A Guide to writing an sMILX plugin

sMILX has four types of plugins:

- A dock window plugin such as a log window or scripting console (see the python plugin)
- An extension that implements one or a few context menu items for a type of data (see the denoise plugin)
- A plugin that implements a display for another type of data including loading and saving of that data such as shape models (see the ssm plugin)
- A full plugin that does all of the above.

Extensions

An extension is a light-weight plugin that provides just a few context menu entries by sub-classing the basic milxQt classes like milxQtImage or milxQtModel. An example of an extension is the deNoise plugin. Here the milxQtModel class is subclassed so that it can display models as well, implements a couple of further members and adjusts the context menu to suit.

A standard window, such as a model window, will show that an exentsion is available (show as a context menu entry). Clicking it will cause a new window to open, which is effectively the sub-classed version of the standard window shown before, but the context menu will be different and allow the user to execute the extra members in the sub-class.

This type of plugin is a quick and easy way to have a good start to develop a plugin. Only requires sub-classing a class like milxQtImage or milxQtModel.

Dock Plugins

Dock plugins are useful for display information about the data or provide a different interface to the data such as a command line interface or shell. An Example of a dock plugin is the python plugin, which provides a way to run all aspects of sMILX via a python script.

Creation

To create your own plugin:

1. Download the latest “plugin_guide.tar.bz2” file from the home page or from within the sMILX.Help.
2. Choose a name for your plugin, you will be replacing _RENAME_ with that name. For arguments sake, lets call it DiffusionTensor.
3. Extract “plugin_guide.tar.bz2” and copy the milxQt_RENAME_Plugin.h and milxQt_RENAME_Plugin.cpp files to a directory for your plugin in the “plugin” directory of SMILI. In this example we will call the directory "dti".
4. Use find and replace to replace the string “_RENAME_” with the name of your plugin in both files and the CmakeLists.txt file.
5. Use find and replace to replace the string “_MYPLUGINCLASS_” with the name of your helper plugin class in both files (see SSMPlugin as an example of such a class). In this example, we call it "DTIPlugin". The helper class implements most of the plugin features for a plugin window. If you don't have a class, ensure the relevant members (such as isPluginWindow()) of the plugin class return false and remove references to milxQt_MYPLUGINCLASS_.
6. Repeat this with the CmakeLists.txt file, which will build the plugin.
7. You are now ready to implement your plugin. Implement the appropriate members of the plugin class, leaving the others blank or returning NULL. For implementing a threaded plugin, put your code into the “run()” member of the class.
8. Subclass one of the milxQt* classes, like milxQtModel or milxQtRenderWindow etc. And add the relelvant areas where instead NULL was returned. For example, for the case of milxQtModel subclassed plugin, you will need to update modelResult(), isPluginWindow(), pluginWindow(), loadExtension() and uncomment relevant parts of the constructor. See the attached DTI example upto this point as an example.

Troubleshooting

Help! My plugin doesn't load!?!
Check that you have linked all the relevant libraries correctly. DLLs or shared objects with unresolved externals will be ignored by the sMILX plugin loader. If this is correct, check the naming of the classes in your plugin class. Especially at the line “Q_EXPORT_PLUGIN2(_RENAME_Plugin, milxQt_RENAME_PluginFactory);”. Finally, check to see if you have not redefined/reimplemented an implemented function in milxQtPluginInterface.
I get the error “cannot allocate an object of abstract type ‘Classname’” error!
The class milxQtPluginInterface is a virtual class, this means that when milxQtMain calls its members, the members of derived classes are called instead. This requires that your plugin (derived) class MUST have at least the same members (that are = 0) as milxQtPluginInterface. It can have more, but as a minimal, it must have the same members, i.e. it must be a superset of members. Ensure that any member of milxQtPluginInterface that is = 0 is present in your class at least empty. For example, virtual void loadExtension() = 0; is
virtual void loadExtension() {}
Already implemented members of milxQtPluginInterface can be ignored.