Package 'debugme'

April 25, 2024

Title Debug R Packages

Version 1.2.0

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

License MIT + file LICENSE

URL <https://github.com/r-lib/debugme#readme>,
<https://r-lib.github.io/debugme/>

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

Depends R (>= 3.6)

Imports crayon, grDevices

Suggests covr, mockery, R6, testthat (>= 3.0.0), withr

Config/Needs/website tidyverse/tidytemplate

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.2.3

NeedsCompilation no

Author Gábor Csárdi [aut, cre],
Posit Software, PBC [cph, fnd]

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

Repository CRAN

Date/Publication 2024-04-25 07:50:02 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())), level = 2)
```

Arguments

msg	Message to print, character constant.
pkg	Package name to which the message belongs. Detected automatically.
level	The maximum debug level to show this message at.

Value

Invisibly, the message if it is shown, otherwise NULL.

---

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.

This environment variable is read when the package is loaded, and with every call to `debugme()`.

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

debugme has two ways to organize log messages into log levels: a quick informal way, and a more formal one.

For the informal way, 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.

A more formal way is to use log level names: `"FATAL"`, `"ERROR"`, `"WARNING"`, `"INFO"`, `"DEBUG"`, and `"VERBOSE"`. To specify the log level of the message, append the log level to `"!DEBUG"`, with a dash. E.g.:

```
"!DEBUG-INFO Just letting you know that..."
```

To select the log level of a package, you can specify the level either with the number of `!` characters, as above, or adding the log level as a suffix to the package name, separated by a dash. E.g.:

```
Sys.setenv(DEBUGME = "mypackage-INFO")
```

(Use either methods to set the log level, but do not mix them.)

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