Package 'debugme'

October 22, 2017

Title Debug R Packages

Version 1.1.0

Author Gábor Csárdi

Maintainer Gábor Csárdi <csardi.gabor@gmail.com>

Description Specify debug messages as special string constants, and control debugging of packages via environment variables.

License MIT + file LICENSE

LazyData true

URL https://github.com/r-lib/debugme#readme

BugReports https://github.com/r-lib/debugme/issues

RoxygenNote 6.0.1

Imports crayon, grDevices

Suggests covr, mockery, R6, testthat, withr

Encoding UTF-8

NeedsCompilation no

Repository CRAN

Date/Publication 2017-10-22 14:18:09 UTC

R topics documented:

debug *Debug message*

Description

Normally this function is *not* called directly, but debug strings are used. See `debugme()`.

Usage

```
debug(msg, pkg = environmentName(topenv(parent.frame())))
```

Arguments

`msg`	Message to print, character constant.
`pkg`	Package name to which the message belongs. Detected automatically.

Value

The original message.

debugme *Debug R Packages*

Description

Specify debug messages as special string constants, and control debugging of packages via environment variables.

Usage

```
debugme(env = topenv(parent.frame()), pkg = environmentName(env))
```

Arguments

`env`	Environment to instument debugging in. Defaults to the package environment of the calling package.
`pkg`	Name of the calling package. The default should be fine for almost all cases.

Details

To add debugging to your package, you need to

1. Import the debugme package.
2. Define an .onLoad function in your package, that calls debugme. An example:

```
.onLoad <- function(libname, pkgname) { debugme::debugme() }
```

By default debugging is off. To turn on debugging, set the DEBUGME environment variable to the names of the packages you want to debug. Package names can be separated by commas.

Note that debugme checks for environment variables when it is starting up. Environment variables set after the package is loaded do not have any effect.

Example debugme entries:

```
"!DEBUG Start Shiny app"
```

Dynamic debug messsages

It is often desired that the debug messages contain values of R expressions evaluated at runtime. For example, when starting a Shiny app, it is useful to also print out the path to the app. Similarly, when debugging an HTTP response, it is desired to log the HTTP status code.

debugme allows embedding R code into the debug messages, within backticks. The code will be evaluated at runtime. Here are some examples:

```
"!DEBUG Start Shiny app at `path`"
"!DEBUG Got HTTP response `httr::status_code(reponse)`"
```

Note that parsing the debug strings for code is not very sophisticated currently, and you cannot embed backticks into the code itself.

Log levels

To organize the log messages into log levels, you can start the !DEBUG token with multiple ! characters. You can then select the desired level of logging via ! characters before the package name in the DEBUGME environment variable. E.g. DEBUGME=!!mypackage means that only debug messages with two or less ! marks will be printed.

Debug stack

By default debugme prints the *debug stack* at the beginning of the debug messages. The debug stack contains the functions in the call stack that have (and emit) debug messages. To suppress printing the call stack set the DEBUGME_SHOW_STACK environment variable to no.

Redirecting the output

If the DEBUGME_OUTPUT_FILE environment variable is set to a filename, then the output is written there instead of the standard output stream of the R process.

If DEBUGME_OUTPUT_FILE is not set, but DEBUGME_OUTPUT_DIR is, then a log file is created there, and the name of the file will contain the process id. This is is useful for logging from several parallel R processes.

Index