QDL's GUI

Introduction

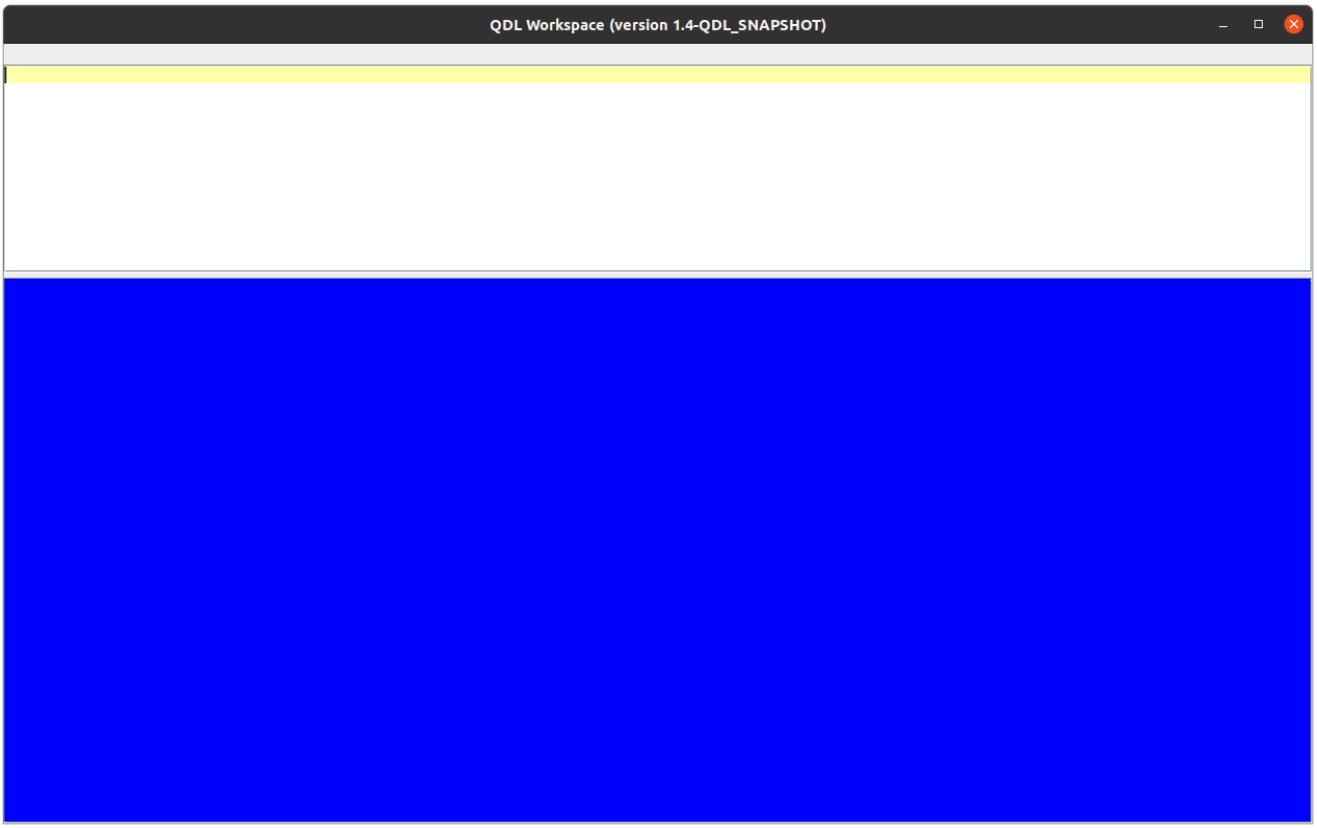
QDL comes with a built in Swing GUI (Graphical User Interface). This is intended to keep with the philosophy that QDL "just works" out of the box everywhere. As long as your environment supports graphics, it should function. QDL is used heavily for server-side work and these generally have no graphical 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 output area is not editable, but you can certainly copy things from it. In both areas, highlighting a work and hitting F1 will pop up help for that topic.

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.
- local clipboard history (ctrl+shift+v)

It should be noted that the syntax highlighting could be better. This is because ideally every keystroke would run the entire input through the lexer 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.)

Shorthand

Editor Operations

Key	Type	Description
^a	E	Select all text
^c	E	Copy current selection to clipboard *or* current line (no selection)
^d	E	delete current line
^j	E	join next line to current
^h	B	run check_syntax on the selected text or current input.
^i	E	paste clipboard as input form
^I	E	paste long input form of clipboard, multiline if linebreaks (useful!!)

^k	E	goes to next occurrence of occurrence of selected text
^@k	E	goes to previous occurrence of selected text
^m	E	toggle comment for line or selection.
^q	W	Quit QDL (w/ save prompt)
^Q	W	Quit QDL now (no prompt)
^r	E	replicate current line or selection
^s	E	save workspace
^v	E	Paste from clipboard
^@v	E	Show local clipboard history
^x	E	Cut current selection to clipboard
^y	E	redo last action
^z	E	undo last action
^backspace	E	delete previous word
^space	E	Complete word
^@enter	W	execute current code
^@page up	B	Go to previous entry in history
^@page down	B	Got to next entry in history
@+up	B	move line up in buffer
@+down	B	move line down in buffer
F1	E	Show help for selected keyword, or general editor help if no selection.
^F1	E	Show QDL keyboard layout

Key:

^a = ctrl + a
@a = alt + a
^@a = ctrl + alt + a

Types:
B = buffer operation
E = edit operation
W = workspace operation

Note about pasting input form.

The clipboard is only read as string. You may opt to try and have it interpreted into input form. This means that if the clipboard can parse as a number, null or boolean it will be pasted as is. If it is a string, it will be turned into a single line. So it the clipboard had

```
mairzy doats
      and dozey
           doates
and liddle
labsie      divey
```

^i would insert

```
'mairzy doats\n\tand dozey \n\t\tdoates\nand liddle\nlabsie\tdivey '
```

where ^I would try to preserve the formatting across lines as

```
'mairzy doats\n'+
'  and dozey \n'+
'    doates\n'+
'and liddle\n'+
'labsie  divey'
```

which is very useful when pasting long formatted text strings (like JSON). In this case, tabs are converted to characters.

Special characters

The QDL keyboard supports many special characters. These are typically accessed with @+key (@ = alt), so that @+p inserts the Greek letter π (pi). Help for the workspace (under keyboard if you want to see the layout, or unicode if you just want a list) is provided.

The Output Area

The output area is, indeed, simply a read-only area that captures the output. If you issue ctrl+mouse button 1, the current result will pop up in an edit window. This window is independent of the workspace, so it will not be updated when the workspace is.

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 – displays a help message

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, was removed *in toto* from the most recent Java release because few 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. Perhaps in Ко́тлин...