Package 'brew'

February 19, 2015

Type Package

Title Templating Framework for Report Generation

Version 1.0-6

Date 2010-09-30

Author Jeffrey Horner

Maintainer Jeffrey Horner <jeffrey.horner@gmail.com>

Description brew implements a templating framework for mixing text and R code for report generation. brew template syntax is similar to PHP, Ruby's erb module, Java Server Pages, and Python's psp module.

License GPL-2

Repository CRAN

Date/Publication 2011-04-13 15:16:08

NeedsCompilation no

R topics documented:

`brew` *Report Brewing For Text and R Output*

Description

`brew` provides a templating system for text reporting. The syntax is similar to PHP, Java Server Pages, Ruby's erb module, and Python's psp module.

Usage

```
brew(file=stdin(),output=stdout(),text=NULL,envir=parent.frame(),
     run=TRUE,parseCode=TRUE,tplParser=NULL,chdir=FALSE)
```

Arguments

file	A connection, or a character string naming the file to read from. stdin() is the default.
output	A connection, or a character string naming the file to print to. stdout() is the default.
text	A character string treated as if it contained lines of a file to read from. Only one of file or text is used as input. Default is NULL.
envir	the environment in which the input is to be evaluated. Default is the caller's environment, useful for nested brew calls.
run	Logical to determine if brew should evaluate the input (run=TRUE) or just parse it (run=FALSE). Useful for debugging.
parseCode	Logical. only relevant when run=FALSE. When TRUE the brewed code is parsed and then silently returned. When FALSE, the brewed code is returned as a list. See the Value section for details.
tplParser	a function to parse the text between '<%%' and '%%>' and return the result as a character vector. The template text is passed to the function as a variable length character vector in the first argument position.
chdir	logical; if TRUE and file is a pathname, the R working directory is temporarily changed to the directory containing file for evaluating. brew will also honor the global option brew.chdir.

Details

brew syntax is quite simple and there are very few delimiters to learn:

- 1.All text that falls outside of the delimiters is printed as-is.
- 2.R expressions between the '<%' and '%>' delimiters are executed in-place.
- 3.The value of the R expression between the '<%=' and '%>' delimiters is printed.
- 4.All text between the '<%\#' and '%>' delimiters is thrown away. Use it as a comment.
- 5.If you place a '-' just before the '%>' delimiter, and it's placed at the end of a line, then the newline is omitted from the output.

The following template contains syntax to exercise all brew functionality:

```
---------------
You won't see this R output, but it will run. <% foo <- 'bar' %>
Now foo is <%=foo%> and today is <%=format(Sys.time(),'%B %d, %Y')%>.
<%# Comment -- ignored -- useful in testing.
    Also notice the dash-percent-gt.
    It chops off the trailing newline.
```

```
    You can add it to any percent-gt. -%>
How about generating a template from a template?
<%% foo <- 'fee fi fo fum' %%>
foo is still <%=foo%>.
---------------
```

The output is:

```
--------------
You won't see this R output, but it will run.
Now foo is bar and today is April 20, 2007.
How about generating a template from a template?
<% foo <- 'fee fi fo fum' %>
foo is still bar.
--------------
```

Also, for power users, there's one more thing:

- 6.Text between the '<%%' and '%%>' delimiters is treated as a brew template and is printed as-is, but the delimiters are changed to '<%' and '%>'. This happens when tplParser=NULL. But if tplParser is a valid function, then the text is passed to tplParser which should return a character vector to replace the text.

NOTE: brew calls can be nested and rely on placing a function named '.brew.cat' in the environment in which it is passed. Each time brew is called, a check for the existence of this function is made. If it exists, then it is replaced with a new copy that is lexically scoped to the current brew frame. Once the brew call is done, the function is replaced with the previous function. The function is finally removed from the environment once all brew calls return.

Value

When `run=TRUE`, the value of the last expression after brewing the input or an object of class 'try-error' containing the error message if brewing failed.

When `run=FALSE` and `parseCode=TRUE`, a function whose environment contains the text vector and the code vector of the parsed expressions after brewing the input. It takes brew's output and envir arguments.

When `run=FALSE` and `parseCode=FALSE`, a list containing the text vector and the unparsed code vector.

Author(s)

Jeffrey Horner <jeff.horner@vanderbilt.edu>

See Also

`Sweave` for the original report generator.

Examples

```
## A port of the Sweave test file.
brew(system.file("brew-test-1.brew",package="brew"),"brew-test-1.tex",envir=new.env())

## Everything you wanted to know about your R session.
brew(system.file("brew-test-2.brew",package="brew"),"brew-test-2.html",envir=new.env())
browseURL(paste('file://',file.path(getwd(),'brew-test-2.html'),sep=''))

## Don't sully up environment, so use envir=new.env(). Nested brew calls will still work.
brew(system.file("example1.brew",package="brew"),envir=new.env())

## Various ways to print R output
library(datasets)
brew(system.file("catprint.brew",package="brew"),envir=new.env())
rm(iris)

## The example from the Details section
brew(system.file("featurefull.brew",package="brew"),envir=new.env())

## Using the tplParser argument
tParse <- function(text) paste('Got this: <',text,'>\n',sep='',collapse='')
brew(system.file("featurefull.brew",package="brew"),envir=new.env(),tplParser=tParse)
rm(tParse)
```

`brewCache` *Caching Brew Templates*

Description

These functions provide a cache system for brew templates.

Usage

```
brewCache(envir=NULL)

brewCacheOn()
brewCacheOff()

setBufLen(len=0)
```

Arguments

`envir`	the `environment` to store text and R expressions for each brewed template.
`len`	length of internal buffers for parsing the templates.

Details

brew can cache parsed templates for potential speedup but only when brew calls are passed filenames, not connections, and when tplParser is NULL.

brew caching is implemented by storing file names, modification times, and the associated text and R expressions in an internal environment. Calling `brewCache()` with an appropriate environment sets the internal cache. Calling without arguments returns the internal cache. The cache is exposed this way mainly for testing, debugging, performance improvement, etc. and this may be off-limits in future releases.

Simple enough, `brewCacheOn()` turns on caching of brew templates and is equivalent to calling `brewCache(envir=new.env(hash=TRUE,parent=globalenv()))`. `brewCache()` without arguments returns the internal environment. Calling `brewCacheOff()` turns off caching by setting the internal environment to `NULL`.

One thing to note: filenames act as keys in the internal cache environment, and brew does nothing to expand them to their full paths. Thus, '/home/user/brew.html' and '~usr/brew.html' will act as separate keys, although on-disk they may actually point to the same file.

`setBufLen()` initializes internal parsing vectors to length `len`. Default is 0.

Value

`brewCache()` without arguments returns the internal cache. All others invisibly return `NULL`.

Author(s)

Jeffrey Horner <jeff.horner@vanderbilt.edu>

See Also

`Sweave` for the original report generator.

Examples

```
## Turn on caching
brewCacheOn()

## Brew a template
brew(system.file("featurefull.brew",package="brew"),envir=new.env())

## Get the internal cache
cache <- brewCache()

## Inspect
as.list(cache)

## Inspect first file cached in list
as.list(cache)[[1]]

## Inspect environment that contains text and parsed code
as.list(as.list(cache)[[1]]$env)
```

```
## Turn off brew caching
brewCacheOff()
rm(cache)
```

Index