

Package ‘profile’

August 22, 2022

Title Read, Manipulate, and Write Profiler Data

Version 1.0.3

Date 2022-08-21

Description Defines a data structure for profiler data, and methods to read and write from the ‘Rprof’ and ‘pprof’ file formats.

License MIT + file LICENSE

URL <https://r-prof.github.io/profile/>,
<https://github.com/r-prof/profile>

BugReports <https://github.com/r-prof/profile/issues>

Imports methods, rlang, tibble (>= 3.0.0), vctrs, withr

Suggests covr, dm, DiagrammeR, DiagrammeRsvg, RProtoBuf, testthat

Encoding UTF-8

RoxygenNote 7.2.1

Collate ‘api.R’ ‘compat-purrr.R’ ‘pprof-from-ds.R’ ‘rprof-read.R’
‘pprof-read.R’ ‘pprof-to-ds.R’ ‘rprof-write.R’ ‘pprof-write.R’
‘profile-package.R’ ‘proto.R’ ‘rprof-from-ds.R’ ‘rprof-to-ds.R’
‘utils.R’

NeedsCompilation no

Author Kirill Müller [aut, cre] (<<https://orcid.org/0000-0002-1416-3412>>),
R Consortium [fnd]

Maintainer Kirill Müller <kirill@cynkra.com>

Repository CRAN

Date/Publication 2022-08-22 09:00:13 UTC

R topics documented:

profile-package	2
read_rprof	2
validate_profile	3

Index	6
--------------	----------

profile-package

profile: Read, Manipulate, and Write Profiler Data

Description

The profiler package defines a stable data format for profiler data, see `validate_profile()` for a definition. It supports reading and writing `Rprof` .out and `pprof` .proto files.

Author(s)

Maintainer: Kirill Müller <kirill@cynkra.com> ([ORCID](#))

Other contributors:

- R Consortium [funder]

See Also

Useful links:

- <https://r-prof.github.io/profile/>
- <https://github.com/r-prof/profile>
- Report bugs at <https://github.com/r-prof/profile/issues>

read_rprof

File I/O for profiler data

Description

These functions read profile data from files, or store profile data to files. Readers call `validate_profile()` on input, writers on output.

`read_rprof()` reads a file generated by `Rprof()`, `write_rprof()` writes in a compatible format.

`read_pprof()` reads a file generated by `pprof -proto`, `write_pprof()` writes a Gzip-compressed file that can be processed with `pprof`.

Usage

```
read_rprof(path, ..., version = "1.0")
```

```
read_pprof(path, ..., version = "1.0")
```

```
write_rprof(x, path)
```

```
write_pprof(x, path)
```

Arguments

path	File name
...	Ignored
version	Version of the data, currently only "1.0" is supported. Pass an explicit value to this argument if your code depends on the data format.
x	Profiler data, see validate_profile()

Details

Use the **proftools**, **profvis**, or **prof.tree** R packages to further analyze files created by the `write_rprof()` function.

Use the **pprof tool** in conjunction with the `_pprof()` functions. The tool is available in the **pprof** R package, or (in newer versions) via `go get github.com/google/pprof`.

Value

Valid profile data (readers), input data (writers).

Examples

```
rprof_file <- system.file("samples/rprof/1.out", package = "profile")
ds <- read_rprof(rprof_file)
ds
if (requireNamespace("RProtoBuf", quietly = TRUE)) {
  pprof_file <- tempfile("profile", fileext = ".pb.gz")
  write_pprof(ds, pprof_file)
}
```

validate_profile *Definition of the profile data format*

Description

The data format is stable between major releases. In case of major updates, compatibility functions will be provided.

The `validate_profile()` function checks a profile data object for compatibility with the specification. Versioning information embedded in the data is considered.

The `dm_from_profile()` function converts a profile to a dm object. The **dm** package must be installed. See `dm::dm()` for more information.

Usage

```
validate_profile(x)
```

```
dm_from_profile(x)
```

Arguments

x Profile data, e.g., as returned by `read_pprof()` or `read_rprof()`.

Details

The profile data is stored in an object of class "profile_data", which is a named list of [tibbles](#). This named list has the following components, subsequently referred to as *tables*:

- meta
- sample_types
- samples
- locations
- functions (Components with names starting with a dot are permitted after the required components, but will be ignored.)

The meta table has two character columns, key and value. Additional columns with a leading dot in the name are allowed after the required columns. It is currently restricted to one row with key "version" and a value that is accepted by `package_version()`.

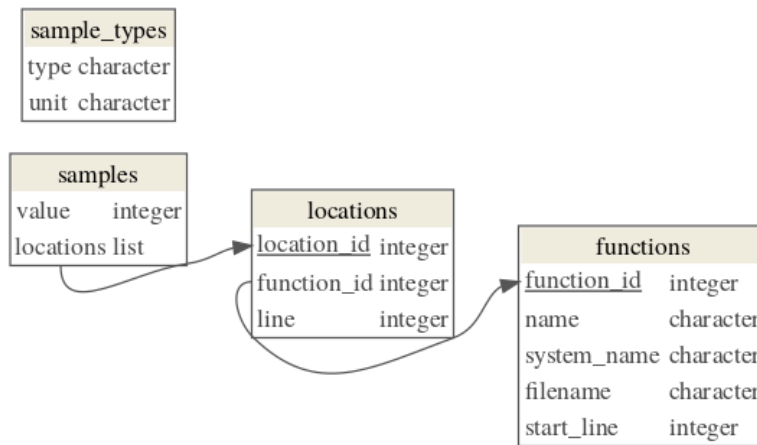
The sample_types table has two character columns, type and unit. Additional columns with a leading dot in the name are allowed after the required columns. It is currently restricted to one row with values "samples" and "count", respectively.

The samples table has two columns, value (integer) and locations (list). Additional columns with a leading dot in the name are allowed after the required columns. The value column describes the number of consecutive samples for the given location, and must be greater than zero. Each element of the locations column is a tibble with one integer column, location_id. For each location_id value a corresponding observation in the locations table must exist. The locations are listed in inner-first order, i.e., the first location corresponds to the innermost entry of the stack trace.

The locations table has three integer columns, location_id, function_id, and line. Additional columns with a leading dot in the name are allowed after the required columns. All location_id values are unique. For each function_id value a corresponding observation in the functions table must exist. NA values are permitted. The line column describes the line in the source code this location corresponds to, zero if unknown. All values must be nonnegative. NA values are permitted.

The functions table has five columns, function_id (integer), name, system_name and file_name (character), and start_line (integer). Additional columns with a leading dot in the name are allowed after the required columns. All function_id values are unique. The name, system_name and filename columns describe function names (demangled and mangled), and source file names for a function. Both name and system_name must not contain empty strings. The start_line column describes the start line of a function in its source file, zero if unknown. All values must be nonnegative.

Data model



Examples

```

rprof_file <- system.file("samples/rprof/1.out", package = "profile")
ds <- read_rprof(rprof_file)
validate_profile(ds)

bad_ds <- ds
bad_ds$samples <- NULL
try(validate_profile(bad_ds))

if (rlang::is_installed("dm")) {
  dm <- dm_from_profile(ds)
  print(dm)
  dm::dm_draw(dm)
}
    
```

Index

`dm::dm()`, 3
`dm_from_profile(validate_profile)`, 3

`package_version()`, 4
`profile(profile-package)`, 2
`profile-package`, 2

`read_pprof(read_rprof)`, 2
`read_pprof()`, 4
`read_rprof`, 2
`read_rprof()`, 4
`Rprof`, 2
`Rprof()`, 2

`tibble`, 4

`validate_profile`, 3
`validate_profile()`, 2, 3

`write_pprof(read_rprof)`, 2
`write_rprof(read_rprof)`, 2