.

getVersion()

Current version of the TG-publish installation:
http://textgridlab.org/1.0/tgpublish/version

listWorldReadables()

A list of TextGrid objects that can be published as single WorldReadable objects is available from: http://textgridlab.org/1.0/tgpublish/listWorldReadables

publish()

Publishes data from the TG-lab, needs a TextGrid URI as resource, and takes the following as input parameters:

- *The RBAC session ID (mandatory)* [As used in the TG-lab]
- *The log parameter (optional – default is "")* [As used in the TG-lab, too]
- *The ignoreWarnings trigger (default is FALSE)* [Any warnings are ignored, if set to TRUE. Objects containing warnings will be published anyhow]
- *The dryRun trigger (default is TRUE)* [If set to TRUE, nothing will really be published, but just checked]

The object to publish is provided with the URI path parameter as following:
http://textgridlab.org/1.0/tgpublish/textgrid:1234/publish?sid=MB896JHG&log=&ignoreWarnings=TRUE&dryRun=FALSE

The publish call only returns an HTTP 201 OK indicating the service did queue the request and will be processing it as soon as possible. If an HTTP error is returned, the publishing process was not started and will not be started. A requests outcome can be verified by calling status().

publishSandboxData()

Publishes ultimatively to the TextGridRep. Currently only used by the Import Tool External (koLibRI). Usage of the sandbox within the TG-lab is envisaged.

publishWorldReadable()

Publishes an object as WorldReadable. The same parameters as in *publish()* apply.

copy()

Copies the objects belonging to the given URIs and their aggregated objects (from aggregations/editions/collections) to the project belonging to the given project ID. Objects from the TG-lab (non-public repository) can be copied and/or to the public repository to own projects, e.g. for further processing.

The following parameters apply:

- *The RBAC session ID (mandatory)* [As used in the TG-lab]
- *The log parameter (optional – default is "")* [As used in the TG-lab, too]
- *The URIs of the objects to copy* [e.g URIs of aggregations/collections/editions (in that case, all subelements are gathered automatically) or URIs of single objects. This parameter is repeatable, hence a list of favourite objects can be built and copied into own projects.]
- *The Project ID* [of the project to copy the objects to.]
- *newRevision* [A boolean parameter to create new revisions from all copied objects (TRUE), or not (FALSE).]

HTTP example of the copy() method:
http://textgridlab.org/1.0/tgpublish/copy?sid=MB896JHG&log=&uri=textgrid:1234&uri=textgrid:2345&projectId=TGPR&newRevision=FALSE

The copy call returns a session UUID as a string that (a) indicates the service queued the request and will be processing it as soon as possible, and that (b) query via the status() call a copy status. If an HTTP error is returned, the copying process was not started and will not be statred subsequently.

As to track the provenience of the copied objects, an *isDerivedFrom* relation is set in the TextGrid metadata of each copied object.

2.6.7. The TG-publish WADL Service Description

The WADL file of the TG-publish response is available as Service WADL.

2.6.8. TG-publish Response

XML Schema

The following shows the embedded PublishResponse XML schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="unqualified">
    <xs:element name="publishResponse" type="publishResponse" />
    <xs:element name="worldReadableMimetypes" type="worldReadableMimetypes" />
    <xs:complexType name="publishResponse">
        <xs:sequence>
            <xs:element maxOccurs="unbounded" minOccurs="0" name="PublishObject"
                type="publishObject" />
            <xs:element minOccurs="0" name="PublishStatus" type="publishStatus" />
        </xs:sequence>
        <xs:attribute name="dryRun" type="xs:boolean" />
    </xs:complexType>
    <xs:complexType name="publishObject">
        <xs:sequence>
            <xs:element maxOccurs="unbounded" minOccurs="0" name="error" type="publishError" />
            <xs:element minOccurs="0" name="referencedUris" type="referencedUris" />
            <xs:element maxOccurs="unbounded" minOccurs="0" name="warning" type="publishWarning" />
        </xs:sequence>
        <xs:attribute name="uri" type="xs:string" />
        <xs:attribute name="pid" type="xs:string" />
        <xs:attribute name="status" type="statusType" />
    </xs:complexType>
```

```
<xs:complexType name="publishError">
    <xs:sequence>
        <xs:element minOccurs="0" name="message" type="xs:string" />
        <xs:element minOccurs="0" name="type" type="errorType" />
    </xs:sequence>
</xs:complexType>
<xs:complexType name="referencedUris">
    <xs:sequence>
        <xs:element maxOccurs="unbounded" minOccurs="0" name="uri" type="xs:string" />
    </xs:sequence>
</xs:complexType>
<xs:complexType name="publishWarning">
    <xs:sequence>
        <xs:element minOccurs="0" name="message" type="xs:string" />
        <xs:element minOccurs="0" name="type" type="warningType" />
    </xs:sequence>
</xs:complexType>
<xs:complexType name="publishStatus">
    <xs:sequence>
        <xs:element maxOccurs="unbounded" minOccurs="0" name="module" type="module" />
    </xs:sequence>
    <xs:attribute name="progress" type="xs:int" use="required" />
    <xs:attribute name="processStatus" type="processStatusType" />
    <xs:attribute name="activeModule" type="xs:string" />
</xs:complexType>
<xs:complexType name="module">
    <xs:sequence>
        <xs:element maxOccurs="unbounded" minOccurs="0" name="message" type="xs:string" />
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" />
    <xs:attribute name="status" type="statusType" />
</xs:complexType>
<xs:complexType name="worldReadableMimetypes">
    <xs:sequence>
        <xs:element maxOccurs="unbounded" minOccurs="0" name="regexp" type="xs:string" />
    </xs:sequence>
</xs:complexType>
<xs:simpleType name="statusType">
    <xs:restriction base="xs:string">
        <xs:enumeration value="OK" />
        <xs:enumeration value="WARNING" />
        <xs:enumeration value="ERROR" />
        <xs:enumeration value="NOT_YET_PUBLISHED" />
        <xs:enumeration value="ALREADY_PUBLISHED" />
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="errorType">
    <xs:restriction base="xs:string">
        <xs:enumeration value="NOT_SPECIFIED" />
        <xs:enumeration value="AUTH" />
        <xs:enumeration value="WRONG_CONTENT_TYPE" />
        <xs:enumeration value="NO_PUBLISH_RIGHT" />
        <xs:enumeration value="PID_GENERATION_FAILED" />
        <xs:enumeration value="MISSING_METADATA" />
        <xs:enumeration value="ALREADY_PUBLISHED" />
        <xs:enumeration value="METADATA_WARNINGS_EXIST" />
        <xs:enumeration value="SERVER_ERROR" />
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="warningType">
    <xs:restriction base="xs:string">
        <xs:enumeration value="NOT_SPECIFIED" />
        <xs:enumeration value="CHECK_REFERENCES" />
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="processStatusType">
    <xs:restriction base="xs:string">
        <xs:enumeration value="FINISHED" />
        <xs:enumeration value="RUNNING" />
        <xs:enumeration value="FAILED" />
        <xs:enumeration value="NOT_QUEUED" />
    </xs:restriction>
```

```
    </xs:simpleType>
</xs:schema>
```

publishResponse Root Element

The *publishResponse* element contains the attribute *dryRun*, the element *publishStatus* and a list of *publishObject* elements.

- *dryRun*
  The attribute *dryRun* tells if a publish process is or was running or just a publish test without any consequences.
- *publishStatus*
  This element contains the current status of the overall publishing process.
  - *module*: Possibly a list of processed modules (not yet used by TG-publish).
  - *progress*: The overall progress in percent. Each module can implement updating of this progress value. Use e.g. for progress bars.
  - *activeModule*: Holds the currently *activeModule* as the qualified name of the corrresponding koLibRI Java ActionModule.
  - *processStatus*: Describes the current overall process status, and might contain the following values:
    - *FINISHED*: A given TG-publish process is finished, either with failure or success
    - *RUNNING*: The current process is still running.
    - *FAILED*: The TG-publish process failed. See object list for reasons.
    - *NOT_QUEUED*: The given TextGrid URI is not yet queued in the TG-publish queue. Either you have not started a publish process with this URI, or the process has not yet begun.
- *publishObject*
  Each of the objects in the list holds the following data, the root Collection or Edition is also provided, normally as the first element of the list:
  - *uri*: The URI of the object to publish.
  - *pid*: The generated PID of the object.
  - *status*: This status describes the status of the single object; it might contain:
    - *OK*: Object was processed successfully.
    - *WARNING*: Object was processed successfully, but there were warnings (can yet be published using the *ignoreWarnings* flag).
    - *ERROR*: An error occured while publishing.
    - *NOT_YET_PUBLISHED*: The object was not yet published (according to TG-auth*).
    - *ALREADY_PUBLISHED*: The object already is published (according to TG-auth*).
- *publishError*: Errors occured processing this object.
  - *AUTH*
  - *WRONG_CONTENT_TYPE*
  - *NO_PUBLISH_RIGHT*
  - *PID_GENERATION_FAILED*
  - *MISSING_METADATA*

    - *ALREADY_PUBLISHED*
    - *METADATA_WARNINGS_EXIST*
    - *SERVER_ERROR*
    - *NOT_SPECIFIED*
- *referencedUris*: List of URIs that are references from within this object, and NOT contained in the current Edition or Collection.
- *publishWarning*: Warnings occured processing this object.
    - *CHECK_REFERENCES*: Check the referenced URIs.
    - *NOT_SPECIFIED*: Other warning.

2.7. TG-workflow

The workflow solution for TextGrid consists of three components:

- The Workflow Engine. TextGrid uses the Generic Workflow Execution Service http://www.gridworkflow.org/kwfgrid/gwes-web/. It is based on High-Level Petri Nets where operations are Petri Net transitions and Data are Petri Net place markers. We use an adapted Web Service plugin for SOAP and a newly created one for REST Services, both of which are available upon request
- The GWES proxy. As the engine runs behind a firewall, the proxy provides for basic operations on the engine. It is integrated with tg-auth*.
- The TextGridLab which
    - lets users compose workflows out of web services
    - is a front-end for accessing and processing workflows and service descriptions
    - provides workflows with input data
    - translates textgrid workflows (TGWF) into Petri Nets used by the engine (GWDL)

2.7.1. Technical Information

- The GWES engine runs in a Tomcat servlet container. All endpoints are hidden by firewall / Apache access rules. Persistence is given by using an eXist XML database instance
- The GWES Proxy is a PHP SOAP Web Service application. It provides the following operations externally:
    - getMyWorkflowIDs (auth, log) – returns a list of Workflow Jobs including workflowID, textual description, and property list of the user owning the SessionID given by auth
    - uploadWorkflow (auth, log, gwdl, targetproject) – takes a GWDL workflow document (a string containing XML) and starts the workflow under the user's name. Output data is written to the specified target project
    - abortWorkflow (auth, log, workflowID) – cancels and removes a workflow job from the engine memory
    - getWorkflowDocument (auth, log, workflowID) – returns the GWDL of the current workflow state (for a completed Workflow usually with in-

line TextGrid URIs to be evaluated further)

- Each TGWF workflow composed using the TextGridLabs workflow wizard will be transformed into GWDL by using a fixed skeleton where CRUD operations are prepended and appended and Metadata of new TextGridObjetcs is being created by a rule from the input objects.
- Service Descriptions and TextGrid Workflow descriptions are formally described using XML schema and used in the TextGridLab using the JAXB XML Binding.

2.7.2. URLs

The GWES Workflow Engine is used unmodified and can be retrieved from:
http://www.gridworkflow.org/kwfgrid/gwes-web/

Subversion Repository

GWES Proxy: https://develop.sub.uni-goettingen.de/repos/textgrid/trunk/middleware/workflow/GWESProxy

TextGridLab Workflow Code: https://develop.sub.uni-goettingen.de/repos/textgrid/trunk/lab/workflow/info.textgrid.lab.workflow

WSDL

Current stable repository: https://textgridlab.org/1.0/workflow/GWESproxy.wsdl

XSD

The following XSD files include documentation annotation:

Service Description: https://develop.sub.uni-goettingen.de/repos/textgrid/trunk/lab/base/info.textgrid.lab.workflow/resources/ServiceDescription.xsd

TextGrid Workflow description: https://develop.sub.uni-goettingen.de/repos/textgrid/trunk/lab/base/info.textgrid.lab.workflow/resources/TGWFv6.xsd

The latter is being superseded by a slightly modified version, however, the documentation still applies:
https://develop.sub.uni-goettingen.de/repos/textgrid/trunk/lab/base/info.textgrid.lab.workflow/resources/tgwf2_v0.xsd

2.8. TG-import

The External Import Tool serves to import data either into the TextGridLab or directly into the TextGridRep. There are different policies to import data:

- Data may be imported just by copying files and folders into a hotfolder; the required TextGrid metadata files will be created automatically, a given folder structure will be obtained using TextGrid aggregations.
- A DFG-Viewer METS file may be provided (for the time being, download and transfer to the hotfolder required): All linked files will be downloaded; the structure of the two StructMaps (logical and physical) will be imported as TextGrid aggregations.
- If you are able to provide TextGrid metadata files as well, the complete import policy may be used. No metadata files will be created automatically. This procedure requires knowledge of the TextGrid Metadata XSL Schema and the concept of TextGrid Objects.

All three policies may be used to import data into the TG-lab or the TG-rep; only the appropriate service endpoints need to be configured. Importing into the TG-lab means that data only is visible to the owner (and to other users which have been assigned reading and/or editing permission by the owner); data can as well be edited and worked with within a TextGrid project in the TG-lab and published later on using the TG-publish GUI. If you import into the TG-rep, your data will be published into the TextGrid Sandbox to be verified before publishing data into the TG-rep is enabled.

In any case a TextGrid project ID and an RBAC session ID are required, both provided in the TextGridLab. Using a TextGrid (or Shibboleth) account, a user can log in into the TextGridLab and create TextGrid projects, check out their project ID and get a session ID from the menu Help > Authentication.

2.8.1. Import Using SVN and Maven with Eclipse

Prerequisites

Please have Apache Maven and an Apache SVN client ready to use.

Checking out the koLibRI TextGrid Addon Module

```
svn checkout https://develop.sub.uni-goettingen.de/repos/kolibri/tags/2012-08-29_
kolibri-textgrid-import_TG2.0 kolibri-addon-textgrid-import
```

Building an Eclipse project

```
mvn eclipse:eclipse
```

Import the folder (containing the pom.xml file) to Eclipse using file > import > existing project into workspace.

Configuring the koLibRI

See "Configuration" below.

Running the koLibRI

Add an Eclipse runtime configuration:

- Create a new Java application in the Eclipse Runtime Menu.
- Use de.langzeitarchivierung.kolibri.WorkflowTool as main class.
- Add -c [path to chosen config file].
- In case you an increase of memory capacity of the VM is required, please use for example -Xmx1024m.

Run the Application.

Imports can be verified either in the imported TextGridLab's project or in the TextGridRep Sandbox, depending on configuration.

2.8.2. Import Using SVN and Maven via Command Line

Prerequisites

Please have Apache Maven and an Apache SVN client ready to use.

Checking out the koLibRI TextGrid Addon Module

```
svn checkout https://develop.sub.uni-goettingen.de/repos/kolibri/tags/2012-08-29_
kolibri-textgrid-import_TG2.0 kolibri-addon-textgrid-import
```

Building the module

```
mvn package
```

Configuring the koLibRI

See "Configuration" below.

Running the koLibRI

Run with your chosen configuration file:

```
mvn exec:java -Dexec.args="-c path/to/config_file.xml"
```

Imports can be verified either in the imported TextGridLab's project or in the TextGridRep Sandbox, depending on configuration.

2.8.3. Import Using the koLibRI JAR File

Downloading the software

Please download the configuration and folder ZIP file containing the required config files and templates first: kolibri-textgrid-import.zip

Extract to your preferred working folder. The koLibRI Command Line Module will be neccessary, prepared for the usage of the TextGrid import. Please also put this JAR into your working folder: kolibri-cli-2.0-SNAPSHOT.jar

The folder structure should now show as follows:

```
kolibri-textgrid-import
  | config
    | tglab_config.xml
    | tgrep_config.xml
    | policies.xml
    | ...some more koLibRI config files...
  | folders
    | dest
    | hotfolder
      | data
    | log
    | temp
    | work
  | kolibri-cli-2.0-SNAPSHOT.jar
```

Configuring the koLibRI

See "Configuration" below.

Starting the koLibRI workflow tool

After correct configuration and completed data copying, koLibRI can be started, requiring a Java Virtual Machine using Java 6. Change into your work directory containing the JAR and the config files and type:

```
java -jar kolibri-cli-2.0-SNAPSHOT.jar -c config/tglab_config.xml
```

Imports can be verified either in the imported TextGridLab's project or in the TextGridRep Sandbox, depending on configuration.

2.8.4. Configuration

Chosing configuration file from template

There are two template configuration files in the config/ folder:

- **tglab_config.xml**
  To be used to import data into the TG-lab, so data can be worked with within the chosen TextGrid project. The data will not be visible to users other than the owner and to users which have been assigned reading and/or editing permission by the owner. All non-public services are preconfigured in this file.
- **tgrep_config.xml**

to be used to import directly to the TG-rep. Data is visible to the public immediately (first in the TextGrid Repository Sandbox, after ultimate publishing for the public).

Please choose one of the files according to your import plans.

2.8.5. Editing the Config File

Commonly used settings

- **<field>defaultPolicyName</field>**
  Setting the import policy: The parameter defaultPolicyName can address the following policies (as existing in the policies.xml file): Edit the config file of your choice, and choose a value. Depending on your import policy, other configuration values need to be set, please see below.
  - **aggregation_import**
    This policy is used to automatically create TextGrid metadata for each file out of the file name and the detected file format. For every folder, a TextGrid aggregation is created and imported, so the folder structure will appear in TextGrid as in the import folder itself.
  - **complete_import**
    Using this policy, all given files will be imported without additional metadata being created, so a complete set of TextGrid objects including TextGrid metadata will be required, containing TextGrid URIs (e.g. by using the TG-crud's method #GETURIS and arrange them accordingly). File extensions for existing TextGrid editions, collections, works, aggregations, XML and metadata files may be configured if needed.
  - **continue_import**
    Use this policy to continue a broken or stopped import (e.g. in case of an error). Configure the hotfolder to be the temp folder the files were processed in.
  - **delete_import**
    A set of objects already imported can be deleted from the sandbox again. This procedure may be used with an URI list (as a file) or by giving a root URI. Please see configuration of the class DeleteFiles.
  - **publish_import**
  - An set of objects already imported will be ultimately published. Uses the TG-publish service.
  - **dfgviewermets_import**
    Takes as input one (or more) DFG Viewer METS files according to the DFG Viewer METS Specification, creating a folder structure from the physical and logical StructMap which subsequently will be imported into the TextGrid. MODS and/or TEI metadata will be mapped to TextGrid metadata via existing MODS/TEI XSL transformation files, or via custom XSL files.
- **<field>tgcrudServerUrl</field>**
  Choosing the TG-crud service: Depending on your chosen import location (TG-lab or TG-rep) the TG-crud endpoint will already be configured correctly.
- **<field>rbacSessionId</field>, <field>projectId</field>**
  Authentication and project settings: Add the two values with your TextGrid Project ID (projectId) and your Session ID (rbacSessionId).
- **<field>logParameter</field>**
  TextGrid logging: No entry required (currently no longer in use)

- **<field>getPids</field>**
  PID generation: If set to TRUE, persistent identifiers will be generated for any TextGrid object using the GWDG Handle service. Only useful if data is ingested directly into the TG-rep.

Aggregation import configuration

Using aggregation_import, set the data as described above and run the koLibRI.

- **<field>hotfolderDir</field>**
  Choosing a hotfolder: As hotfolder ./folders/hotfolder/data/ is pre-configured. Copy your data to publish there. The data is copied before processing starts, so the original data will not be touched. The first aggregation will be the subfolder in data/ (only using aggregation_import).
- **<field>useBaseUrisInAggregations</field>**
  Base URIs: With policy aggregation_import, one can choose if TextGrid base URIs are put into the generated aggregations (e.g. textgrid:1234), or the generated absolute URIs (e.g. textgrid:1234.0). The latter would be mandatory if directly importing into the TG-rep.

Complete import configuration

Using complete_import, just set the data as described above and run the koLibRI.

- **<field>hotfolderDir</field>**
  Choosing a hotfolder: As hotfolder ./folders/hotfolder/data/ is pre-configured. Copy your data to publish there. Data will be copied before processing starts, so the original data will not be touched. All data will be imported exactly as prepared by the user.

DFG Viewer METS import configuration

Using dfgviewermets_import, set the data as described above and run the koLibRI.

- **<field>hotfolderDir</field>**
  Choosing a hotfolder: As hotfolder ./folders/hotfolder/ data/ is pre-configured. Put all your METS files there. For each METS file a root aggregation/edition/collection will be created, please see below. It is possible to put more than one METS file into the hotfolder. koLibRI then processes the import concurrently with a configurable number of threads (please see general configuration options in the koLibRI configuration file).
- **<field>rootAggregationMimetype</field>**
  DFG Viewer aggregations: For DFG Viewer Import you can chose the format of your root aggregation (there is one root aggregation for every METS file). It can be chosen to be imported as a TextGrid Aggregation (text/tg.aggregation+xml), Edition (text/tg.edition+tg.aggregation+xml) or Collection (text/tg.collection+tg.aggregation+xml). Custom XSLT stylesheets for metadata creation can be specified in the properties of <class name="actionmodule.textgrid.DfgViewerMetadataProcessor">.

Publish configuration

For final publishing of objects (only after sandbox publishing), use the policy publish_import

- **<field>hotfolderDir</field>**
  Change the hotfolderDir value to the temp folder of the import process to delete in the hotfolder, e.g. ./folders/temp/1318521646580_data/. Absolute pathes will work as well.

  URIs will be taken from the (at import time) created URI mapping file *.IM.tsv stored in the folder.

Delete Configuration

Data already published can be deleted if imported into the TextGrid Sandbox before. Change the policy to delete_import.

- **<field>deleteViamappingFile</field>**
  Set to TRUE if you want to use a mapping file created at import. Set hotfolderDir parameter accordingly. Set to FALSE if you want to use the URI of your root object, that predecessors shall be deleted. Requires setting of rootUri.
- **<field>hotfolderDir</field>**
  Change the hotfolderDir value to the temp folder of the import process to delete in the hotfolder, e.g. ./folders/temp/1318521646580_data/. Absolute pathes will work as well. The absolute path to the .IM.tsv file itself cannot be used.

  URIs will be taken from the (at import time) created URI mapping file *.IM.tsv stored in the folder.
- **<field>rootUri</field>**
  Set the root URI to e.g. textgrid:1234.0 to delete anything referenced by that root aggregation (recursively).

2.8.6. Logging and Keeping Mapping Information Files

All imports are logged to /folders/log/. Keep all folders in the /folders/temp/ folder, as well as any *.IM.tsv files for later publication or deletion policies. If PIDs are created, the PID mapping is stored to *.PD.tsv files.

Change further parameters

Further change of parameters is not recommended.

3. TextGrid Repository Outreach

The TextGrid Repository was released in Version 2.0 and this productive installation is running stabe and quite performant. Three main issues will be adressed in TextGrid 3:

High availibility of all machines and services will be further examined and developed – including e.g. load balancing, usage of STONITH, parallel instances of services, etc. Performance and scalibility will be addressed (e.g. using the eXist database only for XPath queries and evaluate other database systems for metadata and fulltext search). Finally, one main goal of the third project funding period of TextGrid will be to build up a sustainable TextGrid infrastructure.