1 Contributing to the UI

Contributing to the graphical user interface basically works analogous to standard Eclipse — see the Eclipse Platform Developers' Guide for more information on that. This section lists some of the TextGridLab specifics.

1.0.1 Contributing to Menus and Toolbars

The TextGridLab features a *Tools* menu and a corresponding toolbar in which your tool should appear. To do so, you should contribute a command that opens your tool in some way – typically this means switching to a dedicated perspective and maybe opening some parts. Contribute your command to the menu `tools` and to the perspectives bar. `plugin.xml` excerpts, as examples from the *CollateX* plugin (you'll want to use Eclipse's graphical editor for the plugin.xml to create these entries!):

```
<menuContribution
 allPopups="false"
 locationURI="toolbar:perspectivesbar">
 <command
 commandId="info.textgrid.lab.collatex.openPerspective"
 style="push">
 </command>
</menuContribution>
<menuContribution
 allPopups="false"
 locationURI="toolbar:perspectivesbar">
 <command
 commandId="info.textgrid.lab.collatex.openPerspective"
 style="push">
 </command>
</menuContribution>
```

Another typical use case is to add a command to the context menu of TextGrid objects, e.g., in the navigator or in the search results view. To do so, you can use Eclipse's expression syntax for the `plugin.xml` and check for adaptability to `TextGridObjects`. Here is an excerpt from the import tool's `plugin.xml` that adds entries to context menus whenever one or more TextGrid objects are selected:

```
<extension
 point="org.eclipse.ui.menus">
  <menuContribution
 allPopups="false"
 locationURI="menu:file?after=file.anchor6">
     <command
 commandId="info.textgrid.lab.core.importexport.import"
 mnemonic="%command.mnemonic.import"
 style="push">
     </command>
     <command
 commandId="info.textgrid.lab.core.importexport.export"
 mnemonic="%command.mnemonic.export"
 style="push">
     </command>
  </menuContribution>
  <menuContribution
 locationURI="popup:org.eclipse.ui.popup.any?after=popup.anchor2">
     <command
 commandId="info.textgrid.lab.core.importexport.export"
 mnemonic="%command.mnemonic.export.1"
 style="push">
    <visibleWhen>
       <iterate
 ifEmpty="false"
 operator="or">
          <or>
         <adapt
 type="info.textgrid.lab.core.model.TextGridObject">
         </adapt>
         <adapt
 type="info.textgrid.lab.core.aggregations.ui.model.Aggregation">
         </adapt>
          </or>
       </iterate>
    </visibleWhen>
     </command>
  </menuContribution>
</extension>
```

In the associated handler, you can then get the selection from the event and adapt the contents of the `IStructuredSelection` to `TextGridObject` in order to deal with them. An examplary handler that deals with quite some cases is the one from the Export tool. This handler already contains quite some complex handling: It first checks whether the selected object is an Import/Export specification (and if so, just opens it in the Export editor). Otherwise, it checks whether we already have an export editor open: If so, the selected objects are added to that, otherwise, a new editor is created and the selected objects are added.

1.0.1 Dealing with your own kind of object

If your use case includes to offer an own, project specific kind of object and the appropriate tools to edit these objects, then you'll probably want to define your own *TextGrid content type* (similar to a MIME type), a corresponding Eclipse content type and an editor that deals with that.

The **TextGrid content type** is responsible for registering your format and providing the TextGridLab-specific properties. Have a look at the CollateX content type for collation sets as an example. If you use Eclipse's plugin manifest editor to add this, you'll get additional help in its tooltips.

We also added a matching Eclipse content type for this file type. This is required for the general eclipse features like defining a matching editor for your content type.

If you have added these three definitions to your `plugin.xml`, your content type will also appear in the *New Object* dialog. The default action when the user selects your content type in the *New Object* dialog will be to create a new object of your content type and open it in your editor. Your editor will also be available in the corresponding context menu of objects of your content type and it will be opened by default on double-clicking existing objects of your content type.

If possible, you should define your editor as a standard `EditorPart` that deals with `IFileEditorInputs`. If you do so, you will also get appropriate locking behaviour without having to do anything for that.

1.0.1 Locking

While your user works on an object, that object should be *locked*. As long as a user holds a lock on the object, other users' attempts to lock the object fail with a message indicating the user who has the lock, and other users will not be able to modify the object.

The TextGridLab will **automatically lock and unlock** objects for a user as long as (1) the object is edited in an editor derived from *EditorPart* and registered in the usual Eclipse mechanism, *and* (2) the Editor's input is an `IFileEditorInput` that corresponds to (and thus adapts to) `TextGridObject`.

Otherwise, use the LockingService to **manually lock and unlock** the object. Please note that, for reasons of liveliness, the server side lock by TG-crud will only last for 30 minutes, the object needs to be relocked during that period. The TextGridLab's locking service will do that for you automatically.

Please note that locking is not strictly required, it is mainly a *fail fast* mechanism. If a user tries to update an object that someone else modified in the meantime, TG-crud will throw an UpdateFault and the user might be in a situation where he either cannot save his modifications or reloads the metadata and overwrites the other person's changes. Locking will cause the process to fail *before* the user makes any modifications.