QDL's GUI

Introduction

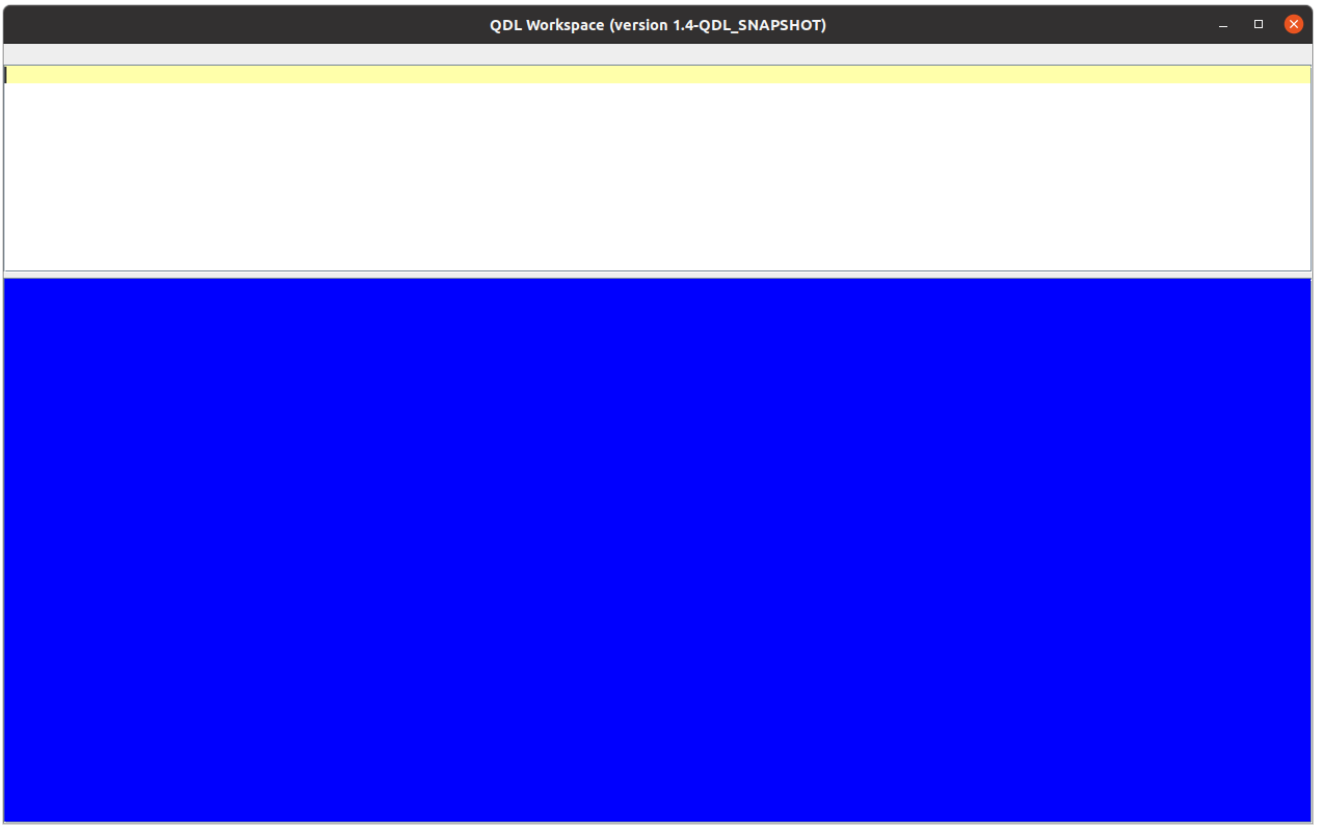
QDL comes with a built in Swing GUI (Graphical User Interface). This is intended to keep with the philosophy that QDL "just works" out of the box everywhere. As long as your environment supports graphics, it should function. QDL is used heavily for server-side work and these generally have no graphical user environment. As such, the character mode workspace will continue to be the de facto work environment.

Getting started

To invoke the GUI, you simply add the **-gui** flag to the command line

```
java -jar qdl.jar -cfg path/to/config.xml -name config_name -gui
```

And you will be greeted with this



This has an input area (white) and an output area (blue). The ouput area is not editable, but you can certainly copy things from it.

The Bottom Line

The usual *modus operandi* of QDL remain the same: type in QDL expressions or workspace commands, get results. This entire interface just allows for more convenience input. The major innovation is that the line editor is replaced with a much snazzier dedicated QDL editor. Everything in the workspace is still there and everything works the same from the command line. You still make buffers, execute them, etc. Since the input area is now a full fledged QDL-aware , multi-line editor, you will need a lot fewer buffers in practice.

The Input Area

This supports (unlike the command line version) multiple lines and statements. Simply type as per normal. *To execute, enter **ctrl+enter***. The result will display in the blue area.

The input area supports

- QDL's keyboard (view mappings with **`)help keyboard`**)
- syntax highlighting
- auto complete (type ctrl+space)
- auto-indent and parenthesis/bracket matching.

It should be noted that the syntax highlighting could be better. This is because ideally every keystroke would run the entire input through the parser and pick it apart, but that makes performance miserable after a point (type a key, go get a cup of coffee…) Therefore, regular expressions are used to more or less identify syntax. This allows for real time typing and a responsive interface.

(QDL is backed by a grammar and uses ANTLR, which is an industrial strength parser generator precisely because it is in general impossible for regular expressions to parse such a language.)

Operations

There are a few operations generally that are supported

- ctrl+↑ (up arrow) – move backwards through the issued commands. The commands and corresponding output are displayed.
- ctrl+↓ (down arrow) – move forwards through the issued commands.
- ctrl + q = quit
- ctrl + s = issue the **`)save`** command on your behalf.

The Output Area

The output area is, indeed, simply a read-only area that captures the output.

The Editor

Buffers work as per usual, and instead of the line editor, there is a graphical editor. This is really just an input area with no execution. It has exactly three operations

- F1 – help displays a message about the following two operations.
- ctrl+s – save the current buffer. If a file, it will be written to the disk, if anything else, it is updated in the workspace.
- ctrl+q – quit the editor. You will get a warning if you want to quit.

Most importantly, the editor is multi-threaded, so if you have an editor window open, you can save it (ctrl+s) and the result is immediately available in the workspace to run or work with. You can just leave the window open if you want and test out snippets of code then update the window, for instance. Do remember that operations on the editor will update the QDL workspace, but you may need to use `)b save handle` to save files. (If you are saving your workspace and forget, QDL will save the pending changes there so you can just save them later, though you should make sure you proactively keep your files updated as you want them to be.)

Hey, you should make a full featured GUI with menus etc…

Yes I should. At this writing (Aug. 2022) it is clear that Swing in Java is going the way of the Dodo. Largely this is due to vastly improved graphics (so Swing applications looks awful without a lot of platform and hardware specific tweaking, which defeats the purpose) or people are just interested in phone apps. It will remain as a basic cross-platform solution for some time to come (though it will officially cease upgrades in 2026 and go into maintenance mode). The designated successor, JavaFX, we removed *in toto* from the most recent Java release because nobody used it and it was more of a scripting language, not a GUI development language in the final analysis. At some point, a full featured GUI for QDL is certainly planned, but it is unclear where or how that should be written. Prehaps in Ко́тлин...