Tcl Filters for the Pandoc Document Processor°

Detlef Groth, Bioinformatics Group,
University of Potsdam, Germany

2021-11-17

°This presentation was originally given at SQLite and Tcl 2021 on November 17th, 2021. The code examples for Pikchr and PIC graphics where added later.

About myself

- Postdoc University of Potsdam since 2007
- Lectures:
  - Statistical Bioinformatics (R)
  - Databases and Practical Programming (Python and Tkinter)
  - Programming with R (GUI's with tcltk)
  - Machine Learning in Bioinformatics (R)
  - Programming Expertise (C/C++ GUI's with fltk (cpptk?))
  - Practical Bioinformatics (R)
- Documents:
  - lecture slides (LaTeX, Markdown)
  - scientific articles, analysis reports (LaTeX, Markdown)
- Approach literate programming:
  - R/Sweave/LaTeX
  - R/knitr/LaTex
  - pandoc with filters

Pandoc - universal document converter

- https://pandoc.org
- converter for many document formats
- can use filters to execute embedded programming code and display the results
- languages to be used within documents
- languages to write filters
- default languages to write filters:
  - Haskell
  - Lua
- other filter creation languages:
  - Python - panflute, pandocfilters, ...
  - Perl - Pandoc::Filter
  - Javascript ...
  - Tcl (not yet)

Pandoc and it's processing logic

- not a line based approach
- instead input document is transformed to an Abstract Syntax Tree (JSON)
- filters take the input stream and modify the JSON code
- finally pandoc converts the filter output stream to the requested document format


```
pandoc -f Markdown --filter filter-app -t beamer \
    -o beamer.pdf  --pdf-engine xelatex beamer.Rmd
```

pandoc-filter vs tmdoc

- *tmdoc* - last years presentation: https://www.youtube.com/watch?v=lfIPM5eyuVA
- *tmdoc* - execute Tcl code within Tex and Markdown files
- *mkdoc* - extract Markdown code from source code to create API documentation
- *tmdoc* vs pandoc-tcl-filter:
  - *tmdoc* - line oriented, limited set of in and output formats tex/md pdf, html
  - *pandoc* - AST oriented, many more input formats, more than 50 output formats
  - *tmdoc* - lightweight nice and small Tcl only approach
  - *pandoc* - heavy but powerful full feature approach

pandoc-tcl-filter

- requirements:
  - Tcl 8.6
  - rl_json (retains types)
- reads stdin
- check for code blocks with .tcl attribute
- evaluates code blocks and appends output to the JSON code

Example input:

Input in code blocks with triple! backticks:

````
```{.tcl}
parray tcl_platform
```
````

````
Document compiled at:
`tcl clock format [clock seconds]
   -format "%a %Y-%m-%d %I:%m %P"` CET.
````

Hint: single backticks currently only in unnested paragraphs currently.

Example output:

```
parray tcl_platform
```

```
tcl_platform(byteOrder)      = littleEndian
tcl_platform(engine)         = Tcl
tcl_platform(machine)        = x86_64
tcl_platform(os)             = Linux
tcl_platform(osVersion)      = 5.14.13-100.fc33.x86_64
tcl_platform(pathSeparator)  = :
tcl_platform(platform)       = unix
tcl_platform(pointerSize)    = 8
tcl_platform(threaded)       = 1
tcl_platform(user)           = groth
tcl_platform(wordSize)       = 8
```

Document compiled at: Mon 2021-11-22 04:11 pm CET.

Code chunk settings

- major chunk setting:
  - *label=labelname* chunk name, important for debugging
  - *echo=true* - should the code be shown
  - *results="show"* - what to do with the results either "show", "hide", "asis"
- code chunks settings will be converted to a Tcl dictionary

Inline codes with single backticks should be just short Tcl statements.

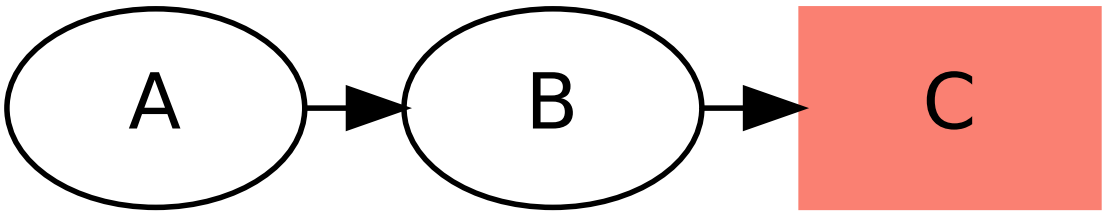
Graphviz dot/neato

Embed dot/neato code and display the results
(triple backticks required !!):

````
```{.dot ext=pdf}
digraph G {
    { rank=same; A ; B ; C }
    A -> B -> C
    C[shape=box,style=filled,color=salmon];
}
```
````

Output

```
digraph G {
    { rank=same; A ; B ; C }
    A -> B -> C
    C[shape=box,style=filled,color=salmon];
}
```



The filter code I

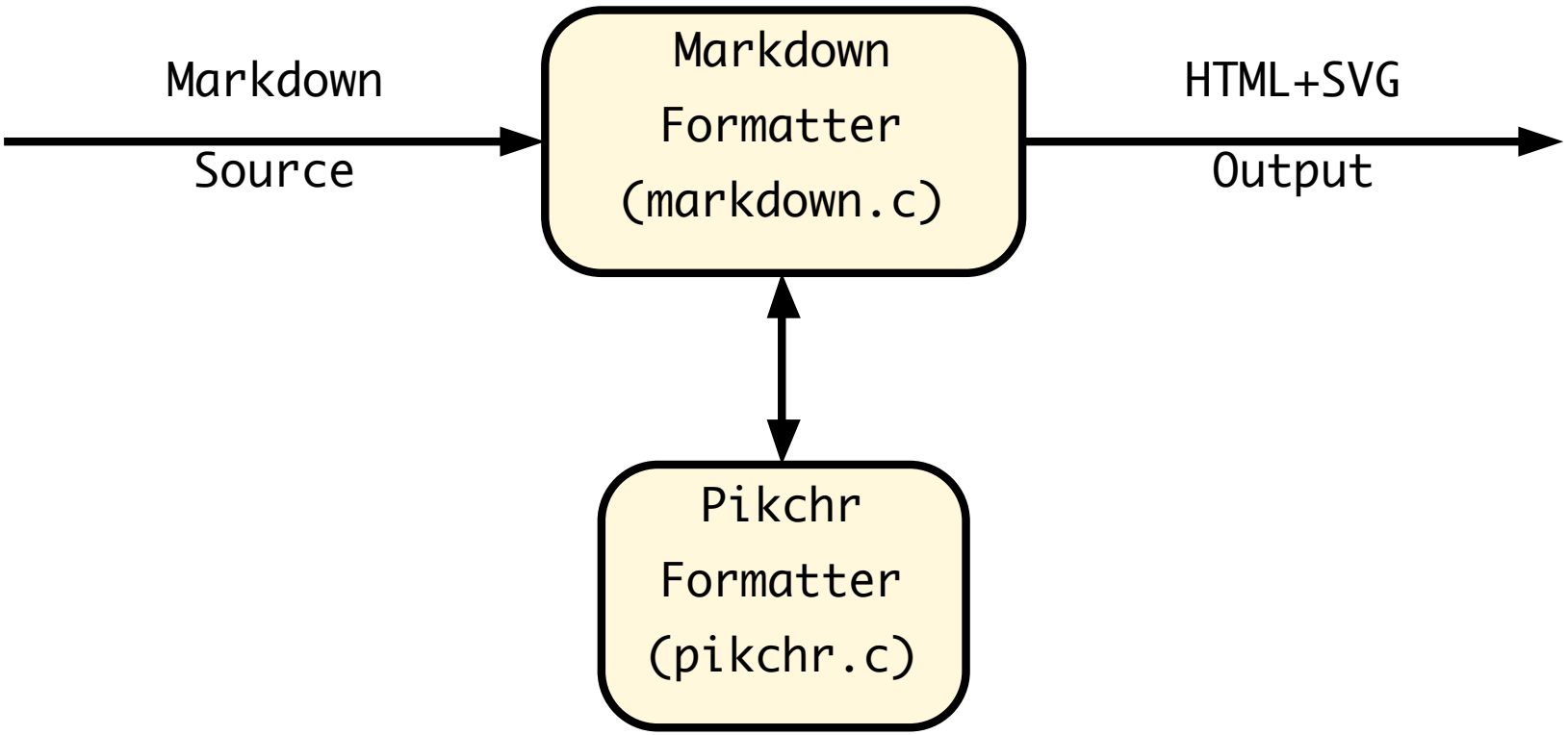
```
proc filter-dot {cont dict} {
  global n
  incr n
  set def [dict create results show eval true fig true  \
          include true imagepath images app dot \
          label null ext svg]
  set dict [dict merge $def $dict]
  set ret ""
  set owd [pwd]
  if {[dict get $dict label] eq "null"} {
    set fname [file join $owd [dict get $dict imagepath] dot-$n]
  } else { ;# ... }
  # ...
  set out [open $fname.dot w 0600]
  puts $out $cont
  close $out
  #...
}
```

The filter code II

```
proc filter-dot {cont dict} {
    # ...
    set res [exec [dict get $dict app] -T[dict get $dict ext] \
        $fname.dot -o $fname.[dict get $dict ext]]
    if {[dict get $dict results] eq "show"} {
        set res $res ;# should be empty
    } else {
        set res ""
    }
    set img ""
    if {[dict get $dict fig]} {
        if {[dict get $dict include]} {
            set img $fname.[dict get $dict ext]
        }
    }
    return [list $res $img]
}
```

Pikchr filter - example

```
arrow right 200% "Markdown" "Source"
box rad 10px fill cornsilk "Markdown" "Formatter" "(markdown.c)" fit
arrow right 200% "HTML+SVG" "Output"
arrow <-> down 70% from last box.s
box same "Pikchr" "Formatter" "(pikchr.c)" fit
```

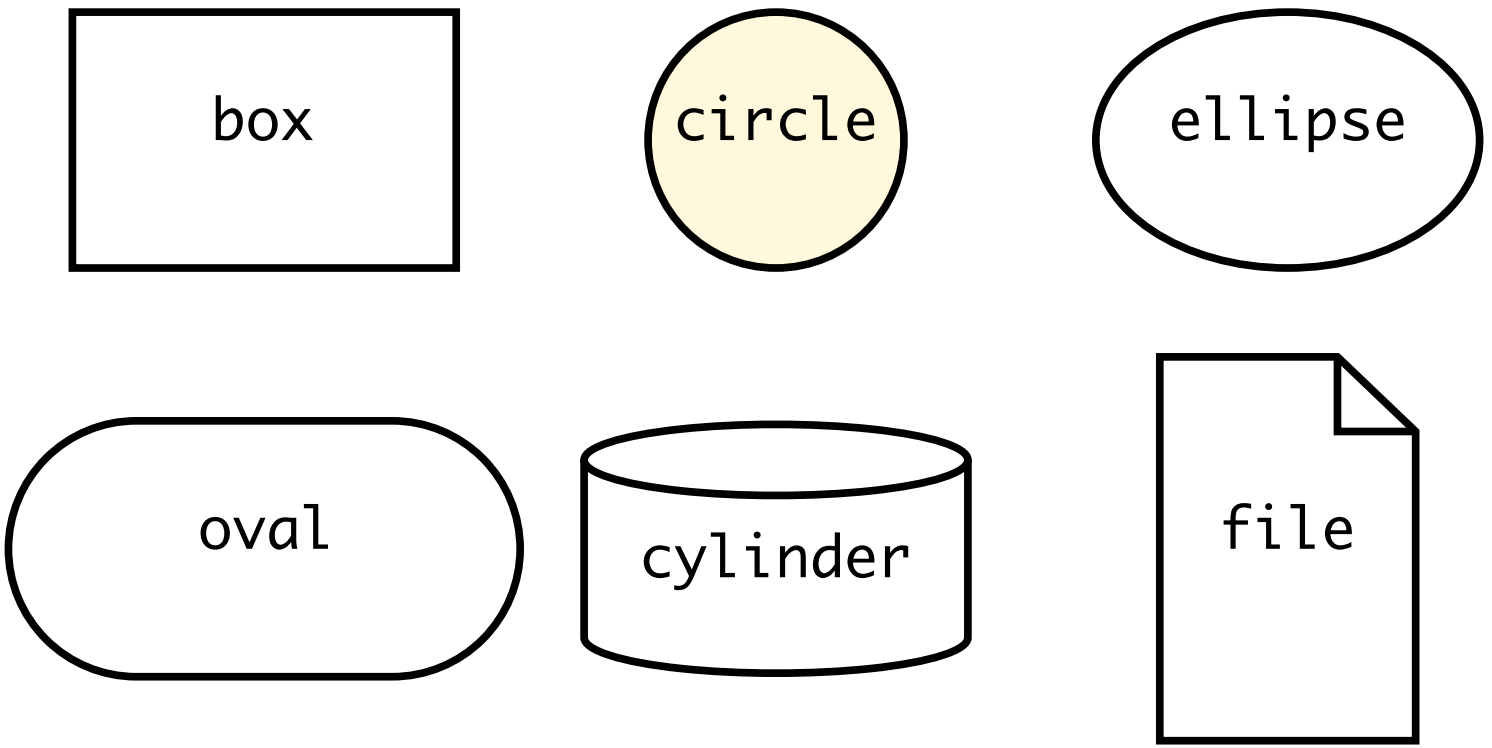


Pikchr filter - the code

````
```{.pik ext=pdf font=Monaco fontsize=12 width=400 height=220}
arrow right 200% "Markdown" "Source"
box rad 10px fill cornsilk "Markdown" "Formatter" "(markdown.c)" fit
arrow right 200% "HTML+SVG" "Output"
arrow <-> down 70% from last box.s
box same "Pikchr" "Formatter" "(pikchr.c)" fit
```
````

Fossil Pikchr filter - example

```
box "box"
circle "circle" fill cornsilk at 1 right of previous
ellipse "ellipse" at 1 right of previous
oval "oval" at .8 below first box
cylinder "cylinder" at 1 right of previous
file "file" at 1 right of previous
```

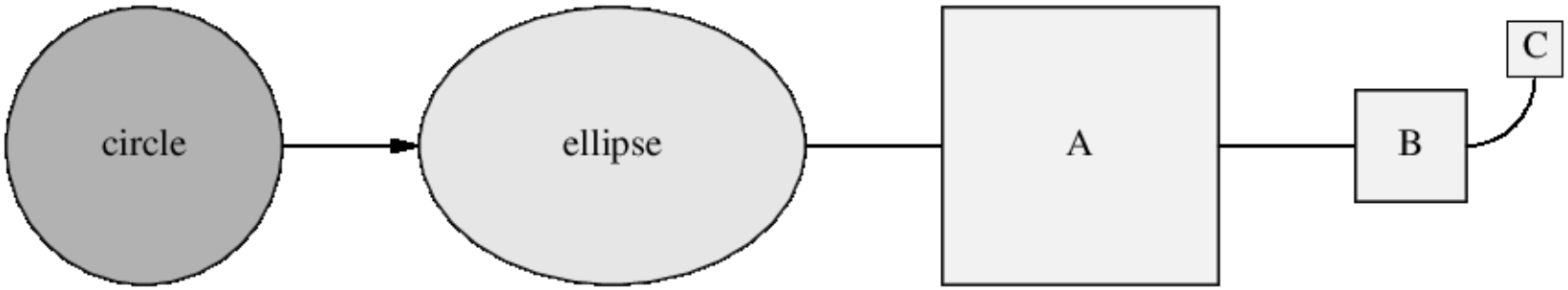


Pikchr filter - the code

````
```{.pikchr ext=pdf font=Monaco fontsize=16 width=400 height=180}
box "box"
circle "circle" fill cornsilk at 1 right of previous
ellipse "ellipse" at 1 right of previous
oval "oval" at .8 below first box
cylinder "cylinder" at 1 right of previous
file "file" at 1 right of previous
```
````

Pic filter - example

```
circle "circle" rad 0.5 fill 0.3; arrow ;
ellipse "ellipse" wid 1.4 ht 1 fill 0.1 ; line;
box wid 1 ht 1 fill 0.05 "A";
spline;
box wid 0.4 ht 0.4 fill 0.05 "B";
arc;
box wid 0.2 ht 0.2 fill 0.05 "C";
```



Pic filter - code

````
```{.pic ext=png}
circle "circle" rad 0.5 fill 0.3; arrow ;
ellipse "ellipse" wid 1.4 ht 1 fill 0.1 ; line;
box wid 1 ht 1 fill 0.05 "A";
spline;
box wid 0.4 ht 0.4 fill 0.05 "B";
arc;
box wid 0.2 ht 0.2 fill 0.05 "C";
```
````

Eqn filter - example

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Code (triple backticks):

````
```{.eqn label=eqnsam include=false}
x = {-b +- sqrt{b sup 2 - 4ac}} over 2a
```

![](images/eqnsam.png){#id width=130}
````

Errors ? Child interpreter reports them!

````
```{.tcl echo=true results="show"}
set y
```

```{.tcl}
set y 1
```
````

```tcl
set y
```

```
Error: can't read "y": no such variable
    while executing
"set y"
```

```tcl
set y 1
```

```
1
```

tpack

https://github.com/mittelmark/DGTcl/tree/master/apps/tpack

- Goal: single file applications build from simple Tcl script
- Starkit: sometimes virus scan problems on Windows
- Zipkit: additional packages on Tcl 8.6, 8.7 still alpha
- Solution: tpack (Tcl 8.4, Tcl 8.5, Tcl 8.6, Tcl 8.7)
  - single file, *tpack.tcl*, consisting of
  - tcllib code to extract tar files
  - tar extraction code into temp folder
  - application code
  - null character
  - attached tar archive (app.vfs folders can be used)
  - all in a single file *app.tcl*

tpack(ed) application layout

```
#!/usr/bin/env tclsh
----------------------------
 part of tcllib tar library
----------------------------
 code to extract tar
 pseudo starkit package
----------------------------
  application code
----------------------------
   null character
----------------------------
tar archive organized
    like starkit
----------------------------
```

Summary

- embedding Tcl code in Pandoc supported documents
- writing filters for other tools and languages with Tcl
- own filter should be placed in folder *filter* beside of the application
- example filter, directory *filter* in same directory script:
  - *filter-dot.tcl* - `proc filter-dot`
  - *filter-mtex.tcl* - `proc filter-mtex`
  - *filter-tsvg.tcl* - `proc filter-tsvg`
- standalone application pandoc-tcl-filter.tapp via *tpack.tcl*
- TODO's:
  - backticks in lists, Emph, Code, Para etc methods
  - terminal mode
  - Windows, OSX, using in RStudio
  - docx, odt and other WYSIWYG formats??

Links

- https://github.com/mittelmark/DGTcl
- https://wiki.tcl-lang.org/page/tpack
- https://wiki.tcl-lang.org/page/pandoc-tc-filter
- https://mittelmark.github.io/release/pandoc-tcl-filter.tapp
- inspired by https://wiki.tcl-lang.org/page/pandoc - rl_json filter example
- tpack https://raw.githubusercontent.com/mittelmark/DGTcl/master/apps/tpack/tpack.tcl
- inspired by https://wiki.tcl-lang.org/page/Another+Tcl+module+maker

Acknowledgment

- pandoc-tcl-filter:
  - Torsten Berg - initial pandoc/rl_json code on the Wiki
- tpack:
  - Aaron Faupell, Andreas Kupries Tcl only tar code
  - Richard Suchenwirth - initial wrapping code for dll's on the Wiki
  - D. Bohdan - Tcl only code for decompression of lz4 archives

Session Info

Tcl: `tcl set tcl_patchLevel`

Tcl: 8.6.10

Makefile

```
iconv -f ISO-8859-15 -t UTF-8 ../pandoc-filter.Rmd \
    > pandoc-filter-utf-8.Rmd
pandoc -f Markdown -t beamer --slide-level 2 \
    --include-in-header header.tex \
    --filter pandoc-tcl-filter.tapp --pdf-engine xelatex \
    -o ../pandoc-filter2.pdf pandoc-filter-utf-8.Rmd
```

Haskell filters ?

To compile Haskel filters: “install cabal, then install pandoc using cabal ...”

```
[groth@bariuke pandoc]$ cabal install pandoc
...
[groth@bariuke pandoc]$ du -hs  ~/.cabal/
1.9G    /home/groth/.cabal/
```

Or using pandoc-tcl-filter:

```
groth@piano(3:1002):build$ ls -lh pandoc-tcl-filter.tapp
-rwxr-xr-x. 1 groth groth 333K Nov 16 10:51
                                    pandoc-tcl-filter.tapp
```