Package 'highlight'

January 18, 2023

Title Syntax Highlighter

Version 0.5.1

Description Syntax highlighter for R code based on the results of the R parser. Rendering in HTML and latex markup. Custom Sweave driver performing syntax highlighting of R code chunks.

License GPL (>= 3)

URL `https://github.com/hadley/highlight`

BugReports `https://github.com/hadley/highlight/issues`

Depends R (>= 3.2)

Imports grDevices, tools

Encoding UTF-8

RoxygenNote 7.2.3

NeedsCompilation yes

Author Hadley Wickham [cre],
Romain Francois [aut],
Andre Simon [ctb]

Maintainer Hadley Wickham `<hadley@rstudio.com>`

Repository CRAN

Date/Publication 2023-01-18 12:00:09 UTC

R topics documented:

---

highlight-package *Syntax Highlighter for R*

Description

Syntax highlighter for R based on output from the R parser

See Also

The main function of the package is `highlight`.

`highlight` delegates rendering the document to `renderer`s, such as the `renderer_latex` or `renderer_html` and is helped by a detective to make sense of the results from the parser. The package ships a `simple_detective`.

The package also defines a custom sweave driver (`HighlightWeaveLatex`) for latex based on the standard sweave latex driver (`RweaveLatex`) using `highlight` to perform syntax highlighting of R code chunks.

Examples

```
## Not run:
tf <- tempfile()
dump( "glm" , file = tf )

# rendering in html
highlight( tf, output = stdout(),
renderer = renderer_html() )

# rendering in latex
highlight( tf, output = stdout(),
renderer = renderer_latex() )

# Sweave driver using syntax highlighting
if( require( grid ) ){
```

```
v <- vignette( "grid", package = "grid" )$file
file.copy( v, "grid.Snw" )
Sweave( "grid.Snw", driver= HighlightWeaveLatex() )
system( "pdflatex grid.tex" )
if (.Platform$OS.type == "windows"){
shell.exec( "grid.pdf" )
} else {
system(paste(shQuote(getOption("pdfviewer")), "grid.pdf" ),
wait = FALSE)
}
}

unlink( tf )

## End(Not run)
```

`boxes_latex` *Creates the set of latex boxes*

Description

This function returns the set of latex boxes definitions that should be included in the document preamble. The latex renderer includes these definitions automatically when the document argument is TRUE, but not otherwise.

Usage

```
boxes_latex()
```

Value

A character vector containing latex definitions for boxes used by the latex renderer

See Also

`translator_latex` translates text into markup that makes use of these boxes

`css.parser` *Minimal CSS parser*

Description

Minimal CSS parser

Usage

```
css.parser(file, lines = readLines(file))
```

Arguments

`file`	file to parse
`lines`	text lines to parse

Value

A list with one element per style class declaration. Each element is a list which has one element per CSS setting (‘`color`’, ‘`background`’, ...)

Note

The parser is very minimal and will only identify CSS declarations like the following :

```
.classname{
setting1 : value ;
setting2 : value ;
}
```

The line where a declaration occurs must start with a dot, followed by the name of the class and a left brace. The declaration ends with the first line that starts with a right brace. The function will warn about class names containing numbers as this is likely to cause trouble when the parsed style is translated into another language (e.g. latex commands).

Within the css declaration, the parser identifies setting/value pairs separated by ‘`:`’ on a single line. Each setting must be on a seperate line.

If the setting is ‘`color`’ or ‘`background`’, the parser then tries to map the value to a hex color specification by trying the following options: the value is already a hex color, the name of the color is one of the 16 w3c standard colors, the name is an R color (see `colors`), the color is specified as ‘`rgb(r,g,b)`’. If all fails, the color used is black for the ‘`color`’ setting and ‘`white`’ for the ‘`background`’ setting.

Other settings are not further parsed at present.

`external_highlight` *Multi-language source code highlighter*

Description

Multi-language source code highlighter

Usage

```
external_highlight(
  file,
  outfile = stdout(),
  theme = "kwrite",
  lang = NULL,
  type = "HTML",
  line_numbers = FALSE,
  doc = TRUE,
  code
)
```

Arguments

`file`	Source file to highlight
`outfile`	Destination of the highlighted code. When `NULL`, the code is simply returned as a character vector
`theme`	One of the themes. See `highlight_themes` for the list of available themes.
`lang`	The language in which the code is to be interpreted. If this argument is not given, it will be deduced from the file extension.
`type`	Output format. See `highlight_output_types` for the list of supported output types.
`line_numbers`	if `TRUE`, the result will include line numbers
`doc`	if `TRUE`, the result is a stand alone document, otherwise, just a portion to include in a document
`code`	If given, then the source code is not read from the file

Value

Nothing if `outfile` is given, with the side effect of writing into the file. The result as a character vector if outfile is NULL

See Also

`highlight` to highlight R code using the information from the parser

---

`formatter_html` *html formatter*

Description

Wraps tokens into span tags with the class corresponding to the style

Usage

```
formatter_html(tokens, styles, ...)
```

Arguments

`tokens`	tokens to wrap
`styles`	styles to give to the tokens
`...`	ignored

See Also

renderer_html

formatter_latex *Latex formatter*

Description

Combines tokens and styles into a latex command

Usage

```
formatter_latex(tokens, styles, ...)
```

Arguments

tokens	vector of okens
styles	vector of styles
...	ignored

Value

A vector of latex commands

See Also

renderer_latex

Examples

```
formatter_latex( "hello world", "blue" )
```

getStyleFile *helper function to get a style file*

Description

helper function to get a style file

Usage

```
getStyleFile(name = "default", extension = "css")
```

Arguments

`name`	the name of the style file to look for
`extension`	the file extension (css, sty, or xterm)

Details

the search goes as follows: first the current working directory then the directory ~/.R/highlight, then the stylesheet directory in the installed package

Value

the name of the first file that is found, or NULL

`header_html` *html renderer header and footer*

Description

these functions build the header function and the footer function used by the html renderer

Usage

```
header_html(document, stylesheet)

footer_html(document)
```

Arguments

`document`	logical. If `TRUE` the built header and footer functions will return the beginning and end of a full html document. If `FALSE`, the built functions will only return the opening and closing '`<pre>`' tags.
`stylesheet`	stylesheet to use. See `getStyleFile` for details on where the stylesheet can be located.

Value

header and footer functions.

See Also

`renderer_html` uses these functions to create a renderer suitable for the '`renderer`' argument of `highlight`

Examples

```
h <- header_html( document = FALSE )
h()
h <- header_html( document = TRUE, stylesheet = "default")
h()
f <- footer_html( document = TRUE )
f()
f <- footer_html( document = FALSE )
f()
```

header_latex *latex header and footer*

Description

These functions return appropriate header and footer functions for the latex renderer

Usage

```
header_latex(document, styles, boxes, minipage = FALSE)

footer_latex(document, minipage = FALSE)
```

Arguments

document	logical. If TRUE the header and footer functions will create the full document (including preamble with boxes and styles)
styles	a vector of style definitions to include in the preamble if document is TRUE
boxes	a vector of boxes definitions to include in the preamble if document is TRUE
minipage	if TRUE, the highlighted latex is included in a minipage environment

Value

A function is returned, suitable for the header or footer argument of the latex renderer

Examples

```
h <- header_latex( document = FALSE )
h()
f <- footer_latex( document = FALSE )
f()
```

`highlight` *syntax highlighting based on the R parser*

Description

The `highlight` function performs syntax highlighting based on the results of the `parse` and the investigation of a detective.

Usage

```
highlight(
  file,
  output = stdout(),
  detective = simple_detective,
  renderer,
  encoding = "unknown",
  parse.output = parse(file, encoding = encoding, keep.source = TRUE),
  styles = detective(parse.output),
  expr = NULL,
  final.newline = FALSE,
  showPrompts = FALSE,
  prompt = getOption("prompt"),
  continue = getOption("continue"),
  initial.spaces = TRUE,
 size = c("normalsize", "tiny", "scriptsize", "footnotesize", "small", "large", "Large",
    "LARGE", "huge", "Huge"),
  show_line_numbers = FALSE,
  ...
)
```

Arguments

`file`	code file to parse. This is only used if the `parse.output` is given
`output`	where to write the rendered text. If this is anything else than the default (standard output), the `sink` function is used to redirect the standard output to the output.
`detective`	the detective chooses the style to apply to each token, basing its investigation on the results of the `parse`
`renderer`	highlight delegates rendering the information to the renderer. This package includes html and latex renderers. See `renderer_html` and `renderer_latex`
`encoding`	encoding to assume for the file. the argument is directly passed to the `parse`.
`parse.output`	output from the `parse`. If this is given, the arguments `file` and `encoding` are not used
`styles`	result of the detective investigation. A character vector with as many elements as there are tokens in the parser output

`expr`	In case we want to render only one expression and not the full parse tree, this argument can be used to specify which expression to render. The default (NULL) means render all expressions. This feature is used by the sweave driver shipped with this package. See `HighlightWeaveLatex`
`final.newline`	logical. Indicates if a newline character is added after all tokens.
`showPrompts`	if TRUE, the highlighted text will show standard and continue prompt
`prompt`	standard prompt
`continue`	continue prompt
`initial.spaces`	should initial spaces be displayed or skipped.
`size`	font size. only respected by the latex renderer so far.
`show_line_numbers`	logical. When TRUE, line numbers are shown in the output.
`...`	additional arguments, currently ignored.

Value

The resulting formatted text is returned invisibly. It is also written to the output if the output is not `NULL`

See Also

`renderer_html` and `renderer_latex` are the two implementation of renderers currently available in this package.

`simple_detective` is an example detective which does a very simple investigation.

Examples

```
## Not run:
tf <- tempfile()
dump( "jitter", file = tf )
highlight( file = tf, detective = simple_detective,
renderer = renderer_html( document = TRUE ) )
highlight( file = tf, detective = simple_detective,
renderer = renderer_latex( document = TRUE ) )


## End(Not run)
```

`HighlightWeaveLatex` *Sweave driver performing syntax highlighting*

Description

Sweave driver using the highlight latex renderer to perform syntax highlighting of input R code in sweave chunks.

Usage

```
HighlightWeaveLatex(
  boxes = FALSE,
  bg = rgb(0.95, 0.95, 0.95, maxColorValue = 1),
  border = "black",
  highlight.options = list(boxes = boxes, bg = bg, border = border)
)
```

Arguments

`boxes`	if `TRUE`, code blocks are wrapped in boxes.
`bg`	background color for code boxes.
`border`	color to use for the border of code boxes.
`highlight.options`	Can be used instead of the other arguments to set the `boxes`, `bg` and `border` settings.

Details

This sweave driver is very similar to standard driver that is included in '`utils`'. The difference is that input R code and verbatim output is rendered using `highlight` enabling syntax highlighting of R code.

Instead of using '`Sinput`' and '`Soutput`' commands, this driver uses '`Hinput`' and '`Houtput`' and defines these commands at the very beginning of the document, letting the user the option to overwrite them as necessary.

Latex boxes defined by the latex renderer (`renderer_latex`) and style definitions needed are also written at the beginning of the document.

Because highlight does not use verbatim environments, the user of this driver can freely redefine the '`Hinput`', '`Houtput`' and '`Hchunk`' environments to achieve greater control of the output latex document than with the standard driver.

Value

A sweave driver, suitable for the '`driver`' argument of `Sweave`

Examples

```
## Not run:
# using the driver on the grid vignette
require( grid )
v <- vignette( "grid", package = "grid" )$file
file.copy( v, "grid.Snw" )
Sweave( "grid.Snw", driver= HighlightWeaveLatex() )

## End(Not run)
```

highlight_output_types *List of available output types supported by external_highlight*

Description

List of available output types supported by `external_highlight`

Usage

```
highlight_output_types()
```

Value

A character vector with the list of supported types

highlight_themes *List of themes supported by external_highlight*

Description

List of themes supported by `external_highlight`

Usage

```
highlight_themes()
```

Value

A character vector with the names of the themes

Hweave *Weaving and Tangling with syntax highlighting*

Description

`Hweave` and `Htangle` are similar to `Sweave` and `Stangle`, but they take advantage of the custom driver shipped with this package

Usage

```
Hweave(
  file,
  driver = HighlightWeaveLatex(),
  syntax = HweaveSyntaxNoweb,
  encoding = "",
  ...
)

Htangle(
  file,
  driver = HighlightTangle(),
  syntax = HweaveSyntaxNoweb,
  encoding = "",
  ...
)
```

Arguments

`file`	Path to Sweave source file
`driver`	The actual workhorse, see the Details section in `Sweave`
`syntax`	`NULL` or an object of class `SweaveSyntax` or a character string with its name. See the section `Syntax Definition` in `Sweave`
`encoding`	The default encoding to assume for `file`
`...`	Further arguments passed to the driver's setup function.

Details

These functions exist for the purpose of the `\VignetteEngine` option in vignette introduced in R 3.0.0

`highlight` loads the `highlight` vignette engine at load time. Client packages must declare to use it with the `VignetteBuilder` field in their `DESCRIPTION` file

The vignette engine looks for files matching the pattern `"[.][hHrRsS]nw$"` although in order to distinguish vignettes using this engine and the default Sweave engine, the recommandation is to use vignette with the `".Hnw"` extension.

`renderer` *highlight renderer*

Description

The function builds a renderer, suitable for the renderer argument of the highlight function. In the highlight process, renderers are responsible to render the information in the target markup language.

Usage

```
renderer(translator, formatter, space, newline, header, footer, ...)
```

Arguments

`translator`	This argument should be a function with one argument. The translator needs to work token characters so that they display nicely in the target markup language.
`formatter`	The formatter should be a function with at least two arguments: the tokens and the styles. These two arguments are supplied to the formatter by the highlight function. The formatter should wrap tokens and styles into the target markup language. For example, the formatter used by the html renderer makes a ‘`<span>`’ tag of ‘`class`’ given by the ‘`styles`’ and content given by the ‘`token`’.
`space`	This should be a function with no argument. The output of this function should be a character vector of length one giving the representation of a space character in the target language. For example, in the latex renderer, the function returns ‘`"{\ }"`’.
`newline`	This should be a function with no argument. The output of the function is a character vector of length one giving the representation of a newline character in the target language.
`header`	This should be a function with no argument. The output of this function is a character vector of arbitrary length. The elements of the output are written before the highlighted content. headers and footers are used to embed the highlighted tokens into some markup. For example, the header used in the html renderer starts a ‘`<pre>`’ tag that is closed by the footer. headers and footer might also be used to write style definitions such as CSS, STY, ...
`footer`	This should be a function with no argument. The output of this function is written after all tokens.
`...`	Additional arguments. This might be used to store additional renderer specific objects.

Details

Implementations of renderers should call this function to ensure that a proper renderer is created. At the moment, no checking is performed to ensure that the built object complies with the expected interface, but this is very likely to change.

Value

A ‘`renderer`’ object. Renderer objects define the interface expected by the `highlight` function. At the moment, a renderer object is a list of class ‘`renderer`’ containing elements: ‘`translator`’, ‘`formatter`’, ‘`space`’, ‘`newline`’, ‘`header`’ and ‘`footer`’.

See Also

The `renderer_html` implements a renderer using html markup, ‘`<span>`’ tags and CSS.

The `renderer_latex` implements a latex renderer.

`simple_detective` *Simple detective*

Description

This detective only uses semantic information to make its investigation.

Usage

```
simple_detective(x, ...)
```

Arguments

`x`	output of the parser. The detective is only interested in the '`token`' column of the data.
`...`	ignored

Value

a vector of styles grouping similar tokens together

Examples

```
## Not run:
p <- parse( text = deparse( jitter ), keep.source=TRUE )
simple_detective( p )

## End(Not run)
```

`space_latex` *LaTeX renderer*

Description

renderer implementation targetting latex markup. The result markup uses the latex '`alltt`' package to achieve true type renderering and therefore does not depend on verbatim-like environments.

Usage

```
space_latex()

newline_latex()

renderer_latex(
  document = TRUE,
  boxes = boxes_latex(),
  translator = translator_latex,
  formatter = formatter_latex,
  space = space_latex,
  newline = newline_latex,
  stylesheet = "default",
  styles = styler(stylesheet, "sty", styler_assistant_latex),
 header = header_latex(document, styles = styles, boxes = boxes, minipage = minipage),
  footer = footer_latex(document, minipage = minipage),
  minipage = FALSE,
  ...
)
```

Arguments

`document`	logical. Should the renderer create the full document or only the code section, assuming the document is already created. Using FALSE is used by the sweave driver shipped with this package.
`boxes`	a function that returns definitions of latex boxes used for non standard characters. The reason for using boxes is that some character need to be escaped to be rendered, and unfortunately, escaping turns alltt off, which does not produce satisfying rendering. This argument is used by the header function when the document argument is TRUE. It is also used in the sweave driver at the very beginning of the document
`translator`	translation of characters into latex markup. See `translator_latex` for details
`formatter`	latex formatter. Tokens are wrapped into a latex command related to the style they should honor.
`space`	returns a space character that does not get reduced by latex
`newline`	returns a newline character
`stylesheet`	stylesheet to use.
`styles`	style definitions inferred from the parsing of the stylesheet. See `styler` and `styler_assistant_latex`.
`header`	returns the header. If the document argument is TRUE, the header contains the style definitions and the boxes definitions. If it is FALSE, a minimal header is produced to turn alltt on. In the latter case, boxes and style definitions are assumed to have been inserted already, latex will not compile the document otherwise.
`footer`	returns the footer. Depending on the document argument, either a minimal footer is produced (turning off alltt) or the full latex document is closed.
`minipage`	if TRUE, the highlighted latex is included in a minipage environment
`...`	Additional arguments

Value

a ‘`renderer`’ object, suitable for the ‘`renderer`’ argument of `highlight`.

Examples

```
## Not run:
r <- renderer_latex(document = T )
r$space()
r$newline()
r$boxes()
r$translator( "# the hash symbol gets a latex box" )

## End(Not run)
```

`styler` *Style definition generator*

Description

This generates style definitions either by including a language specific style file (e.g. sty file for latex) or by parsing a css stylesheet

Usage

```
styler(stylesheet, extension = "css", assistant)
```

Arguments

`stylesheet`	name of the stylesheet
`extension`	extension of the language specific format for the stylesheet.
`assistant`	function to which the styler delegates understanding of the parser output

Details

First, the function attempts to retrieve a language specific stylesheet using the `getStyleFile` function. If a language specific stylesheet is found, it returns the content of the file as a character vector.

Second, the function attemps to find a css stylesheet using `getStyleFile`, parse the css declarations using the `css.parser` function, and delegates to the ‘`assistant`’ which is responsible to translate the results of the css parser into language specific declarations.

Value

a character vector containing style declarations in the target language

See Also

`styler_assistant_latex` gives a concrete implementation of the assistant for the latex language

Examples

```
## Not run:
styler( "default", "sty", styler_assistant_latex )

## End(Not run)
```

---

styler_assistant_latex
latex styler assistant

Description

This function takes the output of the `css.parser` and produces latex style definitions from it.

Usage

```
styler_assistant_latex(x)
```

Arguments

`x`	output from `css.parser`

Details

The function create a new latex command for each css declaration, i.e. each item of the list ‘`x`’ it is passed.

The assistant currently honours the following css settings: color, ‘`text-decoration:underline`’, ‘`font-weight:bold[er]`’ and ‘`font-style:italic`’

Value

a vector of latex style definitions corresponding to (a subset of) the output of the parser

See Also

`styler`

translator_html *html renderer using span tags and CSS*

Description

implementation of the renderer that renders the information as a series of '<span>' html tags

Usage

```
translator_html(x, size)

space_html()

newline_html()

renderer_html(
  document = TRUE,
  translator = translator_html,
  formatter = formatter_html,
  space = space_html,
  newline = newline_html,
  header = header_html(document, stylesheet),
  footer = footer_html(document),
  stylesheet = "default",
  ...
)
```

Arguments

x	argument to the translator. Returned as is.
size	font size. ignored
document	logical. Indicates if the renderer should render a full document or simply a '<pre>' section containing the highlighted tokens. This argument is used by the header_html and footer_html to build appropriate header and footer.
translator	Since the highlighted tokens are wrapped in a '<pre>' tag, no further translation is needed.
formatter	html formatter. creates '<span>' tags for all tokens. See formatter_html
space	returns a space character
newline	returns a newline character
header	html header. Depending on the 'document' argument, this will be a function building a the beginning of a complete html document (starting with '<html>') including css definitions or simply a function returning '<pre>' enabling the renderer to be used to just render the syntax as part of a bigger document.
footer	html footer. Depending on the 'document' argument, this will either close the full document (close the '</html>' tag) or simply close the '</pre>' tag.

`stylesheet`	stylesheet to use. This is used by the header when document is TRUE. The content of the stylesheet is copied verbatim into a '`<style>`' tag in that case. See `getStyleFile` for details on where the stylesheet can be located
`...`	Additional arguments. unused.

Value

A renderer capable suitable for the '`renderer`' argument of `highlight`

See Also

`renderer` for a description of the interface this renderer is implementing.

`highlight` takes a renderer argument to which it delegates rendering.

`translator_latex` *LaTeX translator*

Description

This function translates character vectors so that they nicely print in LaTeX. In particular this uses latex boxes.

Usage

```
translator_latex(
  x,
  size = c("normalsize", "tiny", "scriptsize", "footnotesize", "small", "large", "Large",
    "LARGE", "huge", "Huge")
)
```

Arguments

`x`	text to translate
`size`	font size

Value

translated text

See Also

the latex renderer: `renderer_latex` uses this translator.

Index