Pandoc-Filters

The document converter pandoc is widely used in the R community. One feature of pandoc is that it can produce and consume JSON-formatted abstract syntax trees (AST). This allows to transform a given source document into JSON-formatted AST, alter it by so called filters and pass the altered JSON-formatted AST back to pandoc. This package provides functions which allow to write such filters in native R code. The package is inspired by the Python package `pandocfilters`.

To alter the AST, the JSON representations of the data structures building the AST have to be replicated. For this purpose, `pandocfilters` provides a set of constructors, with the goal to ease building / altering the AST.

0. Installation

Detailed information about installing pandoc, can be found at http://pandoc.org/installing.html.
NOTE: *There have been API-changes in pandoc 1.16. To account for this changes,* ***`pandocfilters`*** *needs to know the pandoc version. Therefore,* ***`pandocfilters`*** *will try to get the pandoc version at startup. However, it is also possible to get and set the pandoc version via the functions* ***`get_pandoc_version`*** *and* ***`set_pandoc_version`****.*

```
require("pandocfilters", quietly = TRUE, warn.conflicts = FALSE)
# Set the pandoc version.
get_pandoc_version()
```

```
## [1] 1.16
```

```
# Set the pandoc version.
set_pandoc_version(1.16)
```

1. Constructors

As mentioned before, constructors are used to replicate the pandoc AST in R. For this purpose, pandoc provides two basic types, **inline** elements and **block** elements. An extensive list can be found below.

To minimize the amount of unnecessary typing `pandocfilters` automatically converts character strings to pandoc objects of type `"Str"` if needed. Furthermore, if a single inline object is provided where a list of inline objects is needed `pandocfilters` automatically converts this inline object into a list of inline objects.
For example, the canonical way to emphasize the character string `"some text"` would be `Emph(list(Str("some text")))`. Since single inline objects are automatically transformed to lists of inline objects, this is equivalent to `Emph(Str("some text"))`. Since a character string is automatically transformed to an inline object, this is equivalent to `Emph("some text")`. In short, whenever a list of inline objects is needed one can also use a single inline object or a character string, and therefore the following three code lines are equivalent.

```
Emph(list(Str("some text")))
Emph(Str("some text"))
Emph("some text")
```

1.1. Inline Elements

1. `Str(x)`
2. `Emph(x)`
3. `Strong(x)`
4. `Strikeout(x)`
5. `Superscript(x)`
6. `Subscript(x)`
7. `SmallCaps(x)`
8. `Quoted(x, quote_type)`
9. `Cite(citation, x)`
10. `Code(code, name, language, line_numbers, start_from)`
11. `Space()`
12. `SoftBreak()`
13. `LineBreak()`
14. `Math(x)`
15. `RawInline(format, x)`
16. `Link(target, text, title, attr)`
17. `Image(target, text, caption, attr)`
18. `Span(attr, inline)`

1.2. Block Elements

1. `Plain(x)`
2. `Para(x)`
3. `CodeBlock(attr, code)`
4. `BlockQuote(blocks)`
5. `OrderedList(lattr, lblocks)`
6. `BulletList(lblocks)`
7. `DefinitionList(x)`
8. `Header(x, level, attr)`
9. `HorizontalRule()`
10. `Table(rows, col_names, aligns, col_width, caption)`
11. `Div(blocks, attr)`
12. `Null()`

1.3. Argument Constructors

1. `Attr(identifier, classes, key_val_pairs)`
2. `Citation(suffix, id, note_num, mode, prefix, hash)`
3. `TableCell(x)`

2. Alter the AST

To read / write / test the AST the following functions can be used:

2.1. Utility Functions

```
pandoc_to_json <- function(file, from="markdown") {
    args <- sprintf("-f %s -t json %s", from, file)
    system2("pandoc", args, stdout=TRUE, stderr=TRUE)
}

pandoc_from_json <- function(json, to) {
    args <- sprintf("%s | pandoc -f json -t %s", shQuote(json), to)
    system2("echo", args, stdout=TRUE, stderr=TRUE)
}

test_filter <- function(x, to="html") {
    d <- list(list(unMeta=setNames(list(), character())), x)
    pandoc_from_json(as.character(jsonlite::toJSON(d, auto_unbox=TRUE)), to=to)
}
```

2.2. Examples

2.2.1. Lower Case

The following example shows how to obtain the AST from a markdown file (`"lower_case.md"`) and convert every object of type `Str` to lower case.

```
caps <- function(key, value, ...) {
    if (key == "Str") return( Str( tolower(value) ) )
    return(NULL)
}

example_1 <- file.path(system.file(package = "pandocfilters"), "examples", "lower_case.md")
# the file before transformation
readLines(example_1)
```

```
## [1] "## 2.1 What is R?"
## [2] ""
## [3] "R is a system for statistical computation and graphics.  It consists of a"
## [4] "language plus a run-time environment with graphics, a debugger, access to"
## [5] "certain system functions, and the ability to run programs stored in script"
## [6] "files."
```

```
# read connection
input_connection <- textConnection(pandoc_to_json(example_1, from="markdown"))
# write connection
output_connection <- textConnection("modified_ast", open="w")

# apply filter
filter(caps, input=input_connection, output=output_connection)

# convert altered ast to markdown
pandoc_from_json(modified_ast, to="markdown")
```

```
## [1] "2.1 what is r?"
## [2] "--------------"
## [3] ""
## [4] "r is a system for statistical computation and graphics. it consists of a"
## [5] "language plus a run-time environment with graphics, a debugger, access"
## [6] "to certain system functions, and the ability to run programs stored in"
## [7] "script files."
```

```
close(input_connection)
close(output_connection)
```

3. Create a New Document

The constructor functions in the `pandocfilters` package can also be used to create a new document by reproducing the AST. To show how this is done we use the first paragraph of chapter 2.1 from the R-FAQ and apply different inline elements. The output is saved as HTML file.

In general, a `pandocfilter` document is a list of two elements:
- **Metadata**
- **Content**

The content is composed of lists of blocks, where each block itself usually consists of a list of inline elements.

NOTE: *If an inline object is needed and a character vector is given, `pandocfilters` automatically transforms the character string to an 'inline' object of type 'Str'.*

```
# Create a new document.
doc <- document()

# Create a non-standard writer function so we can look at the document while writing it.
cat_writer <- function(x, con, format) {
    args <- sprintf("%s | pandoc -f json -t %s", shQuote(as.character(x)), format)
    x <- system2("echo", args, stdout=TRUE, stderr=TRUE)
    cat(x, sep="\n")
}

# Append a Header and look at the document
args(doc$append_header)
```

```
## function (x, level = 1L, attr = Attr())
## NULL
```

```
doc$append_header( "R Basics" )
doc$write(con=NULL, format="html", writer=cat_writer)
```

```
## <h1>R Basics</h1>
```

```
# Append a level 2 Header
doc$append_header( "What is R?", level=2)

# Append Plain text with inline formating
x <- c(Emph("R"), Space(), "is a system for ")
x <- c(x, c(Strong("statistical computation"), Space(), Strikeout("and"), Space() ))
x <- c(x, c(Superscript("graphics"), ". ", LineBreak(), Subscript("It"), Space()))
x <- c(x, c(SmallCaps("consists"), Space(), Quoted("of", quote_type="SingleQuote")))
x <- c(x, c(Space(), Quoted("a", quote_type="DoubleQuote"), Space()))
x <- c(x, c(RawInline("html", "<i>language</i>"), " plus a run-time environment with"))
x <- c(x, c(" graphics, a debugger, access to ", "certain system functions,"))
```

```
x <- c(x, c(" and the ability to run programs stored in script files."))

doc$append_plain( x )
doc$write(con="test_1.html", format="html")
```

If we look at the output we see that in "test_1.html" the quotes don't look very nice. This can be fixed by using the function `astrapply`. In most cases it is best to look at the AST via `str` to see what should be replaced. Since the quote doesn't look good in the HTML case we replace it with `<q>text</q>`.

```
fix_quotes_fun <- function(x) RawInline("html", sprintf("<q>%s</q>", x$c))

fix_quotes <- function(type, content, ...) {
    if (type == "Quoted") {
        return( lapply(content[[-1]], fix_quotes_fun) )
    }
}

doc$doc <- astrapply(doc$doc, FUN=fix_quotes)
doc$write(con="test_2.html", format="html")
```

4. Create a Table

```
table <- document()
table$append_table(cars[1:4,])
table$write(con=NULL, format="markdown", writer=cat_writer)
```

```
##   speed   dist
##   ------- ------
##   4       2
##   4       10
##   7       4
##   7       22
```

```
table$write(con=NULL, format="html", writer=cat_writer)
```

```
## <table>
## <thead>
## <tr class="header">
## <th align="left">speed</th>
## <th align="left">dist</th>
## </tr>
## </thead>
## <tbody>
## <tr class="odd">
## <td align="left">4</td>
```

```
## <td align="left">2</td>
## </tr>
## <tr class="even">
## <td align="left">4</td>
## <td align="left">10</td>
## </tr>
## <tr class="odd">
## <td align="left">7</td>
## <td align="left">4</td>
## </tr>
## <tr class="even">
## <td align="left">7</td>
## <td align="left">22</td>
## </tr>
## </tbody>
## </table>
```

```
table$write(con=NULL, format="html5", writer=cat_writer)
```

```
## <table>
## <thead>
## <tr class="header">
## <th style="text-align: left;">speed</th>
## <th style="text-align: left;">dist</th>
## </tr>
## </thead>
## <tbody>
## <tr class="odd">
## <td style="text-align: left;">4</td>
## <td style="text-align: left;">2</td>
## </tr>
## <tr class="even">
## <td style="text-align: left;">4</td>
## <td style="text-align: left;">10</td>
## </tr>
## <tr class="odd">
## <td style="text-align: left;">7</td>
## <td style="text-align: left;">4</td>
## </tr>
## <tr class="even">
## <td style="text-align: left;">7</td>
## <td style="text-align: left;">22</td>
## </tr>
## </tbody>
## </table>
```

```
table$write(con=NULL, format="org", writer=cat_writer)
```

```
## | speed   | dist   |
## |---------+--------|
## | 4       | 2      |
## | 4       | 10     |
## | 7       | 4      |
## | 7       | 22     |
```

```
table$write(con=NULL, format="latex", writer=cat_writer)
```

```
## \begin{longtable}[c]{@{}ll@{}}
## \toprule
## speed & dist\tabularnewline
## \midrule
## \endhead
## 4 & 2\tabularnewline
## 4 & 10\tabularnewline
## 7 & 4\tabularnewline
## 7 & 22\tabularnewline
## \bottomrule
## \end{longtable}
```

```
table$write(con=NULL, format="rst", writer=cat_writer)
```

```
## +---------+--------+
## | speed   | dist   |
## +=========+========+
## | 4       | 2      |
## +---------+--------+
## | 4       | 10     |
## +---------+--------+
## | 7       | 4      |
## +---------+--------+
## | 7       | 22     |
## +---------+--------+
```

```
table$write(con=NULL, format="asciidoc", writer=cat_writer)
```

```
## [cols=",",options="header",]
## |===========
## |speed |dist
## |4 |2
## |4 |10
## |7 |4
## |7 |22
## |===========
```