Package ‘htmltidy’

October 3, 2019

**Title**  Tidy Up and Test XPath Queries on HTML and XML Content

**Version**  0.5.0

**Encoding**  UTF-8

**Maintainer**  Bob Rudis  <bob@rud.is>

**Description**  HTML documents can be beautiful and pristine. They can also be wretched, evil, malformed demon-spawn. Now, you can tidy up that HTML and XHTML before processing it with your favorite angle-bracket crunching tools, going beyond the limited tidying that 'libxml2' affords in the 'XML' and 'xml2' packages and taming even the ugliest HTML code generated by the likes of Google Docs and Microsoft Word. It's also possible to use the functions provided to format or ``pretty print'' HTML content as it is being tidied. Utilities are also included that make it possible to view formatted and ``pretty printed'' HTML/XML content from HTML/XML document objects, nodes, node sets and plain character HTML/XML using 'vkbeautify' (by Vadim Kiryukhin) and 'highlight.js' (by Ivan Sagalaev). Also (optionally) enables filtering of nodes via XPath or viewing an HTML/XML document in ```tree``` view using 'XMLDisplay' (by Lev Muchnik). See <https://github.com/vkiryukhin/vkBeautify> and <http://www.levmuchnik.net/Content/ProgrammingTips/WEB/XMLDisplay/DisplayXMLFileWithJavascript.html> for more information about 'vkbeautify' and 'XMLDisplay', respectively.

**Copyright**  file inst/COPYRIGHTS

**URL**  https://gitlab.com/hrbrmstr/htmltidy

**BugReports**  https://gitlab.com/hrbrmstr/htmltidy/issues

**Depends**  R (>= 3.2.0)

**License**  MIT + file LICENSE

**LazyData**  true

**NeedsCompilation**  yes

**Suggests**  testthat, httr, rvest

**LinkingTo**  Rcpp

**Imports**  Rcpp, xml2, XML, htmlwidgets, htmltools

**RoxygenNote**  6.1.1
**Description**

Returns a character vector of available style sheets to use when displaying an XML document.

**Usage**

highlight_styles()

**References**

See [https://highlightjs.org/static/demo/](https://highlightjs.org/static/demo/) for a demo of all highlight.js styles

**Examples**

highlight_styles()
**htmltidy**

Tidy Up and Test XPath Queries on HTML and XML Content

---

**Description**

HTML documents can be beautiful and pristine. They can also be wretched, evil, malformed demon-spawn. Now, you can tidy up that HTML and XHTML before processing it with your favorite angle-bracket crunching tools, going beyond the limited tidying that 'libxml2' affords in the 'XML' and 'xml2' packages and taming even the ugliest HTML code generated by the likes of Google Docs and Microsoft Word. It's also possible to use the functions provided to format or "pretty print" HTML content as it is being tidied. Utilities are also included that make it possible to view formatted and "pretty printed" HTML/XML content from HTML/XML document objects, nodes, node sets and plain character HTML/XML using 'vkbeautify' (by Vadim Kiryukhin) and 'highlight.js' (by Ivan Sagalaev). Also (optionally) enables filtering of nodes via XPath or viewing an XML document in "tree" view using 'xml-viewer' (by Julian Gruber). See [https://github.com/vkiryukhin/vkBeautify](https://github.com/vkiryukhin/vkBeautify) and [https://github.com/juliangruber/xml-viewer](https://github.com/juliangruber/xml-viewer) for more information about 'vkbeautify' and 'xml-viewer', respectively.

**Author(s)**

Bob Rudis (bob@rud.is)

---

**renderXmlview**

Widget render function for use in Shiny

---

**Description**

Widget render function for use in Shiny

**Usage**

renderXmlview(expr, env = parent.frame(), quoted = FALSE)

**Arguments**

- **expr**
- **env**
- **quoted**
Tidy or "Pretty Print" HTML/XHTML Documents

Description

Pass in HTML content as either plain or raw text or parsed objects (either with the XML or xml2 packages) or as an httr response object along with an options list that specifies how the content will be tidied and get back tidied content of the same object type as passed in to the function.

Usage

```r
## S3 method for class 'response'
tidy_html(content, options = list(TidyXhtmlOut = TRUE), verbose = FALSE)
## Default S3 method:
tidy_html(content, options = list(TidyXhtmlOut = TRUE), verbose = FALSE)
## S3 method for class 'character'
tidy_html(content, options = list(TidyXhtmlOut = TRUE), verbose = FALSE)
## S3 method for class 'raw'
tidy_html(content, options = list(TidyXhtmlOut = TRUE), verbose = FALSE)
## S3 method for class 'xml_document'
tidy_html(content, options = list(TidyXhtmlOut = TRUE), verbose = FALSE)
## S3 method for class 'HTMLInternalDocument'
tidy_html(content, options = list(TidyXhtmlOut = TRUE), verbose = FALSE)
## S3 method for class 'connection'
tidy_html(content, options = list(TidyXhtmlOut = TRUE), verbose = FALSE)
```

Arguments

- `content` accepts a character vector, raw vector or parsed content from the xml2 or XML packages.
- `options` named list of options
**Details**

The default option TidyXhtmlOut will convert the input content to XHTML.

Currently supported options:

- Ones taking a logical value: TidyAltText, TidyBodyOnly, TidyBreakBeforeBR, TidyCoerceEndTags, TidyDropEmptyElems, TidyDropEmptyParas, TidyFixBackslash, TidyFixComments, TidyGDocClean, TidyHideComments, TidyHtmlOut, TidyIndentContent, TidyJoinClasses, TidyJoinStyles, TidyLogicalEmphasis, TidyMakeBare, TidyMakeClean, TidyMark, TidyOmitOptionalTags, TidyPreTags
- Ones taking a character value: TidyDoctype, TidyInlineTags, TidyBlockTags, TidyEmptyTags, TidyPreTags
- Ones taking an integer value: TidyIndentSpaces, TidyTabSize, TidyWrapLen

File an issue if there are other libtidy options you’d like supported.

It is likely that the most used options will be:

- TidyXhtmlOut (logical),
- TidyHtmlOut (logical) and
- TidyDocType which should be one of "omit", "html5", "auto", "strict" or "loose".

You can clean up Microsoft Word (2000) and Google Docs HTML via logical settings for TidyWord2000 and TidyGDocClean, respectively.

It may also be advantageous to remove all comments with TidyHideComments.

**Value**

Tidied HTML/XHTML content. The object type will be the same as that of the input type except when it is a connection, then a character vector will be returned.

**Note**

If document parsing errors are severe enough, tidy_html() will not be able to clean the document and will display the errors (this output can be captured with sink() or capture.output()) along with a warning and return a "best effort" cleaned version of the document.

**References**

http://api.html-tidy.org/tidy/quickref_5.1.25.html & https://github.com/htacg/tidy-html5/blob/master/include/tidyenum.h for definitions of the options supported above and https://www.w3.org/People/Raggett/tidy/ for an explanation of what "tidy" HTML is and some canonical examples of what it can do.
Examples

```r
opts <- list(
  TidyDocType="html5",
  TidyMakeClean=TRUE,
  TidyHideComments=TRUE,
  TidyIndentContent=TRUE,
  TidyWrapLen=200
)

txt <- paste0(
  c("<html><head><style>p { color: red; }</style><body><!-- ===== body ====== -->",
     "<p>Test</p></body><!--Default Zone --> <!--Default Zone End--></html>"),
  collapse="")
cat(tidy_html(txt, option=opts))

## Not run:
library(httr)
res <- GET("https://rud.is/test/untidy.html")
# look at the original, un-tidy source
cat(content(res, as="text", encoding="UTF-8"))

# see the tidied version
cat(tidy_html(content(res, as="text", encoding="UTF-8"),
  list(TidyDocType="html5", TidyWrapLen=200)))

# but, you could also just do:
cat(tidy_html(url("https://rud.is/test/untidy.html")))

## End(Not run)
```

xmltreeview-shiny  
Shiny bindings for xmltreeview

Description

Output and render functions for using xmltreeview within Shiny applications and interactive Rmd documents.

Usage

```r
xmltreeviewOutput(outputId, width = "100\%", height = "400px")
renderXmltreeview(expr, env = parent.frame(), quoted = FALSE)
```
Arguments

outputId  
output variable to read from

width, height  
Must be a valid CSS unit (like '100%', '400px', 'auto') or a number, which will be coerced to a string and have 'px' appended.

expr  
An expression that generates a xmltreeview

eval  
The environment in which to evaluate 'expr'.

quoted  
Is 'expr' a quoted expression (with 'quote()')? This is useful if you want to save an expression in a variable.

Description

Widget output function for use in Shiny

Usage

xmlviewOutput(outputId, width = "100", height = "400px")

Arguments

outputId  
outputId

width  
width

height  
height

xml_tree_view  
HTML/XML tree viewer

Description

This uses the xml-viewer JavaScript module to provide a simple collapsible tree viewer for HTML/XML documents, nodes, node sets and plain character HTML/XML in an htmlwidget pane.

Usage

xml_tree_view(doc = NULL, mode = c("traditional", "modern"),
scroll = FALSE, width = "100", height = NULL)

html_tree_view(doc = NULL, mode = c("traditional", "modern"),
scroll = FALSE, width = "100", height = NULL)
xml_view

Arguments

- **doc**: XML/HTML internal document or atomic character vector of HTML/XML content
- **mode**: viewer mode. traditional uses tag notation; modern favors readability over angle brackets.
- **scroll**: should the <div> holding the HTML/XML content scroll (TRUE) or take up the full viewer/browser window (FALSE). Default is FALSE (take up the full viewer/browser window). If this is set to TRUE, height should be set to a value other than NULL.
- **width**: widget div width
- **height**: widget div height

Note

Large HTML or XML content may take some time to render properly. It is suggested that this function be used on as minimal of a subset of HTML/XML as possible or used in a browser context vs an IDE viewer context.

References

xonomy xml viewer

Examples

```r
if(interactive()) {
  txt <- paste0("<note><to>Tove</to><from>Jani</from><heading>Reminder</heading>",
               "<body>Don't forget me this weekend!</body></note>"
  # xml_tree_view(txt)
}
```

Description

This uses the vkbeautify and highlight.js javascript modules to format and "pretty print" HTML/XML documents, nodes, node sets and plain character HTML/XML in an htmlwidget pane.

Usage

```r
xml_view(doc, style = "default", scroll = FALSE, add_filter = FALSE, apply_xpath = NULL, width = "100%", height = NULL)

html_view(doc, style = "default", scroll = FALSE, add_filter = FALSE, apply_xpath = NULL, width = "100%", height = NULL)
```
xml_view

Arguments

- style: CSS stylesheet to use (see highlight_styles()).
- scroll: should the <div> holding the HTML/XML content scroll (TRUE) or take up the full viewer/browser window (FALSE). Default is FALSE (take up the full viewer/browser window). If this is set to TRUE, height should be set to a value other than NULL.
- add_filter: show an XPath input box to enable live filtering? (default: FALSE)
- apply_xpath: Add and apply an XPath query string to the view. If add_filter is TRUE then this query string will appear in the filter box and be applied to the passed in document.
- width: widget width (best to keep it at 100%)
- height: widget height (kinda only useful for knitting since this is meant to be an interactive tool).

Note

Large HTML or XML content may take some time to render properly. It is suggested that this function be used on as minimal of a subset of HTML/XML as possible or used in a browser context vs an IDE viewer context.

References

highlight.js, vkbeautify

Examples

```r
if (interactive()) {
  txt <- paste0("<note><to>Tove</to><from>Jani</from><heading>Reminder</heading>",
          "<body>Don’t forget me this weekend!</body></note>"
  )
  # xml_view(txt)
} 
```
Index

highlight_styles, 2
html_tree_view (xml_tree_view), 7
html_view (xml_view), 8
htmltidy, 3
htmltidy-package (htmltidy), 3

renderXmltreeview (xmltreeview-shiny), 6
renderXmlview, 3

tidy_html (tidy_html.response), 4
tidy_html.response, 4

xml_tree_view, 7
xml_view, 8
xmltreeview-shiny, 6
xmltreeviewOutput (xmltreeview-shiny), 6
xmlviewOutput, 7