1 Accessing TextGrid Data Structures from your Lab plugin

As mentioned before, you will typically just implement Eclipse tooling – see the corresponding documentation and tutorials for that. This section hints at some important concepts to access TextGrid specific data structures in the Lab.

1.0.1 TextGridObject

Instead of files and folders, TextGrid uses *objects* that bundle some content with a set of metadata. Each object has a TextGrid URI (like `textgrid:47k11.0`) that uniquely identifies the object.

In the TextGridLab, there is a data structure called TextGridObject which represents one of those objects. In one Lab instance, there will be at most one `TextGridObject` per object and there are factory methods that produce TextGridObjects from URIs, out of the blue, or from metadata blobs.

1.0.0.1 Accessing actual metadata

Each TextGridObject has a `.getMetadata()` getter that retrieves a copy of the metadata and a corresponding setter that updates the object's metadata. The `makeMetadataPersistent()` method can be used to push the metadata to the repository explicitly, but metadata is also updated whenever you save the object's contents using the standard methods described below.

There are also some helper methods to deal with specific frequently-used metadata fields.

1.0.0.2 Accessing an object's content

We offer an object's content in the same way as Eclipse offers access to files in its workspace: Using Eclipse's `IFile` interface.

Adapter Pattern

Eclipse makes heavy use of the *adapter pattern* to convert between objects. This allows for less tight coupling between classes than inheritance or common interfaces. It works by using a global *adapter manager* which maintains a list of *adapter factories* that can convert between objects and convenience methods (`adaptableObject.getAdapter(targetClass)`) to access this.

Thus, conversion between the layers works by using the adapter 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);
```

The IFiles have methods for reading and writing and they can also be passed, e.g., to plain Eclipse editors (by means of a `IFileEditorInput`, which is a standard Eclipse interface that can also be adapted to (and from) directly from TextGridObjects).

1.0.0.3 Accessing other TextGrid services

The `info.textgrid.lab.core.model` plugin contains APIs for accessing the basic object and project structure, see the API documentation for details. If you need to interact directly with TextGrid services, you will find the respective client libraries in the Lab as well. For TG-search, there is a wrapper and a bunch of utility classes. You might also look into the UI utilities for, e.g., GUI elements that represent search results.

1.0.0.4 Configuring TextGrid services, and AAI

If you do not find ready-made client factories in the corresponding wrapper plugins and need to work with the core service APIs, you'll need to provide an *endpoint* (the address where the service is deployed) and a *session id*.

The **endpoint** is available via the Configuration Service.

To get the *session id*, call RBACSession.getInstance().getSID(boolean). If you pass `true`, this might trigger a login dialog. If the function returns an empty string, the user has not logged in – passing on the empty string to functions that require a session id still allows to access public ressources.