Developer Guide: How to write plugins for the *HELM* *Antibody* *Editor* V2.0

Marco Lanig & Sabrina Hecht
quattro research GmbH, Martinsried
25.03.2015

Table of Contents

1 General

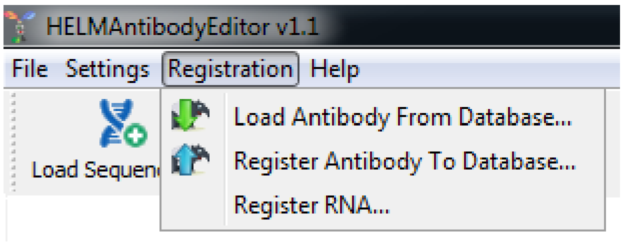
The **HELM Antibody Editor** v2.0 provides a plugin mechanism that can be used to dynamically load custom functionality without touching the main code.

2 Plugin Types

Currently there are four different plugin types available.

2.1 Menu entry plugin

Adds a custom menu entry to an arbitrary menu.

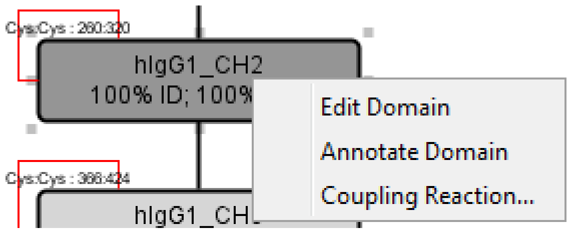


2.2 Toolbar button plugin

Adds a button with custom action to the toolbar below the application menu.

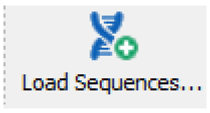
2.3 Domain node popup plugin

Adds routines to the domains context menu which may be used to modify the selected domain.



2.4 Sequence file reader plugin

Replaces the file reader which is used to load sequences into the Antibody Sequence Editor. This should be implemented if readers other than the built-in file readers for FASTA and VNT files are needed.



3 Writing your own plugins

This chapter contains brief instructions on about how to implement own plugins. Own plugins may be bundled into one single project or can use different ones.

3.1 Menu entry plugin

For this kind of plugin *AbstractEditorAction needs to be implemented*. The interface can be found in the editors package *org.roche.antibody.ui.actions.menu*.

Icon and description can be set in the constructor as follows:

```
public MyAction(JFrame parentFrame) {
  super(parentFrame, NAME);

  ImageIcon icon = getImageIcon(IMAGE_PATH);
  if (icon != null) {
    this.putValue(Action.SMALL_ICON, icon);
  }
  this.putValue(Action.SHORT_DESCRIPTION, SHORT_DESCRIPTION);

  setMenuName(MENU_NAME);
}
```

The plugin loader only creates a new menu if it does not exist yet. This allows to extend the default menus (File, Settings, Help) without having to define a new menu.

If a user clicks the new menu item, the implemented *actionPerformed(ActionEvent e)* method is called. Any custom routines like dialogs can be executed from within this method.

3.2 Toolbar button plugin

For a toolbar plugin *ToolBarToggleButton* needs to be implemented. The interface can be found in the editors package *org.roche.antibody.ui.toolbar.buttons.*

Because every toolbar entry should also be accessible via the application menu, the steps mentioned in 3.1 to create a menu entry should be implemented first. The toolbar button plugin should then use the same action.

```
public MyToolbarButton(JFrame parentFrame) {
  super(new MyAction(parentFrame));

  super.setIcon(getEditorAction().getImageIcon(IMAGE_PATH));
  setToolTipText(MyAction.SHORT_DESCRIPTION);
  setText(MyAction.NAME_SHORT_NAME);
}
```

3.3 Domain node popup plugin

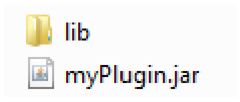
To extend the domain popup an interface needs to be implemented which is castable as *JMenuItem*. This may for example be *JMenuItem* itself, or *JMenu* if several submenus shall be added. The constructor takes two parameters: *JFrame* and *org.roche.antibody.model.antibody.Domain*. A custom *AbstractAction* has to be added to the menu item which takes the Domain as input.

3.4 Sequence file reader plugin

A custom file reader plugin needs to implement the interface *ISequenceFileReader* in the package *com.quattroresearch.antibody*. The only function (*read(..)*) takes the parent frame (for showing message dialogs) and the input file as parameters. It shall return an array of two lists. The first list contains the chain names and the second one the corresponding sequences.

4 Deployment

As already mentioned all plugins may be bundled in a single project. The classes need to be packaged into a single jar file and should be put into the plugins folder. All of the classes inside will be loaded into the classpath. However, if the plugins need external libraries, those should be copied to a subfolder outside the jar file.



lib
myPlugin.jar

5 Configuration

The HELM Antibody Editor won’t automatically load every plugin found inside the plugins folder thereby allowing to deactivate some without repackaging. The “config\ab-application-configuration.properties” contains one line for every plugin type. Add the fully qualified names of the plugin classes that should be loaded in the editor automatically here. Plugins need to be separated by semicolon as shown in the following example.

```
#Plugins
plugins.menu=my.plugin.menu.MenuPlugin1;my.plugin.menu.MenuPlugin2
plugins.toolbar=my.plugin.toolbar.ToolbarPlugin
plugins.popup.domain=my.plugin.domain.DomainPlugin1;my.plugin.domain.DomainPlugin2
plugins.filereader.sequence=my.plugin.file.MyFileReader
```

6 Contact

Please contact lanig@quattro-research.com, hecht@quattro-research.com or schirm@quattro-research.com in case of questions and feedback.