Pandoc based J Syntax Highlighting

John MacFarlane's excellent command line utility Pandoc is a Haskell program that converts to and from various text markup languages. Pandoc's help option lists its supported input and output formats.

The following examples are Linux bash shell commands. Windows shell commands are identical.

```
$ pandoc --help
pandoc [OPTIONS] [FILES]
Input formats:  native, json, markdown, markdown+lhs, rst, rst+lhs, docbook,
                textile, html, latex, latex+lhs
Output formats: native, json, html, html5, html+lhs, html5+lhs, s5, slidy,
                slideous, dzslides, docbook, opendocument, latex, latex+lhs,
                beamer, beamer+lhs, context, texinfo, man, markdown,
                markdown+lhs, plain, rst, rst+lhs, mediawiki, textile, rtf, org,
                asciidoc, odt, docx, epub
```

Pandoc performs some conversions better than others. It does a better job of turning markdown into LaTeX than LaTeX into markdown. It's also better at converting HTML into LaTeX than LaTeX into HTML. Pandoc works best when converting markdown, the simplest of its inputs, to other formats. In fact Pandoc does such a good job of converting markdown to HTML, HTML+MathJax, LaTeX and PDF that many writers are now saving their source documents as markdown text knowing they can easily produce other formats as needed.

As handy as Pandoc's markup conversions are this nifty tool also supports syntax highlighting for over a hundred programming languages. Unfortunately, my favorite language J is not on Pandoc's list of highlighted languages. [1] Where have I run into this problem before? Luckily for me Pandoc is an open source tool and Pandoc's author has made it easy to add new highlight languages.

Pandoc is a Haskell program. I've been aware of Haskell's existence for years but until I decided to take on this specialized Pandoc hack I had never studied or used the language. Usually when you set out to modify a large program in an unfamiliar programming language you're in for what can only be described as an *f'ing educational experience.* It's a testament to the quality of the Haskell's global libraries and standard tools that a complete Haskell novice can effectively tweak large Haskell programs. Here's what you have to do.

1. Install the Haskell Platform. The Haskell Platform is available for all the usual suspects. I've used both the Windows and Linux versions. I almost installed the Mac version on my wife's Mac but resisted the urge.
2. Get with the Cabal. Cabal is the main Haskell package distribution and build utility. Cabal comes with the Haskell Platform and is easily accessed

from the command line. Type `cabal --help` in your favorite shell to view the program's options.

3. Spend sometime playing with Hackage. Hackage contains a large set of Haskell packages including all the source code required to build Pandoc.

After installing the Haskell Platform and familiarizing yourself with Cabal try building Pandoc. This will thoroughly exercise your Haskell system. Instructions for building Haskell packages are here. After reading the package build instructions run the following in your command shell:

```
$ cabal update
$ cabal install pandoc
```

This will download, compile and install a number of Haskell packages. Where Cabal puts the packages depends on your operating system. Cabal saves Linux packages in a hidden local directory. On my machine they ended up in:

```
/home/john/.cabal/lib
```

If you managed to build Pandoc you're now ready to add a new highlighting language. Pandoc uses the `highlighting-kate` package for syntax highlighting. `highlighting-kate` works by reading a directory of Kate editor xml language regex based definition files and generating custom language parsers. We want to generate a custom J parser so we need to download `highlighting-kate` source and add a Kate xml definition file for J.

You can find such a J Kate file on the J Wiki here. Download this file by cutting and pasting and save it as `j.xml`. Now do the following.

1. Run the Pandoc version command `pandoc --version` of the Pandoc you just built to determine the version of the `highlighting-kate` package you need.

2. Use Cabal to unpack the required `highlighting-kate` package. This downloads the required package and creates a temporary subdirectory in your current directory that contains package source code.

   ```
   $ cabal unpack highlighting-kate-0.5.3.2
   Unpacking to highlighting-kate-0.5.3.2/
   ```

3. Move into the temporary subdirectory and copy the Kate `j.xml` file to the package's xml subdirectory.

   ```
   $ cd highlighting-kate-0.5.3.2
   $ cp ~/pd/blog/j.xml ~/temp/highlighting-kate-0.5.3.2/xml/j.xml
   ```

4. Configure the package.

```
$ cabal configure
Resolving dependencies...
Configuring highlighting-kate-0.5.3.2...
```

5. Build the `highlighting-kate` package.

```
$ cabal build
Resolving dependencies...
    ... (omitted) ...
```

6. If `highlighting-kate` builds without problems run the command.

```
$ runhaskell ParseSyntaxFiles.hs xml
Writing Text/Highlighting/Kate/Syntax/SqlPostgresql.hs
Writing Text/Highlighting/Kate/Syntax/Scala.hs
    ... (omitted) ...
```

`ParseSyntaxFiles` scans the package's xml subdirectory and generates language specific parsers. If all goes well you will find `J.hs` in this directory.

```
~/temp/highlighting-kate-0.5.3.2/Text/Highlighting/Kate/Syntax
```

`J.hs`, like all the files referred to in this post, are available in the files sidebar in the `haskell/pandoc` subdirectory.

7. Now rebuild the `highlighting-kate` package. This compiles your new `J.hs` parser file.

```
$ cabal build
Resolving dependencies...
    ... (omitted) ...
```

8. After rebuilding the package run the Cabal copy command to put the modified package in the expected library location.

```
$ cabal copy
Installing library in
/home/john/.cabal/lib/highlighting-kate-0.5.3.2/ghc-7.4.1
```

Now that the highlighting library is up to date we have to rebuild Pandoc. To do this mirror the steps taken to download and build the highlighting package.

1. Use Cabal to unpack the Pandoc package.

```
$ cd ~/temp
$ cabal unpack pandoc-1.9.4.2
Unpacking to pandoc-1.9.4.2/
```

2. Switch to the Pandoc subdirectory and configure the package.

```
$ cabal configure
Resolving dependencies...
[1 of 1] Compiling Main      ( Setup.hs, dist/setup/Main.o )
    ... (omitted) ...
```

3. Rebuild Pandoc.

```
$ cabal build
Building pandoc-1.9.4.2...
Preprocessing executable 'pandoc' for pandoc-1.9.4.2...
    ... (omitted) ...
```

   If all goes well a Pandoc executable will be written to this subdirectory.

```
~/temp/pandoc-1.9.4.2/dist/build/pandoc
```

4. You can check the new executable by running `pandoc --version`. The result should display J in the list of supported languages.

Now that we have a Pandoc that can highlight J we're almost ready to blog gaudy J code. However before doing this we need to install some custom CSS. Custom CSS is not available on free *WordPress.com* blogs. To apply custom coloring schemes get the custom package and learn how to use WordPress's custom CSS editor. As daunting as this sounds it's no problemo for my limited purposes. To enable tango style Pandoc syntax highlighting on your WordPress blog paste `tango.css` into the custom CSS editor, check the "Add my CSS to CSS stylesheet" button and then press the "Save Stylesheet" button. Now your WordPress blog will be sensitive to the HTML span tags generated by Pandoc.

To show that all this hacking works as intended you can check out the Pandoc generated versions of this blog post. I've posted the original markdown source with PDF, LaTeX and HTML versions. All these files are available via the files sidebar. You can generate the HTML version with the command:

```
$ pandoc -s --highlight-style=tango PandocJHighlight.markdown -o PandocJHighlight.html
```

To get other versions simply change the file extension of the output `-o` file.

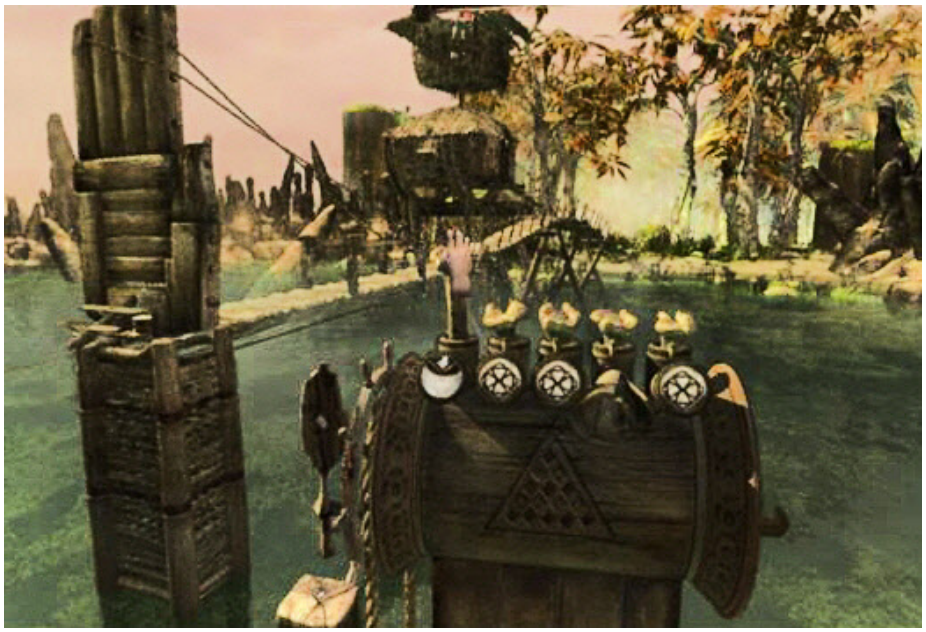

Figure 1: Bonebridge puzzle in MYST IV

Finally we are ready to post syntax highlighted J code. The following J verb generates all possible lock combinations for the MYST IV Bonebridge puzzle in Pandoc's tango style.

At one time I was a big fan of MYST computer games. I always enjoyed being lost in a beautiful puzzle which, if you discard the *beautiful* bit, is a pretty accurate description of my programmer day job.

```
bonebridge=:3 : 0

NB.*bonebridge  v--  lists  totem  symbol  permutations for  bone
NB. bridge.
NB.
NB. The  solution to  this MYST IV puzzle is similiar to the book
NB. shelf puzzle in Tomanha but requires far more  exploration of
NB. the age.
NB.
NB. You are confronted with  5  bones on the lock.  All the bones
NB. move independently. You can see the settings for 4 bones. One
NB. bone  has a  broken display.  The four  visible bones  have 8
NB. symbols on them in the  same order.  The  5th bone also has 8
NB. symbols and you can "safely" infer they are in the same order
NB. as the visible bones.
```

```
NB.
NB. Four bone symbols  match symbols found on totem poles
NB. distributed around the  age. There is a 5th totem pole but
NB. fruit eating mangrees  obscure  the  totem symbol and  I have
NB. never  seen it.  The  totem  poles are  associated  with  age
NB. animals. In addition to the totem poles  there is  a chart in
NB. the  mangree  observation  hut  that  displays  a  triangular
NB. pattern  of  paw  prints.  The  paw  prints  define an  animal
NB. ordering. The order  seems to be how  dangerous a  particular
NB. animal is;  big scary animals  are at the top and vegetarians
NB. are at the bottom.
NB.
NB. Putting the clues together you infer:
NB.
NB. a)  the  bridge  combination  is  some  permutation  of  five
NB. different symbols
NB.
NB. b) two possible symbol orders are given by the paw chart
NB.
NB. c) you know 5 symbols and the 4th is one of the remaining 4
NB.
NB. If this is  the  case  the number of  possible  lock settings
NB. shrinks from 32768 to the ones listed by this verb.
NB.
NB. monad:  bonebridge uuIgnore
NB.
NB.   bonebridge 0

NB. known in paw order
known=.    s: ' square triangle hourglass yingyang'
unknown=.  s: ' clover cross xx yy'

NB. all possible lock permutations
settings=. ~. 5 {."1 tapl known,unknown
assert. ((!8)%!8-5) = #settings

NB. possible ordering - we don't know
NB. what the fifth symbol is but it
NB. occurs in the 3rd slot
order=. 8#s:<''
order=. known (0 1 6 7)} order
order=. unknown (2 3 4 5)} order

NB. keep unknown only in 3rd slot
settings=. settings #~ -. +./"1 (0 1 3 4{"1 settings) e. unknown
```

```
settings=. settings #~ (2 {"1 settings) e. unknown

NB. strict row sequence adverb
srsm=.  1 : '*./"1 u/&> 2 <\"1 y'

NB. retain strictly increasing and strictly decreasing rows
grade=. order i. settings
settings #~ ((< srsm)"1 grade) +. (> srsm)"1 grade
)
```

1 J has its own syntax highlighting tools but they are not part of a document generation system. pandoc's highlighters elegantly feed into many output formats making them far more useful.