TextGrid Manual: Tool Development

Version - 24.2.2008
Work Package - AP 3
Responsible Partner - SUB, DAASI

TextGrid

modular platform for collaborative text editing - a community grid for the humanities

Project: **TextGrid**

Part of D-Grid and the German e-Science Initiative

Funded by the German Federal Ministry of Education and Research (BMBF)
by Agreement 07TG01A-H

Project Duration: February 2006 - May 2009

Document Status: final

Distribution: external

Authors:

Andreas Aschenbrenner, SUB
Stefan Funk, DAASI
Martin Haase, DAASI
Roman Hausner, SUB
Christoph Ludwig, FH Worms
Wolfgang Pempe, Saphor
Ubbo Veentjer, SUB
Thorsten Vitt, TU Darmstadt

Table of Contents

1. introduction

TextGrid[1] strives to be an open ecosystem, where just any e-Humanities initiative can reuse existing functionality, add and share functionality, or tailor a research environment to specific needs and scientific processes. This document describes software developers how to achieve all this.[2]

TextGrid has rich functionalities to be reused,[3] and great efforts are being taken to ensure interoperability with initiatives around the world to further extend that portfolio. With sharing your services and application patterns you hence contribute to the global e-Humanities. At the same time, it is your task to hide the wealth and complexity of the ecosystem from the user, by building tailored research environments.

For tools, there are fundamentally two entry points to TextGrid: (1) a specialised web service (e.g. a lemmatiser, a semantic annotator) can be linked into the service network; (2) applications are built upon the service network and tailored to the needs of its dedicated user group (TextGrid's primary application environment is an Eclipse Rich Client).

Following the outline of the document, you will first learn about the general concepts and how to embed your own web services into the TextGrid environment. Each chapter takes you one step further into TextGrid, with the last chapter describing programming in TextGrid's Eclipse-based application environment. However, if you primarily mean to build your own interactive application using TextGrid's grid abstraction in Eclipse, you may choose to start with the very last chapter jumping to earlier parts of the document as needed.

Accompanying this document is a programming **tutorial**, which describes in detail both the service layer as well as the Eclipse environment for TextGrid.[4] To achieve this, the tutorial creates in a number of steps first, a simple web service for extracting names out of published TextGrid objects, as well as an Eclipse application, which interlinks a TextGrid object, the list of extracted names, as well as the Wikipedia entries for each of those entries. Please use this general documentation in tandem with the tutorial for quickly picking up on the TextGrid development environment.

[1] More information about TextGrid can be found at www.textgrid.info

[2] For adding your digital assets into TextGrid (e.g. collections of digitisations and/or TEI-annotated texts), please refer to the respective TextGrid Manual on www.textgrid.info (to be published January 2009).

[3] Existing services are described in the document "TextGrid components" published on the TextGrid website (for now only in German). Further strengthening of the TextGrid community will include the construction of a community portal, where anybody can describe their services and application patterns.

[4] see the TextGrid documentation at www.textgrid.info for the tutorial

2. an architecture overview

The TextGrid architecture is layered, with the lower layers being long-term stable and the higher layers being as flexible as possible. Entry-points are available on all layers, with maximum openness on top and increasingly rigid constraints towards the basis. Starting development at the topmost layer and successively advancing towards more stability in the infrastructure is recommended.

This architecture approach was built on the following principles:

- to provide an **open, generic infrastructure** - Functionalities are only constrained where indispensable; the application context of a service is not pre-empted. Thereby, services can be re-used in different contexts and re-mixed for efficient development of novel applications.

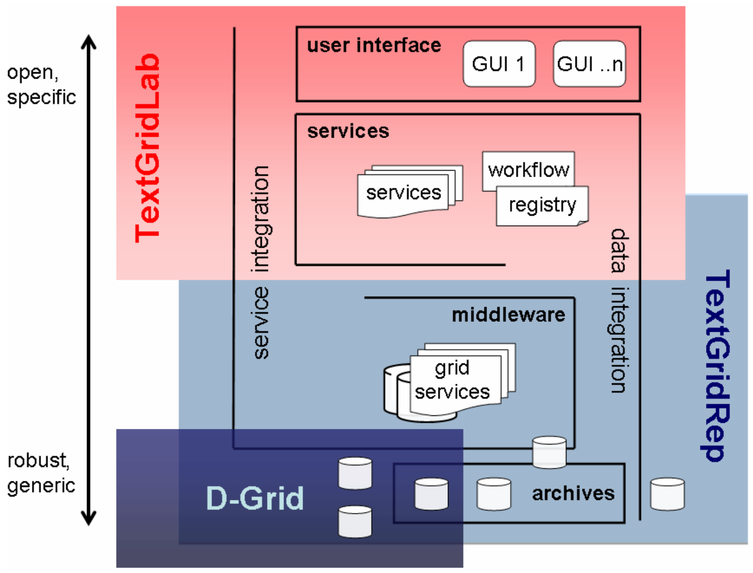


Figure 1 - TextGrid architecture scheme

- to foster **specialized applications** and semantically deep processing - Specialized workflows, targeted goals and unique contexts demand formats, metadata, and interfaces to be freely adapted. TextGrid allows for this. However, there may be different levels of support and interoperability for specific formats, metadata schemata, or interfaces. Thus, TextGrid users can embed their specific requirements in TextGrid at various levels of integration.

- to encourage **participation** - Community participation is crucial to obtain a growing base of scientific texts, service variety, and ultimately the sustainability of the infrastructure. Participation in TextGrid is possible on multiple levels, from tentative usage to active contribution and partnering.

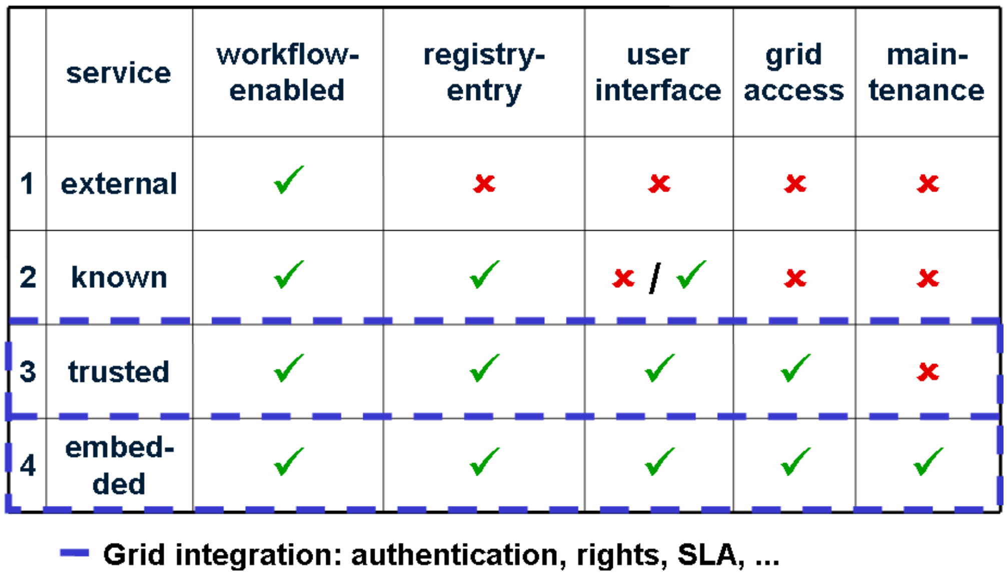
Technologically, these guiding principles are translated into the following conceptual layers. Being a service-oriented architecture, each layer itself may consist of multiple, distributed components and may evolve with a changing technological and organisational context.

- **application environment** - virtual research environments tailored to specific user needs.. The main application environment in the initial project phase is Eclipse-based and geared towards use in philology. However, other user groups or workflows may call for other environments.

- **services** - building blocks of specialized functionality. Atomic functionalities such as tokenization, lemmatizing, or collation are wrapped into individual services to be re-used by other services or plugged into an application environment. A growing community contributes to this tool-kit of interoperable services.
- **middleware and archives** - generic utilities for a stable core. The basic building blocks of the TextGrid infrastructure comply with the interoperability framework for services, however they offer more generic functionality at increased stability and scalability.

3. service environment

In order to foster interoperability, integration into the service environment is possible on several steps. The first step of integration is rather loose, with growing requirements as well as growing possibilities with each additional step. For example, authentication is necessary from the third step on. Each initiative can choose how far they advance into the TextGrid service network.



	service	workflow-enabled	registry-entry	user interface	grid access	main-tenance
1	external	✓	✗	✗	✗	✗
2	known	✓	✓	✗ / ✓	✗	✗
3	trusted	✓	✓	✓	✓	✗
4	embed-ded	✓	✓	✓	✓	✓

– Grid integration: authentication, rights, SLA, ...

Each kind of service (rows) implements specific properties (columns). The following properties are hardly dependent on who hosts a service, and where - anybody, anywhere is invited to join in:

a. **workflow-enabled**
The service can be mashed up with other services in a workflow. The workflow editor in the TextGridLab may embed the service in a comprehensive batch process. Basically any programming language and host environment can be used for the services, yet web service standards form the basis for interoperability.

- WSDL 2.0, SOAP 1.2 (document-literal style) or REST
- character encoding: UTF-8

b. **registry-entry**
The installation of a service registry is planned for the future. This registry allows the documentation of services and their interfaces. The more generic a service is designed, the better for its reuse. For example, a lemmatiser for a specific language should avoid prescribing a specific XML/TEI schema; rather, it can take any stream of text and return the text with respective annotations for the lemmatised words, ideally enabled to parameterize the output markup or at least employ the same tags as other lemmatisers in the service network do.

- provide sufficient documentation with your service
- learn from other services and follow conventions
- test your service exhaustively, and provide feedback-mechanisms for users to report bugs

c. **user interface**

A graphical user interface is available and it can be embedded (with other services) in user environments. Initially, the premier user environment is an Eclipse-based client, though web-based or other clients may follow. Actual user environments may be tailored to a specific user group (e.g. discipline, project), thus each service/tool may be embedded in various user environments.

- the user interface (resp. each user interface, e.g. Eclipse-based, web-based, etc.) embeds nicely into the respective environment and follows its conventions. Documentation can be found at the respective user interface project. See chapter for the Eclipse-related documentation.

d. **grid access**

The service carries all the necessary information to be allowed into the grid. This includes information for authentication and logging. TextGrid is Shibboleth-enabled and will interconnect with the national scientific Shibboleth federation DFN-AAI once productive. For the time being, users can register at TextGrid directly to be granted access. Grid access entails the possibilities to join projects, deposit and share (private) digital objects, and similar activities. In the future, licensing policies may build on authentication information.

- the first parameter of the service interface (WSDL) is a String parameter *auth* to carry authentication information. When communicating with TextGrid services, this parameter should be duly filled with suitable data from (1) passing the authentication information from one service to the next[5], (2) request credentials from TG-auth* directly, or (3) employ e.g. the authentication module in Eclipse to mediate the credentials (incl. the necessary user interaction).[6]
- the second parameter of the service interface (WSDL) is a String parameter *log* with logging information: string containing the URL of the Logging-Service, SessionID for the Logging-Service, Loglevel. The Loglevel can be Loglevel: 0 - Disabled 1 - Error 2 - Warning 3 - Info 4 - Debug.[7]

e. **upkeep**

With the measures a-d, the service implements all requirements for interoperability with other services in the e-Humanities service network. This includes TextGrid services, as well as services from Interedition[8] and other sources. Beyond technical

[5] Unfortunately, mechanisms for the WSDL Header to pass on service meta-data like authentication and logging are not sufficiently implemented in all Frameworks. Our experiments with e.g. Python were not satisfying.
[6] Refer to chapter .
[7] Refer to chapter .
[8] Interedition. http://interedition.huygensinstituut.nl/

measures, however, the availability and maintenance of the service needs to be ensured.

- trusted partners with good documentation of their services and a stable organisational environment are awarded this label.

4. initialization

For initializing the TextGrid environment, there are two steps to take: (a) connect to the TextGrid configuration service, and (b) initialize the logging service. The configuration service provides endpoints of the infrastructure services (e.g. where to fetch files from, where to send logging information). There is a separate configuration service for each environment: the productive environment and the test instances.

4.1. configuration, Java client library

For use in Java client code (for Eclipse plug-ins, see the subsequent section), we provide a convenience library to access the config service. The jar file is available at https://develop.sub.uni-goettingen.de/repos/textgrid/trunk/middleware/confclient/

For a query to the config service, you only need to instantiate the ConfservClient class and call one of the methods getValue() and getAll(). The class ConfservClientConstants provides String constants that can be used as arguments in the calls to getValue() and as keys into the resulting HashMap of getAll().

```
import info.textgrid.middleware.confclient.ConfservClient;
import info.textgrid.middleware.confclient.ConfservClientConstants;

...

ConfservClient csc = new ConfservClient
("http://textgridlab.org/axis2/services/confserv");
String tmp = csc.getValue(ConfservClientConstants.LOG_SERVICE);
```

After above code snippet, `tmp` may hold the value "`http://textgridlab.org/axis2/services/textlog`". We will describe the log service in more detail below.

4.2 configuration, Web Service-based

First of all, you need to establish, where all the components of the TextGrid API are. For this, you only need a single point of contact, the configuration service (of the productive environment) at

```
https://textgridlab.org/axis2/services/confserv
```

This service is your gate to all the infrastructural environment information you may need, and hence the only hard-coded endpoint in your code.[9] The service is available as SOAP-based (SOAP 1.1/2.0) and RESTful. You can either retrieve all available endpoints

[9] Besides the convenience factor, we may in the future establish multiple entry-points and dynamically allocate those, which are close to the very user or those with less load. The configuration service will then allocate the most suitable service.

(`getAllValues`), or request one specific endpoint (`getValue`). The most important available endpoints are defined by the following constant strings:

- `logservice` ... for logging; see chapter
- `tgauth` ... authentication - get a session ID; see chapter
- `tgsearch` ... metadata and XML content search, for retrieval; see chapter
- `tgcrud` ... ingest and retrieve digital objects; see chapter

In addition to the service endpoints, the service also tells you the current TextGrid object metadata namespace and the location of the corresponding schema file:

- `ns` ... namespace of TextGrid data
  http://textgrid.info/namespaces/metadata/core/<date of version>
- `schema` ... the XML Schema for the TextGrid core encoding
  www.textgrid.info/schemas/textgrid-metadata_<date of version>.xsd

For those services that run over a particularly long period, you may want to establish a mechanism for re-retrieving the environment setting. We recommend a pattern using org.apache.axis2.engine.**ServiceLifeCycle** - for closer instructions please refer to an Axis2 manual of your convenience.

4.3 configuration, in Eclipse

We provide a Eclipse Plug-in that integrates a client for the config server into Eclipse and Eclipse-based Rich Clients. The singleton `info.textgrid.lab.conf.client.ConfClient` in the Plug-In available at

```
https://develop.sub.uni-
goettingen.de/repos/textgrid/trunk/lab/info.textgrid.lab.conf/
```

offers the basic functions to connect to the configuration service and access the current settings there. Download and enable this plug-in in your client.

```
...
import info.textgrid.lab.conf.client.ConfClient;
import info.textgrid.middleware.confclient.ConfservClientConstants;
...

try {
  ConfClient confClient = ConfClient.getInstance();
  String value =
confClient.getValue(ConfservClientConstants.LOG_SERVICE, false);
} catch (RemoteException e) {
  // do something
}
```

The plug-in adds an entry in your preferences dialog that allows you to specify the endpoint of the config service.

4.4. logging, Web Service-based

TextGrid caters for a dedicated logging mechanism, which enables chained logging across multiple services no matter where those services are hosted and what their system environment is like. A logging server receives and stores logging messages from just anywhere via a Web Service interface. A logging session ID then identifies a chain of services in a specific workflow. The session ID is passed along with other information as the second WSDL parameter from service to service.

In the following we describe a typical logging session:

First, a client starts a new logging session with the logging service's initialize operation. This operation that takes no input data returns a unique logging ID that clients need to provide in all subsequent requests.

Clients then start putting information into the log by calling the "log" operation. The log operation's input message consists of the log ID that was previously issued by the initialize operation, a string identifying the origin of the log message, and the log message proper. "Client" can either be the party that originally called the initialize operation or any other service to which it passed the logging ID. By convention, services that are aware of the TextGrid logging mechanism get this information (dubbed "loginfo") as one of their input parameters in form of a space separated list of strings comprised of the logging service's endpoint, the logging ID, and a log level indicator that indicated which events the service is supposed to log.

The log can be read by the getLogFragment operation. This operation expects the logging ID and a log index (of the XML Schema type int) and responds with a list of log strings that includes all log messages starting at the given index up to the most recent log entry. Each log string consists of a timestamp, the origin and the actual message.

Finally, the client is supposed to call the endSession operation that deletes the complete log and releases all associated resources allocated by the logging services. If a client fails to call endSession, then the resources are freed automatically 24 hours after the last read or write access to the log identified by the logging ID.

4.5. logging, Java Client library

We provide a convenience library for logging in Java-based environments. This library is intended for use in web services that receive the loginfo (containing logging ID, etc - see above) from their caller. Consequently, it does not provide methods to read log entries. The library can be retrieved from

```
https://develop.sub.uni-
goettingen.de/repos/textgrid/trunk/middleware/textgridlogger/TextgridLog
ger.jar
```

The following sample code presumes that the loginfo is stored in a String variable of the same name.

```
import info.textgrid.middleware.textgridlogger.*;
...
```

```
   TextgridLogger logger = new TextgridLogger(loginfo);
   logger.log("log message", "this.servicename", loglevel);
   ...
```

where

```
   String loginfo - you'll get the loginfo string passed on from the caller
   (the TextGridLab or another service) as a second parameter
   int loglevel - 0/Disabled  1/Error  2/Warning  3/Info  4/Debug
```

4.6. logging, Eclipse-based

The Eclipse Plug-in `info.textgrid.lab.log` provides mechanisms for logging and for viewing previously logged information. On initialization, it connects to the log service and initializes a new log session. The thus generated loginfo can be passed on to child services for logging.

```
   ...
   import info.textgrid.middleware.textgridlogger.*;
   import info.textgrid.lab.log.logsession;
   ...
   logsession log = logsession.getInstance();
   String loginfo = log.getLoginfo();
   ...
```

where

```
   String loginfo - to be passed on as the second web service parameter;
   ( <URL-logging-server> + " " + <session-ID> + " " + <loglevel> )
```

Apart from this API, the plug-in also offers a user interface to set the right log level and view log messages. The corresponding view has the ID "info.textgrid.lab.log.views.logview" and can be found in Window/Show View/Other ... with the entry "Log Category - Log View".

5. TG-auth*

TG-auth* consists of two parts. With "N" replacing the asterisk, it enables for authentication of users in the TextGrid environment. With "Z", it serves as a powerful authorisation engine properly defining who may access resources.

As for authentication, an infrastructure based on the Internet2 software *Shibboleth* is used. For authorization, we use a role-based access control solution called *openRBAC* where permissions are stored in an LDAP database. The whole process of authenticating and authorizing access to a specific resource is as follows:

- the TextGridLab-internal browser (or an external one) is pointed to an internet resource (in the following, the *WebAuth* resource) protected by a Shibboleth Service Provider (SP).
- before allowing access to the WebAuth resource, the SP redirects the browser to a list where the users can select their home institution (or Identity Provider, IdP). After selection the browser is redirected to the login page of the IdP. On successful login the IdP redirects back to the SP and attributes are sent, too. We use the attributes `eduPersonPrincipalName` (ePPN, form:), `givenName` and `sn` (surname)
- now the WebAuth resource processes the request which includes:
  - adding the user by their ePPN to the RBAC database (if not present yet)
  - creating a SessionID and activating all roles the user has in RBAC
  - delivering a result page to the browser with SessionID and ePPN both human-visible and machine-readable in the headers
- finally, the SessionID can be used by services in the lab to represent the user. The ePPN is used currently for display purposes.
- authorization now amounts to check if
  - a given resource can be treated with
  - a given operation
  - given the roles activated for the presented SessionID.

  There are several functions dedicated to this purpose both in the TextGrid-specific layer built for openRBAC and in its core functions.
- Besides these functions for processing authorization requests and returning access decisions, both the openRBAC core and the TextGrid-specific layer contain numerous functions for managing roles, permissions, resources, users and sessions.

5.1. Obtaining Authentication Information

In the TextGridLab, the Singleton class info.textgrid.lab.authn.RBACSession manages all issues relevant to authentication. Thus when a service needs to know the SessionID of the current user, the following code can be used:

```
import info.textgrid.lab.authn.RBACSession;
...
String sessionID = RBACSession.getInstance().getSID(false);
String ePPN = RBACSession.getInstance().getEPPN();
...
```

Calling `getSID(false)` returns the SessionID currently active. This could also be the null string in the case the user had not authenticated before. Contrary to that, `getSID(true)` first checks this and, if empty, opens a modal (i.e. GUI-blocking) dialogue that lets the user authenticate. This dialogue basically holds the browser which points to the WebAuth resource located at the URL the Config Server returns for the key ConfservClientConstants.AUTHZ. It is also possible to override the value `getSID()` returns by setting the environment variable `TEXTGRIDLAB_SID` before starting the Lab.

5.2. interacting with the authorization system - TG-authZ, Web-Service-based

The RBAC functions are defined at the WSDL at https://textgridlab.org/tgauth/rbacSoap/wsdl/tgextra.wsdl. The actual parameters and return values can be read from the WSDL, whereas we shortly describe the meaning of each function in its respective category here. The RBAC endpoint of the currently active TextGridLab instance can be obtained from the instance's configuration server, using key ConfservClientConstants.TG_AUTH.

users and sessions	authenticate	Internal function used by applications to identify themselves (via shared secret between RBAC and WebAuth).
	userExists	Checks whether this ePPN exists in RBAC.
	getSid	Returns some random SessionID suitable for RBAC.
roles	createProject	Creates a project. Projects are roles with various sub-roles, i.e Leader, Administrator, ... Creates default roles with default permissions which can be adapted afterwards.
	tgAddActiveRole	Activates a role for a session.
	tgDropActiveRole	De-activates a role.
	tgAssignedRoles	Returns the roles the requesting user has, in any project. Can be called for another user by project leaders, then roles will be limited to that project.

	tgAssignedProjects	Returns projectIDs of all projects the user has any role in.
	getAllProjects	Returns all project IDs stored in this RBAC instance.
	addMember	Project leaders can assign users into specific roles.
	deleteMember	Delete a role from a user.
	getMembers	All members in the project, caller must be member.
	getProjectDescriptio n	Name and description of project(s) identified by ID(s).
	deactivateProject	Deactivated projects cannot be modified or read anymore (except published resources). However, information is preserved so that the project can be re-activated by manual modification of database.
	getLeader	Returns Project Leader/s (i.e. who have *delegate* right on the respective *project* resource).
resources[10]	(registerResource)	Registers a resource in the database. Assigns standard permissions to roles in project which can be adapted afterwards. User needs *create* right on project resource. **Only for use by TG-crud**.
	(unregisterResource)	Removes resource from database. User needs *delete* right on resource. **Only for use by TG-crud**.
	getOwner	Returns ePPN of owner that was set in registerResource.
rights	tgGrantPermission	Enables given operation for given role on given resource. Resource may be normal (file) or project. Users need *delegate* right on resource or project.
	tgRevokePermission	Disables this permission, see tgGrantPermission for limitations.
	getRights	Returns permissions for given resource or projectID. Can be used by project leaders to check permissions for another project member.
	publish	Sets the isPublic flag to TRUE. The resource will then be world-readable. This is irreversible.

[10] The resource operations, especially registerResource and unRegisterResource, are not API and **must not be used by clients**. Clients (including services) should call TG-crud.create and TG-crud.delete instead, see section Fehler: Referenz nicht gefunden.

	isPublic	Returns status of isPublic flag.
access control	getObjects	Returns URIs of all resources in project. Caller must have some role in the project.
	tgCheckAccess	Returns access decision for given operation on given resource for session.

6. TG-search

TG-search provides several interfaces for text retrieval and metadata search. To understand what TG-search does, it might be useful to have a look on the data lifecycle in TextGrid:

Once created (or modified), a dataset and its metadata are stored in the grid. This is done by TG-crud (method “write” or “update” – see below). But that’s not all what happens with the data. For the purpose of project-specific as well as cross-project search the metadata and structure data is also stored in an XML database[11]. In order to enable cross-project search, every dataset of structure data is also stored in an extra instance which is transformed to a certain uniform, TEI-based markup, the so called Baseline Encoding[12].This transformation is performed by a specialized TG-crud module, the Adaptor Manager[13].

TG-search per default enables access to these two collections:

```
/db/metadata
/db/structure/baseline
```

Project-specific encoded data is stored in

```
/db/structure/original
```

and is accessible via individual XQueries or a specialized instance of TG-search.

The following description covers only the combined search over the two first mentioned collections, which is internally performed by parameter-generated XQueries. Also, it describes exclusively the Web Service layer (SOAP and REST). The Research-tool in the TextGridLab (described in the user documentation) builds upon the web services as described here.

[11] Currently deployed: eXist 3.0 (http://exist-db.org)
[12] See the documentation to the TextGrid Baseline Encoding; available on www.textgrid.de
[13] See also Report 3.2, "Textannotationen und Adaptoren"; available (in German) on www.textgrid.de

6.1. Parameters

Name	Description	Examples (GET)
sid	the session ID provided by TG-auth* (see chapter Fehler: Referenz nicht gefunden, AuthN)	
log	the log string (see chapter)	
q	Query string for (full) text search.	q=hier
sd	structure data, names of elements (separated by "|") constraining the text search	`sd=p|head`

md	metadata; Retrieval of metadata, matching datasets and/or constraining a text search to datasets complying to the specified `metadata`. Key-value pairs are connected with a colon, if there are more than one, they have to be separated using "|". You can filter for certain elements by using "$" instead of a string value. Searching for attribute values is also possible: `agent@role:goe@author` finds `<agent role="author">Goethe</agent>`	`md=title:wörterbuch|agent:campe|uri:$`
		`md=agent@role:campe@author`
		`md=agent:goe schill grass|agent:$`

mdns	metadata namespace; overrides the default metadata namespace. Useful for retrieval of legacy data.	`mdns=http://textgrid.info/namespaces/metadata/core/2008-03-13`

xpath	Search for text and element nodes in
	• baseline-encoded textual/structure data
	• core metadata
	resp. in the particular namespaces.
		`xpath=//div/head`
		`xpath=//agent[@role='author']&opt=metadata:1`

xquery	Searching via XQuery, only makes sense using POST	`See below (POST)`

dbparams	Mirrors the parameters to handle the server-side result-sequences of the REST-interface of eXist (http://www.exist-db.org/devguide_rest.html#N1030B): `start:` index of first item in the result sequence to be returned `max:` maximum number of items to be returned `session:` eXist session id as returned by previous request	`dbparams=start:1|max:10|session:1`
opt	Additional features for • Text query (parameter **q**): key-word-in-context-mode: `opt=kwic:1` • Metadata query (parameter **md**): auto-completion of result string: `opt=ac:1` • XPath query (parameter **xpath**): search in metadata: `opt=metadata:1`	`q=am&sd=head&opt=kwic:1` `md=agent@role:goe@author&opt=ac:1` `xpath=//uri&opt=metadata:1`
rel	< the relational database has a variable query mechanism at this point >	

Currently supported combinations of parameters[14]

```
q, (md, mdns?)?, sd?, dbparams?, (opt=kwic:1)?
md, mdns?, dbparams?, (opt=ac:1)?
xpath, dbparams?, (opt=metadata:1, mdns?)?
xquery
```

[14] in DTD style – optional elements marked with "?", `sid` and `log` not considered

6.2. REST interface

The most effective way to approach TG-search from outside the TextGridLab is to use the REST interface via POST. The query parameters have to be modelled as XML fragment, e.g.

```
<tg:meta xmlns:tg="http://meta.textgrid.de">
   <tg:sid>RBAC_SESSION_ID</tg:sid>
   <tg:log>LOG_STRING</tg:log>
   <tg:md>agent:goe|agent:$|type:text|type:$|uri:$</tg:md>
   <tg:q>ich</tg:q>
   <tg:opt>kwic:1</tg:opt>
   <tg:dbparams>start:1|max:100</tg:dbparams>
</tg:meta>
```

The service endpoint is
`http://textgridlab.org/axis2/services/Metadata/`

For an example REST client implementation based on Axis2-1.4 see
`https://develop.sub.uni-goettingen.de/repos/textgrid/trunk/lab/info.textgrid.lab.search`

6.3. SOAP/WSDL

See
`http://textgridlab.org/axis2/services/Metadata?wsdl`

6.4. Accessing TG-Search, Eclipse-based

The class info.textgrid.lab.search.TextGridQuery represents a Query objects to be handed over to the search service. It basically has the methods for setting the parameters mentioned above. The results of the search will be held in the instance of class info.textgrid.lab.search.ResultHolder. This class will be given the raw answer of the Service and will parse it to extract the information contained therein (method fetchDeferredChildren). This will contain SingleSearchResult objects, which themselves can contain TextGridObjects together with the information how many search hits there were, and optionally the context (KWIC) where the search term was found.

Here are the methods that will set the parameters for a TextGridQuery:

```
setQueryText(String text)
setQueryMetadata(Map<String, String> queryFields)
setQueryBaselineElements(String[] bes) // term must occur in these BEs
setKwic(boolean kwic)         // return context of search term
setFullMetadata(boolean fmd) // return just URI or full MD?
setWantOriginal(boolean wantOriginal) // search in original encoding?
setPortion(String maxHits, String start, String session)
```

Possible keys for the Metadata Map are agent, type, title, project.

The following code shows how a simple list of all TextGridObjects of type „workflow“ can be retrieved. These objects will be displayed in a jface TableViewer (corresponding LabelProvider omitted).

```
import info.textgrid.lab.search.ResultHolder;
import info.textgrid.lab.search.TextGridQuery;
import info.textgrid.lab.ui.core.UpdatingDeferredListContentProvider;
[...]
         viewer = new TableViewer(parent, SWT.NONE);
         UpdatingDeferredListContentProvider contentProvider
               = new UpdatingDeferredListContentProvider(
                           TextGridObject.class);
         viewer.setContentProvider(contentProvider);
[...]
         ResultHolder resultHolder = new ResultHolder();
         TextGridQuery query = new TextGridQuery(resultHolder);
         HashMap<String, String> queryFields = new HashMap<String,
String>();
         queryFields.put("type", "workflow");
         query.setQueryMetadata(queryFields);
         viewer.setInput(resultHolder);
```

7. TG-crud

The TextGrid crud service is a web service to create, read, update and delete TextGrid resources, which can be TEI encoded documents as well as image files, e.g. TIFF or JPEG images. It is the interface to storing information to the replicated grid environment and the TextGrid databases – the central XML database (eXist) as well as the RDF-based relation database (Sesame), and furthermore the role based access control system (RBAC) using the TG-auth service. The crud service also checks access permissions and ensures that the TextGrid repository stay consistent.

It also uses the Adaptor Manager to convert TEI documents into the TextGrid baseline encoding using XSLT scripts, which also are stored in the XML database for efficient structural search. The Adaptor Manager also is capable of extracting relation information from the TEI files and the generated baseline eocnded files (as contained links to other TextGrid resources and XML schema references) and put them into the RDF database.

The service's input and output parameters to use the crud service and interface with the TextGrid repository are described in the service's WSDL file (`https://textgridlab.org/axis2/services/TGCrudService?wsdl`). The service can be accessed via SOAP, either by simply using the provided Java service stubs[15] (created by Axis2) or by setting up other SOAP-based web service clients if Java is not available or another web service framework shall be used. Furthermore a REST-based interface is provided (which also comes with Axis2), but was not heavily tested.

7.1. service methods and parameters

The TG-crud web service provides the following methods to access the TextGrid repository. Its input and output parameters will be described below except the two *sessionId* and *logParameter*, which already were mentioned in TG-auth* [16] and Logging[17] and must be used here. The schema of the TextGrid metadata is available at `http://www.textgrid.info/schemas/textgrid-metadata_2008-07-24.xsd`.[18] This schema is used in the crud service's WSDL file to create the TextGrid metadata object via Axis Data Binding.

getVersion

Input parameters: *sessionId (String), logParameter (String)*

Output parameters: *tgCrudVersion (String)*

Faults: *none*

[15] The current Java TG-crud client stubs are provided here for quick TG-crud access: https://develop.sub.uni-goettingen.de/repos/textgrid/trunk/middleware/tgcrud/

[16] cf. chapter 5. TG-auth*

[17] cf. chapter 4.4. logging, Web Service-based

[18] For a more detailed description of the TextGrid metadata please have a look at the TextGrid metadata and baseline encoding paper at www.textgrid.de

Description: getVersion returns the version name, number and date of the currently used crud service as a string, e.g. „v1.0 beta rc3 "energize" 2008-11-14“. Please use this version string to identify the TG-crud version in case of error or bug reports.

create

Input parameters: *sessionId (String), logParameter (String), tgObjectMetadata (textgridMetadataType), data (base64Binary)*

Output parameters: *tgObjectMetadata (textgridMetadataType)*

Faults: *authFault, ioFault, metadataParseFault*

Description: From the given TextGrid object – the metadata and data file – a new TextGrid resource will be created: (1) The data and metadata file will be stored to the grid, (2) relation information will be extracted from the metadata file as well as, if possible, from the data file, and stored to the RDF database, (3) the metadata file will be stored to the XML database and also the data file, if possible (using the Adaptor Manager) and finally (4) the resource will be registered to the RBAC using the TG-auth* service. In the returned TextGrid Object Metadata some middleware metadata will be provided, such as the URI of the object, its size, the creation date, and some more. The URI is created from the given metadata and some project and date information[19]. This URI is a unique TextGrid identifier and identifies the created resource unambiguously in the TextGrid repository and is hence used for retrieving or updating the resource again later.

readMetadata

Input parameters:*sessionId (String), logParameter (String), uri (anyURI)*

Output parameters: *tgObjectMetadata (textgridMetadataType)*

Faults: *authFault, ioFault, metadataParseFault, objectNotFoundFault*

Description: The metadata file of the TextGrid object with the given URI will be loaded from the grid. Some metadata will be fetched from the RBAC via TG-auth* and be filled in the metadata file, e.g. the object's owner and its permissions.

read

Input parameters: *sessionId (String), logParameter (String), uri (anyURI)*

Output parameters: *tgObjectMetadata (textgridMetadataType), data (base64Binary)*

Faults: *authFault, ioFault, metadataParseFault, objectNotFoundFault*

Description: This method retrieves the metadata file as in readMetadata, and additional the data file will be delivered, too.

[19] For a more detailed description of the TextGrid metadata please have a look at the TextGrid metadata and baseline encoding paper at www.textgrid.de

updateMetadata

Input parameters: *sessionId (String), logParameter (String), uri (anyURI), tgObjectMetadata (textgridMetadataType)*

Output parameters: *tgObjectMetadata (textgridMetadataType)*

Faults:*authFault, ioFault, updateConflictFault, metadataParseFault, objectNotFoundFault*

Description: Updating only the metadata of a TextGrid object is possible with this method by just giving the new metadata file and the URI. The URI in the given metadta file (`tgObjectMetadata.administrative.middleware.uri`) will not be used, but the lastModifiedDate (`tgObjectMetadata.administrative.middleware.lastModifiedDate`) is nesecarry to check the metadata's actuality. A change of adaptors (`tgObjectMetadata.relational.hasAdaptor`) is not permitted yet.

update

Input parameters: *sessionId (String), logParameter (String)*

Output parameters: *tgObjectMetadata (textgridMetadataType)*

Faults:*authFault, ioFault, updateConflictFault, metadataParseFault, objectNotFoundFault*

Description: Within this method the data file is updated, too. For the metadata all the rules mentioned above do apply. An update of adaptors in the metadata file is permitted in this method.

delete

Input parameters: *sessionId (String), logParameter (String), uri (anyURI)*

Output parameters: *deleted (boolean)*

Faults: *authFault, ioFault, objectNotFoundFault, relationsExistFault*

Description: This URI's document will be deleted from the TextGrid repository. A deletion is only possible yet, if you (1) have the role of a administrator, (2) the object is not published yet, and (3) the object is not involved in any chain of relations within the RDF database.

7.2. fault messages

In the following table all TG-crud fault mesages will be described. Every fault message has two parameters, which describe the error and its possible cause. The parameter *faultMessage* is about what went wrong in crud, what the service could not do, e.g. „Failure storing data to XML database“, „Unable to read metadata file from URI 'textgrid:...'“ or „Metadata update denied“. The parameter *cause* explains what caused the error in crud, which can be messages from exceptions that occured in underlying services, e.g. in TG-auth* while checking access permissions, in JavaGAT while accessing grid resources or in the several used database clients (XML and RDF databases).

authFault	An authFault occurs, if a certain action is not permitted using the given session ID, e.g. a project member tries to *delete a* TextGrid resource, which is not allowed at the moment or someone tries to *update* an already published resource, that only permits readonly access.
ioFault	A more general fault, that always occurs, if there are any problems with grid or database access.
objectNotFoundFault	A requested resource is not found, e.g. the URI is not existing.
metadataParseFault	The TextGrid metadata could not be parsed correctly. This fault can occur if invalid metadata files are sent to the TG-crud or some XSLT adaptor scripts fail to parse the given TEI data file.
relationsExistFault	If a TextGrid object shall be deleted from the repository and there are any relations in the RDF database, the deletion is not permitted yet.
updateConflictFault	An updateConflictFault occurs, if the lastModifiedDates or the hasAdaptor relation content given in the metadata differs from the values stored to the grid. That means another person probably already updated the resource. The actual data and metadata must then be *read* again and be merged, e.g. using the TextGrid collation tool. After that an update can be done.

7.3. usage of the Java TG-crud client stub

getVersion request

At first, of course, you need to import the client stub JAR file as following:

```
import
info.textgrid.namespaces.middleware.tgcrud.services.tgcrudservice.
TGCrudServiceStub;
```

You also must import some of the Axis2 classes, e.g.

```
import org.apache.axis2.AxisFault;
import org.apache.axis2.databinding.types.URI;
```

The next step is to create a instance of the stub – the reference **`server_url`** `points to` the endpoint you already got from the config service (cf. chapter 4.1. configuration, Java client library):

```
TGCrudServiceStub stub = new TGCrudServiceStub(server_url);
```

As first service call we ask the service for its version. Therefore we do create instances of the following objects: `GetVersion`, `GetVersionType` and `GetVersionResponseType`.

```
GetVersion getVersionRequest = new GetVersion();

GetVersionType getVersionType = new GetVersionType();
```

Then fill the GetVersionType with the authentication and logging parameters (**`rbacSessionId`** and **`logParam`**) provided by the config service and assign the GetVersionType to the getVersionRequest:

```
getVersionType.setSessionId(rbacSessionId);

getVersionType.setLogParameter(logParam);

getVersionRequest.setGetVersion(getVersionType);
```

Now we can create and fill our `getVersionResponse` Parameter, that will provide us with the services response parameters, the serbice call happens in the `stub.getVersion()` method:

```
GetVersionResponse getVersionResponse =
stub.getVersion(getVersionRequest);
```

Finally get the version string returned from the TG-crud service and do whatever you like with it:

```
    String tgCrudVersion =
getVersionResponse.getGetVersionResponse().getTgCrudVersion()
```

read request

To receive a complete TextGrid object – a metadata and a data file – from the TG-crud service, do as in the getVersion request and use the appropriate parameters, take the **`uri`** of the object you mean to retrieve (cf. e.g. the chapter on TG-search):

```
Read readRequest = new Read();
ReadType readType = new ReadType();

readType.setSessionId(rbacSessionId);
readType.setLogParameter(logParam);
readType.setUri(uri);
readRequest.setRead(readType);
```

```
ReadResponse readResponse = stub.read(readRequest);
```

And finally retrieve the service's response:

```
MiddlewareMetadataType metadata =
readResponse.getReadResponse().getTgObjectMetadata();

DataHandler data = readResponse.getReadResponse().getData();
```

7.4. handling large files - MTOM

MTOM (SOAP Message Transmission Optimization Mechanism) allows for the transfer of binary attachments of arbitrary size via SOAP, as well as increased performance through caching. More information about MTOM can be found on the Axis2 homepage[20], or the specification by the W3C[21]

For (a) enabling MTOM in the client, use

```
<code>
stub._getServiceClient().getOptions().setProperty(Constants.Configuratio
n.ENABLE_MTOM,
Constants.VALUE_TRUE);
</code>
```

(b) enabling caching at client side

```
<code>
stub._getServiceClient().getOptions().setProperty(Constants.Configuratio
n.CACHE_ATTACHMENTS,
Constants.VALUE_TRUE);
stub._getServiceClient().getOptions().setProperty(Constants.Configuratio
n.ATTACHMENT_TEMP_DIR,
"/textgrid/tmp/");
stub._getServiceClient().getOptions().setProperty(Constants.Configuratio
n.FILE_SIZE_THRESHOLD,
"4096");
</code>
```

[20] http://ws.apache.org/axis2/1_4_1/mtom-guide.html
[21] http://www.w3.org/TR/soap12-mtom/

8. TextGridLab's Object Model

TG-crud offers a rather different interface for saving information than Eclipse: Eclipse's storage system is based on a workspace containing named projects containing named files and folders, rather oriented on traditional file systems. TG-crud, however, maintains a set of objects described by a set of metadata (like authors, title etc.). Both can use URIs to identify files.

The plugins info.textgrid.lab.core.model and info.textgrid.lab.core.efs.TG-crud try to bridge the gap between both systems and to offer a convenient way for TextGridLab plugins to access both TextGridRep's objects and metadata.

8.1. three implementation layers

We use three implementation layers to manage our objects:

◦ The **model layer** represents TextGridRep's objects using their metadata, in TextGridObject's. This layer is completely implemented by TextGrid, and it is the layer clients like object browsers or metadata editors usually talk with and which commands and menu entries on TextGrid objects are usually contributed to.

◦ The **resource layer** containing `IFile`s, `IProject`s and other `IResource`s is the layer (content) editors usually see. It is the standard way of interacting with files in Eclipse, and it is completely implemented by Eclipse.

◦ The **EFS layer** contains the backend that performs the actual reading and writing. It is implemented by TextGrid extending an interface defined by Eclipse, and clients usually don't interact directly with this interface

Conversion between the layers works by using the *adaptor pattern* as provided by Eclipse. I.e., if you have a TextGridObject and need an IFile,

```
TextGridObject textGridObject;
/* ... fill textGridObject ... */
IFile file = (IFile) textGridObject.getAdapter(IFile.class);
if (file != null)
  /* do something with file */
```

or vice versa

```
IFile file;
/* ... fill file ... */
TextGridObject textGridObject = (TextGridObject)
file.getAdaptor(TextGridObject.class);
```

To read or write data from or to a TextGridObject, you can then use IFile's methods *setContents()* and *getContents()*, which take or give you input streams for the object's contents.

8.2. Getting a TextGridObject – From a TextGrid URI

```
URI uri;
/* get the object's URI somehow into uri */
TextGridObject object = TextGridObject.getInstance(uri, true);
```

This creates a new TextGridObject from a TextGrid URI, calling TG-crud#readMetadata if neccessary. If you set the last argument to `false`, the call to readMetadata may be delayed.

8.3. Getting a TextGridObject – From a Metadata blob

You need to pass in the metadata as an XML tree fragment hanging from an OMElement (that's AXIOMs element representation). The OMElement should contain the tgMetadataObject.

```
OMElement element;
boolean complete;    // if false, element contains only fragmentary
metadata
/* get some metadata into element */
TextGridObject object = TextGridObject.getInstance(element, complete);
```

If you create a TextGridObject from an XML fragment, there will be no immediate network access. This is especially useful if you create a lot of TextGridObjects at once (think search) and do not want a web service call for every single one. With the last argument (*complete*), you determine whether the metadata fragment you pass in is complete, i.e. it contains all the metadata TextGrid knows about for this TextGridRep object. If you specify false here and access a field not present in the metadata fragment at a later point in time, the object is finally completed by a call to TG-crud's readMetadata operation.

8.4. Representing TextGridObjects

If your plugin somehow represents / contains a bunch of TextGrid objects (e.g. search results or object browser), your objects (the stuff you put in your StructuredSelections and/or that your ContentProvider provides) must adapt[22] to TextGridObjects (and they should adapt to IFile, as well; once they have a TextGridObject *tgo* they do so by forwarding the adaption request to TextGridObject, i.e. return *tgo.getAdaptor(IFile.class))*.

Either your objects are PlatformObjects or IAdaptables and you provide an adaptor factory, or you simply put the TextGridObjects themselves in your selection/viewer.

8.5 Contributing new kinds of documents

When contributing a new tool, you may need to specify a new content type, as well. In our metadata record, the content type's identifier belongs in the field *middleware/client/format*. In TextGridLab, you can see content type support e. g. in the new and import dialog, but it is

[22] See, e.g., Wayne Beaton (2008): Adapters. http://www.eclipse.org/articles/article.php?file=Article-Adapters/index.html for an introduction to the adapter pattern in Eclipse. Also have a look at the class info.textgrid.lab.core.swtutils.AdapterUtils and its JavaDocs.

also used to determine the default editor for an object you open from the navigator or the search results. From a developer's point of view, there are methods in TextGridObject to retrieve an object's content type, and there is the class *TGContentType* with static methods to list the available content types.

To contribute a new content type to the TextGridLab, go to the Extensions tab in your plug-in manifest and add an extension to the *info.textgrid.lab.core.model.contentTypes* extension point. As an example, this is the extension that contributes XML support, in its XML form:

```
      <extension
            point="info.textgrid.lab.core.model.contentTypes">
         <contentType
               additionalContentTypePattern="xml/.*"
               eclipseContentType="org.eclipse.core.runtime.xml"
               extension="xml"
               image="icons/XML.gif"
               internal="false"
               name="XML Document"
               typeID="text/xml">
         </contentType>
      </extension>
```

The most important part is the *type ID*, which contributes the content that goes to the *administrative/client/format* field in the TextGrid metadata record. We also connect an Eclipse content type (providing the association with existing editors), a file extension for local links and copies to the file, a human-readable (and localizable) description and an icon for objects of this type. The detailed documentation for the extension point can be found in its extension point description, which is available inside Eclipse.

Extending the New Object wizard

Lab users create new objects using the New Object wizard. On the first page, users select the content type of the object. By default, the wizard will optionally allow the user to enter descriptive metadata on the second page and, when the user clicks *Finish*, create a new, empty, unsaved object with the given metadata and open the default editor for the new object.

You can customize this behaviour by contributing to the extension point *info.textgrid.lab.ui.core.newObjectContributor*. Two customization options are available:

- you can implement metadata preparation and override the action for the wizard's *Finish* button by associating an *INewObjectPreparator* implementation using the *preparator* attribute, and
- you can add custom wizard pages using *wizardPage* subelements.

For example, the XML editor contributes additional wizard pages to select a schema and a root element for new XML files. It generates an XML skeleton and sets it as the initial contents for the editor (using TextGridObject's *setInitialContents()* method) and it opens the XML editor and switches to the XML perspective when you press the *Finish* button.

For more and up-to-date details on both the TextGridLab's object model and these extension mechanisms, please see the respective class and package javadocs.