The ANSI Terminal

QDL has support for an ANSI terminal. This allows for a command history (accessible with up and down arrows) as well as unicode support.

Invoking QDL with the ANSI aka ISO 6429 terminal:

Add the flag **`-ansi`** to the command line, e.g.

```
bash$ java -jar qdl.jar -ansi
```

You may get the standard mappings for unicode in the the session by typing

```
)help unicode
```

Abbreviations used here.

Just to keep the typing down, we will write

^ = press the control key first

@ = press the ALT key first.

E.g.

^x = press the control key + (lower case) x

@x = press the ALT key + (lower case) x

@X = press the ALT key + SHIFT key + (upper case) x

Generally we do not use ^P (so upper case control letters here).

The keys are laid out so that either you add ALT to the key for the alternate character you want, e.g.

@= yields ≡

or some, such as assignment, are not so easy to map to a keyboard, so these are paired as best as possible, @: and @" are resp. left and right assignment

Special characters

There are several unicode characters available when using the ANSI terminal. `@=` should return `≡.` for instance. You should always check what is current in the QDL workspace with

```
)help unicode
```

Note that in external editors, (like nano or vim) special characters are not supported. The built-in line editor though does support them. The reason is that the key mappings are local to QDL. To get them

generally would require hacking, *ahem* editing the system keymap file, but that would change the behavior of every application E.g. @P might no longer cause some application to print, but would insert the Greek letter π. If you want to do that, have at it, but QDL should not just take over anyone's system.

A technical note

It is actually quite hard to make a command line interface in Java. The reason is that Java does not process input from the console in a reasonable way and requires quite a bit of fudging. On top of this, the way that ISO 6429 works is that certain combinations of characters are interpreted by the actual hardware (such as `^[A` for the up arrow. In theory, you could type this in and get the cursor to move). This overloads the input stream with control information.[1] This is why sometimes if your console (independent of QDL) gets whacky, you may see control characters streaming past you. Because the input stream is overloaded with control characters, (bad choice, but this standard goes back to the 1970's at least) about the only option is to introduce wait states for input. So as you type, everything is timed and if there is nary a split second between key strokes, the assumption is that there is a control character, not that you typed in a control sequence.

This has one very serious ramification: Superfast typing may drop characters and this is because of the wait states in Java stream processing, not QDL. In a similar vein, pasting text from a clipboard tends to get text lost or can even make the JVM crash (since the OS can type much faster than you). The solution is outlined below. The utility of the ANSI terminal is that it is actually platform independent. There are no special libraries (which can require very invasive things like a recompile of your unix kernel in some cases) that need to be used.

QDL's philosophy is to keep it simple and basic at all times so that it just works. If you run it in another environment (such as a Swing-based console that supports more operations), the idea is that none of accommodations typically made for text mode applications preclude running it in as a graphical application.

The Basics

The advantage of using the ANSI terminal is that it supports the following

- command history accessible with up and down arrows
- in place editing of lines using left right arrows, backspace, delete and typing
- home moves to the start of the current line, end to the end of the current line
- Pasting from the system clipboard (with some caveats) is supported

1 Who remembers using a Captain Crunch (popular breakfast cereal) whistle to make free phone calls? The old land line phone system also sent its control information over the same wire (as squeaks and chirps) and amazingly enough, the toy whistle that came with Captain Crunch had the exact right frequencies. Hence if you wanted to place a long distance phone call, you could toot on the whistle appropriately and simply turn off billing. There was no way for the phone company to stop this either. This is canonical example of why sending control information over the same wire as content is a bad idea. Now, back to ISO 6429...

- Unicode support for certain symbols. type **`)help unicode`** in the workspace for more.
- Rudimentary command completion. You can type a few characters of a built-in function and hit the tab key. This will either complete *the signature* of the function or give you a couple of options:

```
  rem
remap([1,2,3])
remove([1])
```

After typing **`rem`** the tab key was used, resulting in the two options being presented. Note that one of those would be set, so typing **`rema`** and hitting the tab key results in

```
   remap([1,2,3])
```

Note especially that all of this is native Java so it is supported on all platforms.

^c exits, it does not copy to the clipboard.

In most operating systems, hitting ^c does not send the control character, but instead issues a SIGTERM (which means the operating system is shutting down the JVM and is notifying the JVM to that effect, hence it is unrecoverable). As such, on many OS's there is really nothing that can be done to stop it. QDL tries to intercept it if possible, but in general you should be aware of this and, for instance, not try to use ^c to copy things to the clipboard. Use the mouse if possible or see if there is another key sequence (e.g. ^C – control + shift + c in unix) that does that.

Paste

The ANSI terminal does have a mechanism for accessing the system clipboard (if there is one). There are two workarounds for pasting text.

Paste a line or less of text ^v

You can use ^v to paste text. This actually has QDL go out to the clipboard, grab the text and process it in the background like it was typed in, then print it. So if there are embedded line feeds, these are interpreted as well. This is in contrast to than many terminals allow you to paste using the mouse or something else (such as ^V – upper case V – in Ubuntu) – they do this by intercepting the keystroke (so we cannot do anything about it) and simply typing it in really fast as input. As indicated above, this will probably fails for anything other than short text., possibly crashing the JVM and therefore QDL.

Pasting lots of text: ^p. Toggling paste mode.

If you need to paste a large amount of text (e.g. paste in a whole block of QDL to run) you can toggle paste mode with ^p. Hitting ^p a second time exits paste mode. Once in paste mode, ^v faithfully pastes the entire contents of the clipboard. This will change the font color and print a message. In the next example, a couple of lines of QDL are pasted then run:

```
<paste mode on. ^v pastes from clipboard, ^p toggles paste mode>
  f(x) → 3×x^2÷5;
  f(⟦-1;1;5⟧);
<paste mode off>
[0.6,0.15,0.0E0,0.15,0.6]
```

and the entire block of code you paste in will only execute once you hit return (you can do that in inside paste mode or, as in the example above, paste the text, exit paste mode, then hit enter.

Using the line editor, external editors

The line editor is aware of ^v, so you can paste in large quantities of text that way.

External editors like Andy's Editor, vim and nano are full screen and take over when you switch to them, so special characters and paste mode are not supported there. Oh and another plus with the line editor it that is has its own clipboard system for its use, completely independent of any system clipboard, so no matter where you are running it, copy, cut and paste are always available.