

# Package ‘W4MUtils’

October 30, 2025

**Title** Useful Functions for Harmonized W4M Tool Development

**Version** 1.2.1

**Description** Provides a set of utility function to prevent the spread of utility scripts in W4M (Workflow4Metabolomics) tools, and centralize them in a single package.

To note, some are meant to be replaced by the use of dedicated packages in the future, like the parse\_args() function: it is here only to prepare the ground for more global changes in W4M scripts and tools.

This package is used by part of the W4M Galaxy modules, some of them being available on the community-maintained GitHub repository for Metabolomics' Galaxy tools <<https://github.com/workflow4metabolomics/tools-metabolomics>>.

See Delporte et al (2025) <[doi:10.1100/cpz1.70095](https://doi.org/10.1100/cpz1.70095)> for more details.

**License** AGPL (>= 3)

**Encoding** UTF-8

**RoxygenNote** 7.3.2

**Imports** methods, rlang, stats, graphics, grDevices

**Suggests** covr, DT, knitr, optparse, pkgdown, rmarkdown, testthat (>= 3.0.0)

**Config/testthat.edition** 3

**VignetteBuilder** knitr

**Collate** datafram\_helpers.R miniTools.R RcheckLibrary.R  
optparse\_helper.R galaxy.R logging.R utils.R  
Filtering\_datasets.R intensity\_checks.R

**NeedsCompilation** no

**Author** Lain Pavot [aut],  
Melanie Petera [aut, cre],  
Helge Hecht [ctb]

**Maintainer** Melanie Petera <[melanie.petera@inrae.fr](mailto:melanie.petera@inrae.fr)>

**Repository** CRAN

**Date/Publication** 2025-10-30 20:20:21 UTC

## Contents

check_err . . . . .	3
check_one_character . . . . .	4
check_one_complex . . . . .	4
check_one_integer . . . . .	5
check_one_logical . . . . .	5
check_one_numeric . . . . .	6
check_parameter_length . . . . .	7
check_parameter_type . . . . .	10
check_param_type_n_length . . . . .	13
collapse . . . . .	16
collapse_lines . . . . .	17
convert_parameters . . . . .	17
df_force_numeric . . . . .	18
df_is . . . . .	19
df_read_table . . . . .	19
Filtering_datasets . . . . .	20
generic_filter . . . . .	20
get_base_dir . . . . .	22
get_logger . . . . .	22
get_r_env . . . . .	23
import2 . . . . .	23
import3 . . . . .	25
intensity_checks . . . . .	26
intens_check . . . . .	26
in_galaxy_env . . . . .	28
match2 . . . . .	28
match3 . . . . .	29
metab_merge . . . . .	29
mini_tools . . . . .	30
optparse_character . . . . .	30
optparse_flag . . . . .	31
optparse_integer . . . . .	32
optparse_list . . . . .	33
optparse_numeric . . . . .	34
optparse_parameters . . . . .	35
parse_args . . . . .	36
printf . . . . .	38
printfp . . . . .	38
printp . . . . .	39
reproduce_id . . . . .	40
run_galaxy_function . . . . .	41
run_galaxy_processing . . . . .	42
show_galaxy_footer . . . . .	45
show_galaxy_header . . . . .	46
show_sys . . . . .	47
shy_lib . . . . .	48

<i>check_err</i>	3
source_local . . . . .	48
stock_id . . . . .	49
stopaste . . . . .	50
stopaste0 . . . . .	51
stopf . . . . .	52
unmangle_galaxy_param . . . . .	53
unmangle_galaxy_string . . . . .	53
W4MLLogger . . . . .	54
W4MLLogger_.message__ . . . . .	55
W4MLLogger_add_out_paths . . . . .	56
W4MLLogger_finalize . . . . .	56
W4MLLogger_set_debug . . . . .	56
W4MLLogger_set_error . . . . .	57
W4MLLogger_set_info . . . . .	57
W4MLLogger_set_out_paths . . . . .	57
W4MLLogger_set_verbose . . . . .	58
W4MLLogger_set_warning . . . . .	58
W4MLLogger_[info,warning,error,debug,verbose] . . . . .	59
<b>Index</b>	<b>60</b>

---

check_err	<i>Check Errors</i>
-----------	---------------------

---

## Description

`check_err` Generic function that stops in error if problems have been encountered, with printing of the corresponding error information.

## Usage

```
check_err(err_stock)
```

## Arguments

<code>err_stock</code>	vector of results returned by check functions
------------------------	---

## Value

`NULL` (function called for the "stoping" side effects)

## Author(s)

M.Petera

---

check\_one\_character      *check\_one\_character*

---

### Description

Shortcut function to check\_param\_type\_n\_length(), to check a unique character string.

### Arguments

value	The parameter to test.
...	Parameters passed out to the check_param_type_n_length() function

### Value

NULL

### Author(s)

L.Pavot

### See Also

[check\\_param\\_type\\_n\\_length](#)  
[check\\_parameter\\_type,check\\_parameter\\_length](#)

---

check\_one\_complex      *check\_one\_complex*

---

### Description

Shortcut function to check\_param\_type\_n\_length(), to check a unique complex value.

### Arguments

value	The parameter to test.
...	Parameters passed out to the check_param_type_n_length() function

### Value

NULL

### Author(s)

L.Pavot

**See Also**

[check\\_param\\_type\\_n\\_length](#)  
[check\\_parameter\\_type,check\\_parameter\\_length](#)

---

*check\_one\_integer*      *check\_one\_integer*

---

**Description**

Shortcut function to `check_param_type_n_length()`, to check a unique integer value.

**Arguments**

`value`      The parameter to test.  
...      Parameters passed out to the `check_param_type_n_length()` function

**Value**

NULL

**Author(s)**

L.Pavot

**See Also**

[check\\_param\\_type\\_n\\_length](#)  
[check\\_parameter\\_type,check\\_parameter\\_length](#)

---

*check\_one\_logical*      *check\_one\_logical*

---

**Description**

Shortcut function to `check_param_type_n_length()`, to check a unique boolean.

**Arguments**

`value`      The parameter to test.  
...      Parameters passed out to the `check_param_type_n_length()` function

**Value**

NULL

**Author(s)**

L.Pavot

**See Also**

[check\\_param\\_type\\_n\\_length](#)  
[check\\_parameter\\_type,check\\_parameter\\_length](#)

---

[check\\_one\\_numeric](#)      *check\_one\_numeric*

---

**Description**

Shortcut function to `check_param_type_n_length()`, to check a unique numeric value.

**Arguments**

<code>value</code>	The parameter to test.
<code>...</code>	Parameters passed out to the <code>check_param_type_n_length()</code> function

**Value**

`NULL`

**Author(s)**

L.Pavot

**See Also**

[check\\_param\\_type\\_n\\_length](#)  
[check\\_parameter\\_type,check\\_parameter\\_length](#)

---

```
check_parameter_length
```

*check\_parameter\_length - validate parameter's length*

---

## Description

Use this function to validate parameters. You're never assured that provided parameters from users are the right type, or length. This may be the case with your own code as well, if you have undetected bugs in your code.

This function helps prevent unpredictable behaviour coming from bad parameters.

It checks the size of vectors, and the type of values. If the parameter is not the good type or length, the program stops with an explanatory error.

## Usage

```
check_parameter_length(  
  value,  
  expected_size,  
  nth = NULL,  
  func_name = NULL,  
  param_name = NULL,  
  or_more = FALSE,  
  nframe = 1  
)
```

## Arguments

<code>value</code>	The parameter to test.
<code>expected_size</code>	The expected size of the vector. Usually, 1.
<code>nth</code>	This parameter is used in the error message generation. Provide a character vector like "first", "second", "1st", "2nd", ... this must be the number of the parameter if the function.
<code>func_name</code>	By default, the function name is guessed from the stack. But if you want to change it, or if it is not the right function name in error messages, set the right one here.
<code>param_name</code>	Like <code>func_name</code> , by default the param name is guessed. But if you want to change it, or if it is not the right parameter name in error messages, set the right one here.
<code>or_more</code>	When we check the parameter's length, if <code>or_more</code> is TRUE and the value is bigger than <code>expected_size</code> , then, the length check does not occur
<code>nframe</code>	The number of function calls between this function and the function where the value to test is a parameter. for example, if a user calls function A, which calls <code>check_param_*</code> directly, then <code>nframe</code> must be 1 because it is a direct call. But, if the user has called function A, and function A calls function B, and

`check_param_`, is called in function B, then, for `check_param_` to understand it is a parameter coming from function A (and not from function B), we have to tell `check_param_*` that nframe is 2. If the function name is not the right name, it may be because of that. So don't fear testing different values for nframes.

### Value

NULL

### Author(s)

L.Pavot

### See Also

[check\\_parameter\\_type](#),[check\\_parameter\\_length](#)  
[check\\_one\\_integer](#),[check\\_one\\_logical](#),[check\\_one\\_numeric](#)  
[check\\_one\\_complex](#),[check\\_one\\_character](#)

### Examples

```
## here is a simple utility function we will use in this example.
## It is not important
show_last_error <- function(error) {
  dump.frames()
  message(base:::attr(last.dump, "error.message"))
}

## The example really starts here
## we have a simple function like this:
custom_message <- function(text) {
  message(sprintf("Message: %s", text))
}

## this function needs to have a character vector as first
## parameter.
## So, to validate the parameter, we could write:
custom_message <- function(text) {
  check_parameter_type(text, "character")
  message(base::sprintf("Message: %s", text))
}
tryCatch(custom_message(42), error = show_last_error)

## this function needs to have a vector of length 1.
## So, to validate the parameter, we could write:
custom_message <- function(text) {
  check_parameter_type(text, "character")
  check_parameter_length(text, 1)
  message(base::sprintf("Message: %s", text))
}
```

```
tryCatch(custom_message(c("uwu", "owo")), error = show_last_error)

## Or, to be more concise:
custom_message <- function(text) {
  check_param_type_n_length(text, "character", 1)
  message(base::sprintf("Message: %s", text))
}
tryCatch(custom_message(c("uwu", "owo")), error = show_last_error)
tryCatch(custom_message(42), error = show_last_error)

## Let's say the text can be 1 or more elements, and can be null.
custom_message <- function(text) {
  check_param_type_n_length(
    text,
    expected_type = "character",
    or_null = TRUE,
    expected_size = 1,
    or_more = TRUE
  )
  message(paste0(base::sprintf("Message: %s", text), collapse = "\n"))
}
tryCatch(custom_message(c(42, 43)), error = show_last_error)
tryCatch(custom_message(NULL), error = show_last_error)
## no error, because or_null is TRUE
tryCatch(custom_message(character(0)), error = show_last_error)
tryCatch(custom_message(c("uwu", ":3")), error = show_last_error)
## no error, because or_more is TRUE

## With a function that has a lot of parameters, it may be usefull to
## provide the parameter's number. And, because it becomes very long
## to test all those parameters, we will use shortcuts functions
write_msg <- function(
  text,
  font = "owo",
  font_size = 16,
  italic = FALSE,
  bold = FALSE
) {
  check_one_character(text, nth = "1st")
  check_one_character(font, nth = "2nd")
  check_one_numeric(font_size, nth = "3rd")
  check_one_logical(italic, nth = "before last")
  check_one_logical(bold, nth = "last")
  message(paste0(base::sprintf("Message: %s", text), collapse = "\n"))
}
tryCatch(write_msg(text = 42, "font", 16), error = show_last_error)
tryCatch(write_msg("uwu", font = 1, 16), error = show_last_error)
tryCatch(write_msg("uwu", font_size = "16"), error = show_last_error)
tryCatch(write_msg("uwu", italic = "FALSE"), error = show_last_error)
tryCatch(write_msg("uwu", bold = "FALSE"), error = show_last_error)
```

---

check\_parameter\_type    *check\_parameter\_type - validate parameter's type*

---

## Description

Use this function to validate parameters. You're never assured that provided parameters from users are the right type, or length. This may be the case with your own code as well, if you have undetected bugs in your code.

This function helps prevent unpredictable behaviour coming from bad parameters.

It checks the size of vectors, and the type of values. If the parameter is not the good type or length, the program stops with an explanatory error.

## Usage

```
check_parameter_type(
  value,
  expected_type,
  nth = NULL,
  func_name = NULL,
  param_name = NULL,
  or_null = FALSE,
  nframe = 1
)
```

## Arguments

<code>value</code>	The parameter to test.
<code>expected_type</code>	The character vector of the kind: "character", "integer", "logical", ...
<code>nth</code>	This parameter is used in the error message generation. Provide a character vector like "first", "second", "1st", "2nd", ... this must be the number of the parameter if the function.
<code>func_name</code>	By default, the function name is guessed from the stack. But if you want to change it, or if it is not the right function name in error messages, set the right one here.
<code>param_name</code>	Like <code>func_name</code> , by default the param name is guessed. But if you want to change it, or if it is not the right parameter name in error messages, set the right one here.
<code>or_null</code>	When we check the parameter's type, if <code>or_null</code> is TRUE and the value is NULL, then, the type check does not occur
<code>nframe</code>	The number of function calls between this function and the function where the value to test is a parameter. for example, if a user calls function A, which calls <code>check_param_*</code> directly, then <code>nframe</code> must be 1 because it is a direct call. But, if the user has called function A, and function A calls function B, and <code>check_param_</code> , is called in function B, then, for <code>check_param_</code> to understand it

is a parameter comming from function A (and not from function B), we have to tell `check_param_*` that `nframe` is 2. If the function name is not the right name, it may be because of that. So don't fear testing different values for `nframes`.

### Value

NULL

### Author(s)

L.Pavot

### See Also

[check\\_parameter\\_type](#),[check\\_parameter\\_length](#)  
[check\\_one\\_integer](#),[check\\_one\\_logical](#),[check\\_one\\_numeric](#)  
[check\\_one\\_complex](#),[check\\_one\\_character](#)

### Examples

```
## here is a simple utility function we will use in this example.
## It is not important
show_last_error <- function(error) {
  dump.frames()
  message(base:::attr(last.dump, "error.message"))
}

## The example really starts here
## we have a simple function like this:
custom_message <- function(text) {
  message(sprintf("Message: %s", text))
}

## this function needs to have a character vector as first
## parameter.
## So, to validate the parameter, we could write:
custom_message <- function(text) {
  check_parameter_type(text, "character")
  message(base::sprintf("Message: %s", text))
}
tryCatch(custom_message(42), error = show_last_error)

## this function needs to have a vector of length 1.
## So, to validate the parameter, we could write:
custom_message <- function(text) {
  check_parameter_type(text, "character")
  check_parameter_length(text, 1)
  message(base::sprintf("Message: %s", text))
}
tryCatch(custom_message(c("uwu", "owo")), error = show_last_error)
```

```

## Or, to be more concise:
custom_message <- function(text) {
  check_param_type_n_length(text, "character", 1)
  message(base::sprintf("Message: %s", text))
}
tryCatch(custom_message(c("uwu", "owo")), error = show_last_error)
tryCatch(custom_message(42), error = show_last_error)

## Let's say the text can be 1 or more elements, and can be null.
custom_message <- function(text) {
  check_param_type_n_length(
    text,
    expected_type = "character",
    or_null = TRUE,
    expected_size = 1,
    or_more = TRUE
  )
  message(paste0(base::sprintf("Message: %s", text), collapse = "\n"))
}
tryCatch(custom_message(c(42, 43)), error = show_last_error)
tryCatch(custom_message(NULL), error = show_last_error)
## no error, because or_null is TRUE
tryCatch(custom_message(character(0)), error = show_last_error)
tryCatch(custom_message(c("uwu", ":3")), error = show_last_error)
## no error, because or_more is TRUE

## With a function that has a lot of parameters, it may be usefull to
## provide the parameter's number. And, because it becomes very long
## to test all those parameters, we will use shortcuts functions
write_msg <- function(
  text,
  font = "owo",
  font_size = 16,
  italic = FALSE,
  bold = FALSE
) {
  check_one_character(text, nth = "1st")
  check_one_character(font, nth = "2nd")
  check_one_numeric(font_size, nth = "3rd")
  check_one_logical(italic, nth = "before last")
  check_one_logical(bold, nth = "last")
  message(paste0(base::sprintf("Message: %s", text), collapse = "\n"))
}
tryCatch(write_msg(text = 42, "font", 16), error = show_last_error)
tryCatch(write_msg("uwu", font = 1, 16), error = show_last_error)
tryCatch(write_msg("uwu", font_size = "16"), error = show_last_error)
tryCatch(write_msg("uwu", italic = "FALSE"), error = show_last_error)
tryCatch(write_msg("uwu", bold = "FALSE"), error = show_last_error)

```

---

check\_param\_type\_n\_length

*check\_param\_type\_n\_length - to check parameters*

---

## Description

Use this function to validate parameters. You're never assured that provided parameters from users are the right type, or length. This may be the case with your own code as well, if you have undetected bugs in your code.

This function helps prevent unpredictable behaviour coming from bad parameters.

It checks the size of vectors, and the type of values. If the parameter is not the good type or length, the program stops with an explanatory error.

## Usage

```
check_param_type_n_length(
  value,
  expected_type,
  expected_size = 1,
  nth = NULL,
  func_name = NULL,
  param_name = NULL,
  or_more = FALSE,
  or_null = FALSE,
  nframe = 1
)
```

## Arguments

value	The parameter to test.
expected_type	The character vector of the kind: "character", "integer", "logical", ...
expected_size	The expected size of the vector. Usually, 1.
nth	This parameter is used in the error message generation. Provide a character vector like "first", "second", "1st", "2nd", ... this must be the number of the parameter if the function.
func_name	By default, the function name is guessed from the stack. But if you want to change it, or if it is not the right function name in error messages, set the right one here.
param_name	Like func_name, by default the param name is guessed. But if you want to change it, or if it is not the right parameter name in error messages, set the right one here.
or_more	When we check the parameter's length, if or_more is TRUE and the value is bigger than expected_size, then, the length check does not occur

<code>or_null</code>	When we check the parameter's type, if <code>or_null</code> is TRUE and the value is NULL, then, the type check does not occur
<code>nframe</code>	The number of function calls between this function and the function where the value to test is a parameter. for example, if a user calls function A, which calls <code>check_param_*</code> directly, then <code>nframe</code> must be 1 because it is a direct call. But, if the user has called function A, and function A calls function B, and <code>check_param_</code> , is called in function B, then, for <code>check_param_</code> to understand it is a parameter comming from function A (and not from function B), we have to tell <code>check_param_*</code> that <code>nframe</code> is 2. If the function name is not the right name, it may be because of that. So don't fear testing different values for <code>nframes</code> .

**Value**

`NULL`

**Author(s)**

L.Pavot

**See Also**

[check\\_parameter\\_type](#),[check\\_parameter\\_length](#)  
[check\\_one\\_integer](#),[check\\_one\\_logical](#),[check\\_one\\_numeric](#)  
[check\\_one\\_complex](#),[check\\_one\\_character](#)

**Examples**

```
## here is a simple utility function we will use in this example.
## It is not important
show_last_error <- function(error) {
  dump.frames()
  message(base:::attr(last.dump, "error.message"))
}

## The example really starts here
## we have a simple function like this:
custom_message <- function(text) {
  message(sprintf("Message: %s", text))
}

## this function needs to have a character vector as first
## parameter.
## So, to validate the parameter, we could write:
custom_message <- function(text) {
  check_parameter_type(text, "character")
  message(base:::sprintf("Message: %s", text))
}
tryCatch(custom_message(42), error = show_last_error)
```

```

## this function needs to have a vector of length 1.
## So, to validate the parameter, we could write:
custom_message <- function(text) {
  check_parameter_type(text, "character")
  check_parameter_length(text, 1)
  message(base::sprintf("Message: %s", text))
}
tryCatch(custom_message(c("uwu", "owo")), error = show_last_error)

## Or, to be more concise:
custom_message <- function(text) {
  check_param_type_n_length(text, "character", 1)
  message(base::sprintf("Message: %s", text))
}
tryCatch(custom_message(c("uwu", "owo")), error = show_last_error)
tryCatch(custom_message(42), error = show_last_error)

## Let's say the text can be 1 or more elements, and can be null.
custom_message <- function(text) {
  check_param_type_n_length(
    text,
    expected_type = "character",
    or_null = TRUE,
    expected_size = 1,
    or_more = TRUE
  )
  message(paste0(base::sprintf("Message: %s", text), collapse = "\n"))
}
tryCatch(custom_message(c(42, 43)), error = show_last_error)
tryCatch(custom_message(NULL), error = show_last_error)
## no error, because or_null is TRUE
tryCatch(custom_message(character(0)), error = show_last_error)
tryCatch(custom_message(c("uwu", ":3")), error = show_last_error)
## no error, because or_more is TRUE

## With a function that has a lot of parameters, it may be usefull to
## provide the parameter's number. And, because it becomes very long
## to test all those parameters, we will use shortcuts functions
write_msg <- function(
  text,
  font = "owo",
  font_size = 16,
  italic = FALSE,
  bold = FALSE
) {
  check_one_character(text, nth = "1st")
  check_one_character(font, nth = "2nd")
  check_one_numeric(font_size, nth = "3rd")
  check_one_logical(italic, nth = "before last")
  check_one_logical(bold, nth = "last")
  message(paste0(base::sprintf("Message: %s", text), collapse = "\n"))
}

```

```

}
tryCatch(write_msg(text = 42, "font", 16), error = show_last_error)
tryCatch(write_msg("uwu", font = 1, 16), error = show_last_error)
tryCatch(write_msg("uwu", font_size = "16"), error = show_last_error)
tryCatch(write_msg("uwu", italic = "FALSE"), error = show_last_error)
tryCatch(write_msg("uwu", bold = "FALSE"), error = show_last_error)

```

**collapse***collapse - to paste strings with collapse = ""***Description**

`collapse` does exactly what `paste` does, but default `collapse = ""`

**Usage**

```
collapse(..., sep = "")
```

**Arguments**

...	passed to <a href="#">base::paste0()</a>
<code>sep</code>	set the separator. Default is ""

**Value**

NULL

**Author(s)**

L.Pavot

**Examples**

```
collapse(list("a message ", "in multiple", "parts"))
```

---

collapse_lines	<i>collapse_lines - to paste strings with collapse = "\n"</i>
----------------	---

---

### Description

collapse\_lines() does exactly when paste does, but default collapse = "\n"

### Usage

```
collapse_lines(..., sep = "\n")
```

### Arguments

...	passed to <a href="#">base::paste0()</a>
sep	set the separator. Default is "\n"

### Value

NULL

### Author(s)

L.Pavot

### Examples

```
collapse_lines(list("a message ", "in multiple", "parts"))
```

---

---

convert_parameters	<i>Convert Parameters</i>
--------------------	---------------------------

---

### Description

convert\_parameters Applies a list of converters to each values on a list. If a value is modified during the conversion (successfull conversion) then, no further convert will be applied to this value, so values are only converted once.

### Usage

```
convert_parameters(args, converters)
```

### Arguments

args	a named list, which values will be converted.
converters	a vector of function. Each function will be applied to each values with the exception of values already converted by a previous converter.

**Value**

a named list object with values converted by converters.

**Author(s)**

L.Pavot

**Examples**

```
boolean_converter <- function(x) {
  return(if (x == "TRUE") TRUE else if (x == "FALSE") FALSE else x)
}
parameters <- W4MRUtils::convert_parameters(
  list("x" = "TRUE"),
  c(boolean_converter)
)
print(parameters$`some-parameter`)
## "TRUE" has becomes TRUE.
```

**df\_force\_numeric**      *Convert data frame to numeric.*

**Description**

**df\_force\_numeric** Converts integer columns of a data frame into numeric.

**Usage**

```
df_force_numeric(df, cols = NULL)
```

**Arguments**

<b>df</b>	The data frame.
<b>cols</b>	The set of columns to convert to numeric. By default (when set to NULL) all integer columns are converted. Set it to a character vector containing the names of the columns you want to convert, or ton integer vector containing the indices of the columns. Can be used to force conversion of non integer columns.

**Value**

The converted data.frame.

**Examples**

```
# Convert an integer data frame
df <- data.frame(a = as.integer(c(1, 4)), b = as.integer(c(6, 5)))
df <- W4MRUtils::df_force_numeric(df)
```

---

df_is	<i>Test type of a data frame.</i>
-------	-----------------------------------

---

## Description

df\_is This function tests if the columns of a data frame are all of the same type.

## Usage

```
df_is(df, type)
```

## Arguments

df	The data frame.
type	The type you expect the columns to have. It must be one of the R base types: - 'character' ; - 'factor' ; - 'integer' ; - 'numeric' ; - 'logical'.

## Value

TRUE or FALSE.

## Examples

```
# Test if a data frame contains only integers
df <- data.frame(a = c(1, 4), b = c(6, 5))
# should return FALSE since in R all integers are converted to
# numeric by default.
W4MRUtils::df_is(df, "integer")
# should return TRUE.
W4MRUtils::df_is(df, "numeric")
```

---

---

df_read_table	<i>Data frame loading from a file.</i>
---------------	--

---

## Description

df\_read\_table Reads a data frame from a file and possibly convert integer columns to numeric. This function calls the built-in `read.table()` method and then `W4MRUtils::df_force_numeric()`.

## Usage

```
df_read_table(file, force_numeric = FALSE, ...)
```

## Arguments

- `file` The path to the file you want to load. See `read.table()` documentation for more information.
- `force_numeric` If set to TRUE, all integer columns will be converted to numeric.
- `...` Parameter to transmit to the `read.table` function.

## Value

The loaded data frame.

## Examples

```
# Load a data frame from a file and convert integer columns
file_path <- system.file(
  "extdata",
  "example_df_read_table.csv",
  package="W4MRUtils"
)
str(W4MRUtils::df_read_table(
  file_path,
  sep = ",",
  force_numeric = TRUE,
  header=TRUE
))
```

---

## Description

Filtering tools to reduce a dataset, based on W4M 3-tables format Coded by: M.Petera,

R functions to use in R scripts and wrappers to make things easier

V0: script structure + first function

---

## Description

`generic_filter` This function allows to filter variables and samples according to factors or numerical values. It needs 3 datasets: the data matrix, the variables' metadata, the samples' metadata. It generates 3 new datasets corresponding to the 3 inputs filtered.

**Usage**

```
generic_filter(
  var.data,
  meta.samp.data,
  meta.var.data,
  NUM = FALSE,
  ls.num = NULL,
  FACT = FALSE,
  ls.fact = NULL
)
```

**Arguments**

var.data	Data.frame corresponding to the dataMatrix of your 3-tables data set
meta.samp.data	Data.frame corresponding to the sampleMetadata of your 3-tables data set
meta.var.data	Data.frame corresponding to the variableMetadata of your 3-tables data set
NUM	Logical; indicates whether some filtering based on numeric metadata columns should be performed
ls.num	List of character vectors; each vector corresponds to a numerical filter to perform; see Details for vector content
FACT	Logical; indicates whether some filtering based on qualitative metadata columns should be performed
ls.fact	List of character vectors; each vector corresponds to a qualitative filter to perform; see Details for vector content

**Details**

The vectors inside `ls.num` and `ls.fact` must follow a dedicated format. For `ls.fact`, each vector must be constituted of exactly 3 character strings in the following order: (i) the name of a column to filter in one of the metadata table, (ii) the modality inside this column to get rid off, and (iii) "sample" or "variable" depending on the type of metadata you are filtering (i.e. the table that contains the column to filter). For `ls.num`, each vector must be constituted of 4 to 5 character strings in the following order: (i) "sample" or "variable" depending on the type of metadata you are filtering (i.e. the table that contains the column to filter), (ii) the name of the column to filter in the corresponding metadata table, (iii) the type of numerical filtering to apply, to be chosen from "lower", "upper", "extremity" and "between", (iv) the threshold to be used for filtering (in case the 'iii' argument is "extremity" or "between", it is the lower threshold), and (v) in case the 'iii' argument is "extremity" or "between", the upper threshold.

**Value**

a list containing the 3 tables, filtered according to the criteria input

**Author(s)**

M.Petera

`get_base_dir`*get\_base\_dir - to get... the base directory***Description**`get_base_dir`**Usage**`get_base_dir()`**Value**

the directory path of the main script. PWD otherwise.

**Examples**`print(get_base_dir())``get_logger`*Instantiate a Logger***Description**Create a logger of the given name. You can call again `get_logger` and provide the same name to get the same logger. It will not be recreated unless recreate is TRUE.**Usage**`get_logger(name, recreate = FALSE, ...)`**Arguments**

<code>name</code>	the name of the logger to get or create. This name will be used in logs to differentiate from which part of your program comes which lines of log. See the example of usage below.
<code>recreate</code>	logical=FALSE tells whether to recreate the logger of the given name or not. Preferably, one should not recreate a new logger each time.
<code>...</code>	Parameters passed out to the <code>W4MLLogger()</code> function

**Details**`get_logger`

**Value**

A new W4MLLogger instance if it did not exist or if recreate is TRUE. Otherwise, a new W4MLLogger instance.

**Author(s)**

L.Pavot

---

`get_r_env`

*get\_r\_env - provides env vars begining with R\_\**

---

**Description**

Returns a list of env vars if the start with R\_\*.

**Usage**

`get_r_env()`

**Value**

a list of environment variables which begin by R\_.

**Author(s)**

L.Pavot

**See Also**

[run\\_galaxy\\_processing](#)

---

`import2`

*Import two W4M tables*

---

**Description**

import2 Function to import a metadata table file and its corresponding dataMatrix file. import2 performs checks to ensure the identifiers match between the two tables and stops with an explicit error message in case identifiers do not match.

**Usage**

`import2(pathDM, pathMeta, typeMeta, disable_comm = TRUE)`

### Arguments

pathDM	a path to a file corresponding to the dataMatrix
pathMeta	a path to a file corresponding to the metadata table
typeMeta	"sample" or "variable" depending on the metadata content
disable_comm	a boolean with default to TRUE to indicate whether the comment character # should be disabled as a comment tag for the import of the metadata file; when TRUE, # in the metadata table's columns will be considered as any other character.

### Value

a list containing two elements:

- dataMatrix a `data.frame` corresponding to the imported dataMatrix table;
- metadata a `data.frame` corresponding to the imported metadata table

### Author(s)

M.Petera

### Examples

```
dm_path <- system.file(
  "extdata",
  "mini_datamatrix.txt",
  package="W4MRUtils"
)
meta_path <- system.file(
  "extdata",
  "mini_variablemetadata.txt",
  package="W4MRUtils"
)

## import considering # is not a comment character
A <- W4MRUtils::import2(dm_path, meta_path, "variable")
print(A$dataMatrix[1:5, 1:5])
print(A$metadata[1:5, ])

## import considering # is a comment character
B <- W4MRUtils::import2(dm_path, meta_path, "variable", disable_comm = FALSE)
print(B$dataMatrix[1:5, 1:5])
print(B$metadata[1:5, ])
```

---

import3	<i>Import the three W4M tables</i>
---------	------------------------------------

---

## Description

import3 Function to import the three W4M tables from files (dataMatrix, sampleMetadata, variableMetadata) import3 performs checks to ensure the identifiers match between the three tables and stops with an explicit error message in case identifiers do not match.

## Usage

```
import3(pathDM, pathSM, pathVM, disable_comm = TRUE)
```

## Arguments

pathDM	a path to a file corresponding to the dataMatrix
pathSM	a path to a file corresponding to the sampleMetadata
pathVM	a path to a file corresponding to the variableMetadata
disable_comm	a boolean with default to TRUE to indicate whether the comment character # should be disabled as a comment tag for the import of the metadata files; when TRUE, # in the metadata table's columns will be considered as any other character.

## Value

a list containing three elements:

- dataMatrix a `data.frame` corresponding to the imported dataMatrix table;
- sampleMetadata a `data.frame` corresponding to the imported sampleMetadata table;
- variableMetadata a `data.frame` corresponding to the imported variableMetadata table

## Author(s)

M.Petera

## Examples

```
dm_path <- system.file(  
  "extdata",  
  "mini_datamatrix.txt",  
  package="W4MRUtils"  
)  
vm_path <- system.file(  
  "extdata",  
  "mini_variablemetadata.txt",  
  package="W4MRUtils"
```

```

)
sm_path <- system.file(
  "extdata",
  "mini_samplemetadata.txt",
  package="W4MRUtils"
)

## import considering # is not a comment character
A <- W4MRUtils::import3(dm_path, sm_path, vm_path)
print(A$dataMatrix[1:5, 1:5])
print(A$sampleMetadata[1:5, ])
print(A$variableMetadata[1:5, ])

## import considering # is a comment character
B <- W4MRUtils::import3(dm_path, sm_path, vm_path, disable_comm = FALSE)
print(B$dataMatrix[1:5, 1:5])
print(B$sampleMetadata[1:5, ])
print(B$variableMetadata[1:5, ])

```

**intensity\_checks***Intensity checking tools for Galaxy scripting***Description**

Intensity checking tools to compute indices per variable, based on the W4M 3-tables format Original code of first function by A.Fernandes, updates by M.Petera

R functions to use in R scripts and wrappers to make things easier

V0: script structure + first function (intens\_check)

**intens\_check***Checking intensities of a data set following the W4M 3-tables format***Description**

**intens\_check** This function allows to check the intensities with various statistics, number of missing values and mean fold change. It needs 3 datasets: the data matrix, the variables' metadata, the samples' metadata. It generates 3 new datasets corresponding to the 3 inputs filtered.

## Usage

```
intens_check(
  DM,
  SM,
  VM,
  method,
  chosen.stat,
  class.col,
  test.fold,
  class1,
  fold.frac,
  logarithm,
  graphs.output
)
```

## Arguments

DM	Data.frame corresponding to the dataMatrix of your 3-tables data set
SM	Data.frame corresponding to the sampleMetadata of your 3-tables data set
VM	Data.frame corresponding to the variableMetadata of your 3-tables data set
method	Character string among "global", "one_class", "each_class"
chosen.stat	Character string listing the chosen analysis (comma-separated) from mean, sd, median, quartile, decile and NA
class.col	Character string of the name of the sampleMetadata's column with classes (if method = one_class or each_class)
test.fold	Character string among "yes" or "no" to indicate whether mean fold changes should be computed (if method = one_class or each_class)
class1	Character string corresponding to the name of the chosen class (if method = one_class)
fold.frac	Character string; "Top" for class1/other or "Bottom" for other/class1 (if method = one_class)
logarithm	Character string among "log2", "log10" or "none", use for mean fold changes (if method = one_class or each_class)
graphs.output	Character string for the pdf file's access for plotting the graphics

## Details

Three methods proposed: (i) 'global' to perform tests for each variable without distinction between samples, (ii) 'one class' to compute test for one class versus all the remaining samples in an 'other' shared class, (iii) 'each class' in case the class columns contains at least two classes and you want to test each of them individually. The function produces a table with the requested indices, plus a PDF output with corresponding graphics.

## Value

a data frame corresponding to the original variableMetadata table completed with additional columns

**Author(s)**

A.Fernandes, M.Petera

---

`in_galaxy_env`

*in\_galaxy\_env - check if the script has been run by galaxy*

---

**Description**

`in_galaxy_env` returns TRUE if it detects some galaxy-specific environment variables. FALSE otherwise.

**Usage**

`in_galaxy_env()`

**Value**

A logical - whether the script has been run by galaxy or not.

---

`match2`

*Table match check functions*

---

**Description**

`match2` To check if data\_matrix and (variable or sample)metadata match regarding identifiers

**Usage**

`match2(data_matrix, metadata, metadata_type)`

**Arguments**

<code>data_matrix</code>	data.frame containing data_matrix
<code>metadata</code>	data.frame containing sample_metadata or variable_metadata
<code>metadata_type</code>	"sample" or "variable" depending on metadata content

**Value**

character vector a list of errors encountered

**Author(s)**

M.Petera

---

`match3``match3`

---

**Description**

`match3` To check if the 3 standard tables match regarding identifiers

**Usage**

```
match3(data_matrix, sample_metadata, variable_metadata)
```

**Arguments**

<code>data_matrix</code>	data.frame containing data_matrix
<code>sample_metadata</code>	data.frame containing sample_metadata
<code>variable_metadata</code>	data.frame containing variable_metadata

**Value**

character vector a list of errors encountered

**Author(s)**

M.Petera

---

`metab_merge`*Merging a dataMatrix with a metadata file*

---

**Description**

`metab_merge` Function to merge the dataMatrix table with one of its corresponding metadata table (sampleMetadata or variableMetadata)

**Usage**

```
metab_merge(DM, meta, metype = "sample")
```

**Arguments**

<code>DM</code>	Data.frame corresponding to the dataMatrix you want to merge with a metadata table
<code>meta</code>	Data.frame corresponding to the metadata table you want to merge with the dataMatrix
<code>metype</code>	Character string indicating the type of metadata table used; should be one between "sample" and "variable"

**Value**

a `data.frame` corresponding to a combined table with both intensities and metadata

**Author(s)**

M.Petera

`mini_tools`

*Mini tools for Galaxy scripting*

**Description**

Mini tools for Galaxy scripting Mini tools for Galaxy scripting Coded by: M.Petera,  
R functions to use in R scripts and wrappers to make things easier (lightening code, reducing verbose...)  
V0: script structure + first functions V1: addition of functions to handle special characters in identifiers  
V2: addition of a merging function

`optparse_character`

*optparse\_character - define a command parameter as string*

**Description**

To be used with `optparse_parameters`. This function tells the provided parameter is to be parsed as a single string.

**Usage**

```
optparse_character(help = "No documentation yet.", short = NULL, default = 0)
```

**Arguments**

- |                      |  |
|----------------------|--|
| <code>help</code>    | • The help string to display when <code>-help</code> is triggered  |
| <code>short</code>   | • The shortcut for this parameter. For example for a <code>-output</code> param, we could use <code>optparse_flag(short = "-o", ...)</code> to set the " <code>-o</code> " shortcut. |
| <code>default</code> | • The default value this parameter will hold.  |

**Value**

a list to give to `optparse_parameters` to build the whole command line parsing tool.

**Author(s)**

L.Pavot

**See Also**[optparse\\_parameters\(\)](#)**Examples**

```
str(optparse_parameters(  
    a_parameter = optparse_character(),  
    args = list("--a-parameter", "42")  
)
```

---

**optparse\_flag***optparse\_flag - define a command parameter as a trigger*

---

**Description**

To be used with `optparse_parameters`. This function tells the provided parameter is a trigger (logical - TRUE/FALSE). When the trigger parameter is not provided in the command line, the value is FALSE. Otherwise, it is TRUE.

**Usage**

```
optparse_flag(help = "No documentation yet.", short = NULL, default = FALSE)
```

**Arguments**

- |         |  |
|---------|--|
| help    | • The help string to display when <code>-help</code> is triggered  |
| short   | • The shortcut for this parameter. For example for a <code>-output</code> param, we could use <code>optparse_flag(short = "-o", ...)</code> to set the <code>"-o"</code> shortcut. |
| default | • The default value this parameter will hold.  |

**Value**

a list to give to `optparse_parameters` to build the whole command line parsing tool.

**Author(s)**

L.Pavot

**See Also**[optparse\\_parameters\(\)](#)

**Examples**

```
str(optparse_parameters(
    a_parameter = optparse_flag(),
    args = list("--a-parameter")
))
```

**optparse\_integer***optparse\_integer - define a command parameter as an integer***Description**

To be used with `optparse_parameters`. This function tells the provided parameter is to be parsed as an integer.

**Usage**

```
optparse_integer(help = "No documentation yet.", short = NULL, default = 0)
```

**Arguments**

- |         |   |
|---------|---|
| help    | • The help string to display when <code>--help</code> is triggered  |
| short   | • The shortcut for this parameter. For example for a <code>--output</code> param, we could use <code>optparse_flag(short = "-o", ...)</code> to set the <code>"-o"</code> shortcut. |
| default | • The default value this parameter will hold.   |

**Value**

a list to give to `optparse_parameters` to build the whole command line parsing tool.

**Author(s)**

L.Pavot

**See Also**

[optparse\\_parameters\(\)](#)

**Examples**

```
str(optparse_parameters(
    a_parameter = optparse_integer(),
    args = list("--a-parameter", "42")
))
```

---

**optparse\_list***optparse\_list - define a command parameter as a list of objects*

---

## Description

To be used with `optparse_parameters`. This function tells the provided parameter is to be parsed as a list of objects. The `of` parameter tells what type are elements of the list. Each element must be separated by a separator. This separator must be the value given in the `sep` parameter

## Usage

```
optparse_list(  
  help = "No documentation yet.",  
  short = NULL,  
  default = "",  
  of = "character",  
  sep = ",",  
  truevalues = c("TRUE", "true", "1", "t", "T")  
)
```

## Arguments

- |            |  |
|------------|--|
| help       | • The help string to display when <code>-help</code> is triggered  |
| short      | • The shortcut for this parameter. For example for a <code>-output</code> param, we could use <code>optparse_flag(short = "-o", ...)</code> to set the " <code>-o</code> " shortcut. |
| default    | • The default value this parameter will hold.  |
| of         | • This type of elements of this list   |
| sep        | • This character to split on, to get the list  |
| truevalues | • A character vector of different string values to translate it as TRUE value.   |

## Value

a list to give to `optparse_parameters` to build the whole command line parsing tool.

## Author(s)

L.Pavot

## See Also

[optparse\\_parameters\(\)](#)

## Examples

```
str(optparse_parameters(
  a_parameter = optparse_list(of="numeric"),
  b_parameter = optparse_list(of="integer"),
  c_parameter = optparse_list(of="logical"),
  args = list(
    "--a-parameter", "42.7,72.5",
    "--b-parameter", "42.7,72.5",
    "--c-parameter", "TRUE,FALSE,TRUE"
  )
))
```

**optparse\_numeric**

*optparse\_numeric - define a command parameter as an numeric*

## Description

To be used with `optparse_parameters`. This function tells the provided parameter is to be parsed as an numeric.

## Usage

```
optparse_numeric(help = "No documentation yet.", short = NULL, default = 0)
```

## Arguments

- |         |  |
|---------|--|
| help    | • The help string to display when <code>-help</code> is triggered  |
| short   | • The shortcut for this parameter. For example for a <code>-output</code> param, we could use <code>optparse_flag(short = "-o", ...)</code> to set the <code>"-o"</code> shortcut. |
| default | • The default value this parameter will hold.  |

## Value

a list to give to `optparse_parameters` to build the whole command line parsing tool.

## Author(s)

L.Pavot

## See Also

[optparse\\_parameters\(\)](#)

## Examples

```
str(optparse_parameters(
  a_parameter = optparse_numeric(),
  args = list("--a-parameter", "42.72")
))
```

`optparse_parameters`    *optparse\_parameters - parse easily the command line parameters*

## Description

This function is made to be used with the functions optparse\_flag, optparse\_numeric, optparse\_integer, optparse\_character and/or optparse\_list

`optparse_parameters` parses arguments based on its parameters.

You just have to call `optparse_parameters` with named arguments. Each parameter is the result of either optparse\_flag, optparse\_numeric, optparse\_integer, optparse\_character or optparse\_list

## Usage

```
optparse_parameters(
  fix_hyphens = TRUE,
  fix_dots = TRUE,
  add_trailing_hyphens = TRUE,
  args = NULL,
  no_optparse = FALSE,
  ...
)
```

## Arguments

<code>fix_hyphens</code>	logical - whether to turn underscores into hyphens or not
<code>fix_dots</code>	logical - whether to turn points into hyphens or not
<code>add_trailing_hyphens</code>	logical - whether to add trailing hyphens if missing
<code>args</code>	list - The parameters from the commandArgs function
<code>no_optparse</code>	logical - INTERNAL Tells whether to use optparse library or not
<code>...</code>	parameters definition. Must be the result of either those functions:
	<ul style="list-style-type: none"> <li>• <code>optparse_flag</code></li> <li>• <code>optparse_numeric</code></li> <li>• <code>optparse_integer</code></li> <li>• <code>optparse_character</code></li> <li>• <code>optparse_list</code></li> </ul>

**Value**

A list combining all the elements provided by the optparse\_flag, optparse\_numeric, optparse\_integer, optparse\_character and/or optparse\_list arguments

**Author(s)**

L.Pavot

**Examples**

```
args <- optparse_parameters(
  a_integer = optparse_integer(),
  a_float = optparse_numeric(),
  a_boolean = optparse_flag(),
  a_character = optparse_character(),
  a_list = optparse_list(of = "numeric"),
  a_char_list = optparse_list(of = "character"),
  a_int_list = optparse_list(of = "integer"),
  args = list(
    "--a-integer",
    "42",
    "--a-float",
    "3.14",
    "--a-boolean",
    "FALSE",
    "--a-character",
    "FALSE",
    "--a-list",
    "1.5,2,3",
    "--a-char-list",
    "1.5,2,3",
    "--a-int-list",
    "1.5,2,3"
  )
)

str(args)
```

**parse\_args**

*Parse Command arguments*

**Description**

`parse_args` Replacement for the `parseCommandArgs` utility from `batch`. Note that inputs like `script.R some-list c(1, 2, 3)` will result in `args$some-list` to be the string "`c(1, 2, 3)`", and not a vector anymore as this ability was permitted by dangerous behaviours from the `batch` package (the usage of `eval` which MUST NEVER be used on user's inputs).

To get a list of numeric from users, instead of using the `c(1, 2)` trick, please, use regular lists parsing:

```
> args$`some-list`
[1] "1,2"
args$`some-list` <- as.numeric(strsplit(args$`some-list`, ",")[[1]])
> args$`some-list`
[1] 1 2
```

**Usage**

```
parse_args(
  args = NULL,
  convert_booleans = TRUE,
  convert_numerics = TRUE,
  strip_trailing_dash = TRUE,
  replace_dashes = TRUE
)
```

**Arguments**

args	optional, provide arguments to parse. This function will use 'commandArgs()'
	if args is not provided
convert_booleans	logical - tells the function to convert values into logical if their value is "TRUE" or "FALSE".
convert_numerics	logical - tells the function to convert values into numeric if possible.
strip_trailing_dash	<ul style="list-style-type: none"> <li>• tells whether to remove trailing hyphens from the start of the parameter name</li> </ul>
replace_dashes	<ul style="list-style-type: none"> <li>• tells whether to turn trailing hyphens into underscores</li> </ul>

**Value**

a named list object containing the input parameters in values and the parameters names in names

**Author(s)**

L.Pavot

**Examples**

```
## faking command line parameters:

commandArgs <- function() {
  list(
    "--args",
    "param1", "a value",
    "param2", "42"
  )
}
```

```
## extracting command line parameters:
parameters <- W4MRUtils::parse_args(args = commandArgs())
str(parameters)
```

**printf***printf - to format a string and print it***Description**

`printf` calls `sprintf` of its parameters to build the error message and prints with the given message

**Usage**

```
printf(...)
```

**Arguments**

...	Arguments passed on to <a href="#">base::sprintf</a>
	<code>fmt</code> a character vector of format strings, each of up to 8192 bytes.

**Value**

NULL

**Author(s)**

L.Pavot

**Examples**

```
file <- "/tmp/test"
printf("Error in file: ", file)
```

**printfp***printfp - to paste, format and print a string***Description**

`printfp` calls `paste` and `sprintf` of its parameters to build the error message and prints with the given message

**Usage**

```
printfp(x, ...)
```

**Arguments**

- x a list of format string to concatenate before using sprintf on it.
- ... Arguments passed on to [base::paste](#)
- sep a character string to separate the terms. Not [NA\\_character\\_](#).
- collapse an optional character string to separate the results. Not [NA\\_character\\_](#).  
When collapse is a string, the result is always a string ([character](#) of length 1).
- recycle0 [logical](#) indicating if zero-length character arguments should result in the zero-length [character\(0\)](#). Note that when collapse is a string, recycle0 does *not* recycle to zero-length, but to "".

**Value**

NULL

**Author(s)**

L.Pavot

**Examples**

```
file <- "/tmp/test"
printfp(
  list(
    "Very log error message that needs to be cut on multiple lines,",
    "and paste back together, but there are formatings like",
    "%s for example, that provides a placeholder for parameters.",
    "Here %s value is %s."
  ), file
)
```

**printp**

*printp - to format a string and print it*

**Description**

printp calls sprintf of its parameters to build the error message and prints with the given message

**Usage**

`printp(...)`

**Arguments**

- ... Arguments passed on to [base::sprintf](#)
- fmt a character vector of format strings, each of up to 8192 bytes.

**Value**

```
NULL
```

**Author(s)**

L.Pavot

**Examples**

```
file <- "/tmp/test"
printp("Error in file: ", file)
```

**reproduce\_id**

*Reproduce ID*

**Description**

`reproduce_id` `reproduce_id()` reinjects original identifiers and original order into final tables

**Usage**

```
reproduce_id(data_matrix, metadata, metadata_type, id_match)
```

**Arguments**

<code>data_matrix</code>	data.frame containing <code>data_matrix</code>
<code>metadata</code>	data.frame containing sample <code>metadata</code> or variable <code>metadata</code>
<code>metadata_type</code>	"sample" or "variable" depending on <code>metadata</code> content
<code>id_match</code>	' <code>id_match</code> ' element produced by <code>stock_id</code>

**Value**

a named list with two elements: `data_matrix`: the processed data matrix with its original names and order `metadata`: the processed `metadata`, with its original names and order.

**Author(s)**

M.Petera

## Examples

```

myDM <- data.frame(data = 1:6, a = 2:7, b = 3:8, c = 2:7, d = 3:8, e = 2:7)
myvM <- data.frame(variable = 1:6, x = 4:9, y = 2:7, z = 3:8)

A <- W4MRUtils::stock_id(myDM, myvM, "variable")
myDM <- A$dataMatrix
myvM <- A$Metadata
A <- A$id.match

## processing that may change order or requires specific identifiers format
## ...
datamatrix <- as.data.frame(myDM)
sample_metadata <- as.data.frame(myvM)

B <- W4MRUtils::reproduce_id(datamatrix, sample_metadata, "variable", A)
datamatrix <- B$dataMatrix
sample_metadata <- B$Metadata

```

`run_galaxy_function`    *run\_galaxy\_function - automate running functions in galaxy*

## Description

This function executes the provided function as a galaxy processing. This provided function is expected to take two parameters:

- args, a list of command line parameters
- logger, the logger created for the tool

## Usage

```
run_galaxy_function(tool_name, func, ...)
```

## Arguments

tool_name	• The name of the tool
func	• The function to be run, after galaxy header is displayed
...	• Parameters propagated to <a href="#">run_galaxy_processing</a>

## Value

NULL (function called to provide checks to the execution of [run\\_galaxy\\_processing](#))

## Author(s)

L.Pavot

**See Also**

[run\\_galaxy\\_processing](#)

`run_galaxy_processing` *run\_galaxy\_processing - automate running code in galaxy*

**Description**

`run_galaxy_processing` takes the tool's name, and the code to execute. It detects galaxy-specific environment variable, and show headers and footer if we are in a galaxy env.

It will automatically convert command line parameters using `W4MRUtils::parse_args` if `args` is not provided.

Then, it unmangles galaxy parameters (galaxy params / values can be mangled if they contains special characters)

It creates a logger, and provide access to the logger and `args` variables from within the code to execute.

Also, before executing the code, if `source_files` is set to some paths, these paths will be source'd, so the code has access to functions defined in these scripts.

**Usage**

```
run_galaxy_processing(
  tool_name,
  code,
  tool_version = "unknown",
  unmangle_parameters = TRUE,
  args = NULL,
  logger = NULL,
  source_files = c(),
  env = NULL,
  do_traceback = FALSE
)
```

**Arguments**

- |                                  |  |
|----------------------------------|--|
| <code>tool_name</code>           | • Mandatory. The name of the tool to run.  |
| <code>code</code>                | • Mandatory. The code to run the tool  |
| <code>tool_version</code>        | • The version number of the tool to run.   |
| <code>unmangle_parameters</code> | <ul style="list-style-type: none"> <li>• Whether or not to revert mangling from galaxy useful if galaxy produces strange command parameters. Not necessary, but produces more explicit outputs.</li> </ul>   |
| <code>args</code>                | <ul style="list-style-type: none"> <li>• The result of <code>commandArgs</code> function, or from the <code>optparse_parameters</code> function. Can be <code>NULL</code>. In this case, uses <code>commandArgs</code> to get the args.</li> </ul> |

logger	<ul style="list-style-type: none"> <li>You can provide a logger to use. If not provided, a logger will be created with the tool's name.</li> </ul>
source_files	<ul style="list-style-type: none"> <li>You may provide some paths to source before executing the provided code.</li> </ul>
env	<ul style="list-style-type: none"> <li>You may provide a environment object to execute the code within.</li> </ul>
do_traceback	<ul style="list-style-type: none"> <li>logical - tells whether to produce a traceback in case of error.</li> </ul>

**Value**

NULL (the function only execute the inputed Galaxy-based process in a standardised manner)

**Author(s)**

L.Pavot

**Examples**

```
write_r_file_with_content <- function(content) {
  " "
  This function creates a temp R file. It writes the provided
  content in the R file. Then it returns the path of the script.
  "
  path <- tempfile(fileext = ".R")
  file.create(path)
  writeLines(content, con = path)
  return(path)
}
## let's fake a galaxy env
Sys.setenv(GALAXY_SLOTS = 1)
## let's says the tool has been launched with this command line
log_file <- tempfile()
file.create(log_file)
raw_args <- list(
  "--args",
  "--input", "in.csv",
  "--output", "out.csv",
  "--logs", log_file,
  "--one-float", "3.14",
  "--one-integer", "456",
  "--one-logical", "FALSE",
  "--some-floats", "1.5,2.4,3.3",
  "--some-characters", "test,truc,bidule",
  "--debug", "TRUE",
  "--verbose", "FALSE"
)
##
## 
# example 1
## 

my_r_script <- write_r_file_with_content('
my_processing <- function(args, logger) {
```

```

logger$info("The tool is running")
logger$infof("Input file: %s.", args$input)
logger$info("The tool ended.")
}
')

W4MRUtils::run_galaxy_processing(
  "Test tool 1",
  my_processing(args, logger),
  source_file = my_r_script,
  args = W4MRUtils::parse_args(args = raw_args)
)

## 
# example 2

## let's say we have a R script with this content:
path <- write_r_file_with_content('
setup_logger <- function(args, logger) {
  if (!is.null(args$verbose)) {
    logger$set_verbose(args$verbose)
  }
  if (!is.null(args$debug)) {
    logger$set_debug(args$debug)
  }
  if (!is.null(args$logs)) {
    logger$add_out_paths(args$logs)
  }
}
stop_logger <- function(logger) {
  logger$close_files()
}
processing <- function(args, logger) {
  setup_logger(args, logger)
  logger$info("The tool is working...")
  logger$infof("Input: %s.", args$input)
  logger$info("The tool stoping.")
  stop_logger(logger)
  return(NULL)
}')

## wrapper script:

args <- W4MRUtils::optparse_parameters(
  input = W4MRUtils::optparse_character(),
  output = W4MRUtils::optparse_character(),
  logs = W4MRUtils::optparse_character(),
  one_float = W4MRUtils::optparse_numeric(),
  one_integer = W4MRUtils::optparse_integer(),
  one_logical = W4MRUtils::optparse_flag(),
  some_floats = W4MRUtils::optparse_list(of = "numeric"),

```

```
some_characters = W4MUtils::optparse_list(of = "character"),
debug = W4MUtils::optparse_flag(),
verbose = W4MUtils::optparse_flag(),
args = raw_args[raw_args != "--args"]
)

W4MUtils::run_galaxy_processing("A Test tool", args = args, {
  ## processing is from the other R script
  processing(args, logger)
}, source_files = path)
```

---

show\_galaxy\_footer      *show\_galaxy\_footer - shows the footer for galaxy tools*

---

## Description

This function prints the footer to display in galaxy's tools logs

## Usage

```
show_galaxy_footer(
  tool_name,
  tool_version,
  logger = NULL,
  show_packages = TRUE,
  ellapsed = NULL
)
```

## Arguments

tool_name	a character(1) holding the running tool's name.
tool_version	a character(1) holding the running tool's version.
logger	a get_logger("name") instance - if provided, the galaxy footer is output from the logger.
show_packages	logical - Tells whether to display loaded packages and attached packages.
ellapsed	NULL or a character(1) with execution duration.

## Value

NULL

## Author(s)

L.Pavot

**See Also**

[run\\_galaxy\\_processing](#)

**Examples**

```
show_galaxy_footer("Tool Name", "1.2.0")

show_galaxy_footer(
  tool_name = "Tool Name",
  tool_version = "1.2.0",
  logger = get_logger("Some Tool"),
  show_packages = FALSE,
  ellapsed = "14.5 seconds"
)
```

**show\_galaxy\_header**      *show\_galaxy\_header - shows the header for galaxy tools*

**Description**

This function prints the header to display in galaxy's tools logs

**Usage**

```
show_galaxy_header(
  tool_name,
  tool_version,
  args = NULL,
  logger = NULL,
  show_start_time = TRUE,
  show_sys = TRUE,
  show_parameters = TRUE
)
```

**Arguments**

- |                              |   |
|------------------------------|---|
| <code>tool_name</code>       | a character(1) holding the running tool's name.   |
| <code>tool_version</code>    | a character(1) holding the running tool's version.  |
| <code>args</code>            | a list(param="value") - if provided, their are the parameters shown in galaxy header and/or footer.                   |
| <code>logger</code>          | a get_logger("name") instance - if provided, the galaxy footer is output from the logger.                             |
| <code>show_start_time</code> | <ul style="list-style-type: none"> <li>• a logical telling whether to display the start time or not.</li> </ul>       |
| <code>show_sys</code>        | <ul style="list-style-type: none"> <li>• a logical telling whether to display the system variables or not.</li> </ul> |
| <code>show_parameters</code> | <ul style="list-style-type: none"> <li>• a logical telling whether to display the parameters or not.</li> </ul>       |

**Value**

NULL

**Author(s)**

L.Pavot

**See Also**[run\\_galaxy\\_processing](#)**Examples**

```
show_galaxy_header("Tool Name", "1.2.0")
show_galaxy_header(
  tool_name = "Tool Name",
  tool_version = "1.2.0",
  logger = get_logger("Some Tool"),
  show_start_time = FALSE,
  show_sys = FALSE,
  show_parameters = FALSE
)
```

---

**show\_sys***show\_sys - prints env variables related to R*

---

**Description**

prints env variables related to R

**Usage**`show_sys()`**Value**

NULL

**Author(s)**

L.Pavot

**Examples**`show_sys()`

**shy\_lib***Shy Lib***Description**

`shy_lib` Function to call packages without printing all the verbose (only getting the essentials, like warning messages for example)

**Usage**

```
shy_lib(...)
```

**Arguments**

...	Name of libraries to load
-----	---------------------------

**Value**

a list of attached packages

**Author(s)**

M.Petera

**Examples**

```
W4MRUtils::shy_lib("base", "utils")
```

**source\_local***source\_local - source file, from absolute or relative path***Description**

`source_local` Transforms a relative path to an absolute one, and sources the path. This helps source files located relatively to the main script without the need to know from where it was run.

**Usage**

```
source_local(..., env = FALSE, do_print = FALSE, keep_source = TRUE)
```

**Arguments**

...	paths, character vector of file paths to source
env	an environment in which to source the paths
do_print	a logical, telling whether to print sourced paths or not
keep_source	See the parameter keep.source from source

**Value**

a vector resulting from the sourcing of the files provided.

**See Also**

[source\(\)](#)

---

stock\_id

*Stock ID*

---

**Description**

stock\_id Functions to stock identifiers before applying make.names() and to reinject it into final matrices. stock\_id stocks original identifiers and original order needs checked data regarding table match.

**Usage**

`stock_id(data_matrix, metadata, metadata_type)`

**Arguments**

data_matrix	a <code>data.frame</code> containing the <code>data_matrix</code>
metadata	a <code>data.frame</code> containing samplemetadata or variablemetadata
metadata_type	"sample" or "variable" depending on metadata content

**Value**

a names list with three elements:

- id.match a `data.frame` that contains original order of ids, names ;
- dataMatrix the modified data matrix with names sanitized
- Metadata the modified metadata matrix with names sanitized This object can be used in `reproduce_id()` to replace sanitized names in data matrix by original ones, in the right order.

**Author(s)**

M.Petera

## Examples

```

myDM <- data.frame(data = 1:6, a = 2:7, b = 3:8, c = 2:7, d = 3:8, e = 2:7)
myvM <- data.frame(variable = 1:6, x = 4:9, y = 2:7, z = 3:8)

A <- W4MRUtils::stock_id(myDM, myvM, "variable")
myDM <- A$dataMatrix
myvM <- A$Metadata
A <- A$id.match

## processing that may change order or requires specific identifiers format
## ...
datamatrix <- as.data.frame(myDM)
sample_metadata <- as.data.frame(myvM)

B <- W4MRUtils::reproduce_id(datamatrix, sample_metadata, "variable", A)
datamatrix <- B$dataMatrix
sample_metadata <- B$Metadata

```

**stopaste**

*stopaste - to paste string to a message and stop*

## Description

stopaste calls paste of its parameters to build the error message and stops with the given message

## Usage

```
stopaste(...)
```

## Arguments

- ... Arguments passed on to [base::paste](#)
- sep a character string to separate the terms. Not [NA\\_character\\_](#).
- collapse an optional character string to separate the results. Not [NA\\_character\\_](#).  
When collapse is a string, the result is always a string ([character](#) of length 1).
- recycle0 [logical](#) indicating if zero-length character arguments should result in the zero-length [character\(0\)](#). Note that when collapse is a string, recycle0 does *not* recycle to zero-length, but to "".

## Value

NULL

**Author(s)**

L.Pavot

**Examples**

```
tryCatch({  
  file <- "/tmp/test"  
  stopaste("Error in file: ", file)  
, error = function(error) {  
  print(error)  
})
```

---

**stopaste0***stopaste0 - to paste string to a message and stop*

---

**Description**

stopaste calls paste0 of its parameters to build the error message and stops with the given message

**Usage**

```
stopaste0(...)
```

**Arguments**

... Arguments passed on to [base::paste0](#)  
collapse an optional character string to separate the results. Not [NA\\_character\\_](#).  
When collapse is a string, the result is always a string ([character](#) of length 1).  
recycle0 [logical](#) indicating if zero-length character arguments should result  
in the zero-length [character\(0\)](#). Note that when collapse is a string,  
recycle0 does *not* recycle to zero-length, but to "".

**Value**

```
NULL
```

**Author(s)**

L.Pavot

## Examples

```
tryCatch({
  file <- "/tmp/test"
  stopaste0("Error in file: ", file)
}, error = function(error) {
  print(error)
})
```

**stopf**

*stopf - to stop and format message*

## Description

stopf calls sprintf of its parameters to build the error message and stops with the given message

## Usage

```
stopf(...)
```

## Arguments

...	Arguments passed on to <a href="#">base::sprintf</a>
	fmt a character vector of format strings, each of up to 8192 bytes.

## Value

NULL

## Author(s)

L.Pavot

## Examples

```
tryCatch({
  file <- "/tmp/test"
  stopf("Error in %s file.", file)
}, error = function(error) {
  print(error)
})
```

---

unmangle\_galaxy\_param *unmangle\_galaxy\_param* - revert effects of galaxy manglings.

---

## Description

When running a tool from galaxy, the command line may be altered because some forbidden chars have been translated by galaxy.

This function takes args are invert the galaxy's mangling process.

## Usage

```
unmangle_galaxy_param(args)
```

## Arguments

args                named list - contains params\_name=value.

## Value

a named list - with unmangled parameter name and values.

## Author(s)

L.Pavot

## See Also

[run\\_galaxy\\_processing](#), [unmangle\\_galaxy\\_string](#)

---

unmangle\_galaxy\_string

*unmangle\_galaxy\_string* - revert effects of galaxy mangling

---

## Description

Revert effect of string mangling from galaxy on the given string.

## Usage

```
unmangle_galaxy_string(string)
```

## Arguments

string                • the character vector to fix mangling.

**Value**

string - the character vector, fixed.

**Author(s)**

L.Pavot

**See Also**

[run\\_galaxy\\_processing](#), [unmangle\\_galaxy\\_param](#)

W4MLogger

*The W4MLogger Class*

**Description**

This is a simple logger used to make uniform outputs across W4M tools.

See [get\\_logger](#) for example usages.

**Arguments**

name	character vector of length 1 - The name of the logger. Use different loggers with a name specific to each part of your program. The name will appear in the log prefix and helps to determine which part of the program did what
format	character vector of length 1 - The format string for each log line. The default is : "[{{ level }}-{{ name }}-{{ time }}] - {{ message }}"
do_coloring	logical vector of length 1 - By default, the logger uses special control character to give some coloring to the text, depending on the log level (info, warning, error, debug or verbose). This coloring is deactivated in files and if <code>W4MRUtils::in_galaxy_env()</code> returns TRUE. You can force or deactivate the coloring with this parameter.
show_debug	logical vector of length 1 - Tells whether the debug logs must be displayed/written or not. Default is FALSE
show_verbose	logical vector of length 1 - Tells whether the verbose logs must be displayed/written or not. Default is FALSE
show_info	logical vector of length 1 - Tells whether the info logs must be displayed/written or not. Default is TRUE.
show_warning	logical vector of length 1 - Tells whether the warning logs must be displayed/written or not. Default is TRUE.
show_error	logical vector of length 1 - Tells whether the error logs must be displayed/written or not. Default is TRUE.
coloring	named list - This lists maps a logging level to its coloring. Like this: <code>list(debug = "purple", info = "green")</code> Available colors can be found in <code>W4MRUtils::w4m_colors__</code> .
out_func	function - the default function to print messages in the terminal. The default is <code>base::message</code> .

out\_path list of file paths - Provide a list of file path where the logs will be written. It is not possible to separate different levels of logs in different log files for the moment.

**Value**

A W4MLogger instance

**Author(s)**

L.Pavot

**See Also**

[W4MLogger\\$info](#), [W4MLogger\\$warning](#), [W4MLogger\\$error](#), [W4MLogger\\$debug](#), [W4MLogger\\$verbose](#)

---

W4MLogger\_.message\_\_    *W4MLogger\_.message\_\_*

---

**Description**

The function W4MLogger\$.message\_\_ is the function that gets automatically called when W4MLogger\$info, W4MLogger\$debug, W4MLogger\$warning, W4MLogger\$error or W4MLogger\$verbose are invoked. This function is not truly internal, so it has to be considered as external, but should not be exported:

This means its has to do type checking of its inputs, and consider parameters as unsafe.

See [get\\_logger](#) for example usages.

**Arguments**

level is a string. By default its value should be either "info", "debug", "warning", "debug", "verbose" or "INTERNAL". But, if the logger was build with a different color naming, one of the names provided in the "coloring" named list parameter must be used, as it determines the color to use.

... anything, of any length. If this is not a character vector, then, its displayable value will be obtained with `capture.output(str(...))` If the resulting character vector's length is greater than one, then multiple messages will be printed.

**Value**

this logger's instance ( .self )

---

W4MLLogger\_add\_out\_paths

*Adds a file where logs are duplicated*

---

**Description**

W4MLLogger can output logs in file. This function adds a file in which to put logs.

---

---

W4MLLogger\_finalize*W4MLLogger\_finalize*

---

**Description**

The function W4MLLogger\$finalize is the destructor function of this class. It closes every files that was opened by the logger, or that was provided during execution. It has to be considered internal.

The function W4MLLogger\$finalize is the destructor function of this class. It closes every files that was opened by the logger, or that was provided during execution. It has to be considered internal.

The function W4MLLogger\$finalize is the destructor function of this class. It closes every files that was opened by the logger, or that was provided during execution. It has to be considered internal.

The function W4MLLogger\$finalize is the destructor function of this class. It closes every files that was opened by the logger, or that was provided during execution. It has to be considered internal.

The function W4MLLogger\$finalize is the destructor function of this class. It closes every files that was opened by the logger, or that was provided during execution. It has to be considered internal.

---

The function W4MLLogger\$finalize is the destructor function of this class. It closes every files that was opened by the logger, or that was provided during execution. It has to be considered internal.

---

W4MLLogger\_set\_debug*W4MLLogger\$set\_debug*

---

**Description**

This method activate or deactivate the logging of debugs messages

**Arguments**

value	logical TRUE/FALSE to activate/deactivate debug logging
default	logical set to TRUE to use debug by default

**Value**

.self the current W4MLLogger instance

---

W4MLLogger\_set\_error     *W4MLLogger\$set\_error*

---

**Description**

This method activate or deactivate the logging of errors messages

**Arguments**

value	logical TRUE/FALSE to activate/deactivate error logging
default	logical set to TRUE to use error by default

**Value**

.self the current W4MLLogger instance

---

---

W4MLLogger\_set\_info     *W4MLLogger\$set\_info*

---

**Description**

This method activate or deactivate the logging of info messages

**Arguments**

value	logical TRUE/FALSE to activate/deactivate info logging
default	logical set to TRUE to use info by default

**Value**

.self the current W4MLLogger instance

---

---

W4MLLogger\_set\_out\_paths

*Defines in which file logs are duplicated*

---

**Description**

W4MLLogger can output logs in file. This function tells in which file to put logs.

---

W4MLogger\_set\_verbose    *W4MLogger\$set\_verbose*

---

### Description

This method activate or deactivate the logging of verbose messages

### Arguments

value	logical TRUE/FALSE to activate/deactivate verbose logging
default	logical set to TRUE to use verbose by default

### Value

.self the current W4MLogger instance

---

W4MLogger\_set\_warning    *W4MLogger\$set\_warning*

---

### Description

This method activate or deactivate the logging of warnings messages

### Arguments

value	logical TRUE/FALSE to activate/deactivate warning logging
default	logical set to TRUE to use warning by default

### Value

.self the current W4MLogger instance

---

W4MLogger\_[info,warning,error,debug,verbose]  
*Log info/warning/error/debug/verbose messages*

---

## Description

Call one of the following function when you want a message to be printed or written in a log file:

- `your_logger$info("A info message") ;`
- `your_logger$warning("A warning message") ;`
- `your_logger$error("A error message") ;`
- `your_logger$debug("A debug message") ;`
- `your_logger$verbose.("A verbose. message")`

If the corresponding level is activated (with `your_logger$set_info(TRUE)`, `your_logger$set_debug(TRUE)`, etc...), these functions will print the message provided in the terminal and in logs files, if there were some provided at the creation of the logger.

If the corresponding log level is deactivated, these function will not do anything. So, do not hesitate to use them a lot, and activate them when needed.

See [get\\_logger](#) for example usages.

## Author(s)

L.Pavot

# Index

base::paste, 39, 50  
base::paste0, 51  
base::paste0(), 16, 17  
base::sprintf, 38, 39, 52  
  
character, 39, 50, 51  
check\_err, 3  
check\_one\_character, 4, 8, 11, 14  
check\_one\_complex, 4, 8, 11, 14  
check\_one\_integer, 5, 8, 11, 14  
check\_one\_logical, 5, 8, 11, 14  
check\_one\_numeric, 6, 8, 11, 14  
check\_param\_type\_n\_length, 4–6, 13  
check\_parameter\_length, 4–6, 7, 8, 11, 14  
check\_parameter\_type, 4–6, 8, 10, 11, 14  
collapse, 16  
collapse\_lines, 17  
commandArgs, 42  
convert\_parameters, 17  
  
df\_force\_numeric, 18  
df\_is, 19  
df\_read\_table, 19  
  
Filtering\_datasets, 20  
  
generic\_filter, 20  
get\_base\_dir, 22  
get\_logger, 22, 54, 55, 59  
get\_r\_env, 23  
  
import2, 23  
import3, 25  
in\_galaxy\_env, 28  
intens\_check, 26  
intensity\_checks, 26  
  
logical, 39, 50, 51  
  
match2, 28  
match3, 29  
  
metab\_merge, 29  
mini\_tools, 30  
  
NA\_character\_, 39, 50, 51  
  
optparse\_character, 30  
optparse\_flag, 31  
optparse\_integer, 32  
optparse\_list, 33  
optparse\_numeric, 34  
optparse\_parameters, 35, 42  
optparse\_parameters(), 31–34  
  
parse\_args, 36  
printf, 38  
printfp, 38  
printp, 39  
  
reproduce\_id, 40  
run\_galaxy\_function, 41  
run\_galaxy\_processing, 23, 41, 42, 42, 46, 47, 53, 54  
  
show\_galaxy\_footer, 45  
show\_galaxy\_header, 46  
show\_sys, 47  
shy\_lib, 48  
source(), 49  
source\_local, 48  
stock\_id, 49  
stopaste, 50  
stopaste0, 51  
stopf, 52  
  
unmangle\_galaxy\_param, 53, 54  
unmangle\_galaxy\_string, 53, 53  
  
W4MLLogger, 54  
W4MLLogger\$debug, 55  
W4MLLogger\$debug  
    (W4MLLogger\_[info,warning,error,debug,verbose]), 59

W4MLogger\$error, [55](#)  
W4MLogger\$error  
    (W4MLogger\_[info,warning,error,debug,verbose]),  
    [59](#)  
W4MLogger\$info, [55](#)  
W4MLogger\$info  
    (W4MLogger\_[info,warning,error,debug,verbose]),  
    [59](#)  
W4MLogger\$verbose, [55](#)  
W4MLogger\$verbose  
    (W4MLogger\_[info,warning,error,debug,verbose]),  
    [59](#)  
W4MLogger\$warning, [55](#)  
W4MLogger\$warning  
    (W4MLogger\_[info,warning,error,debug,verbose]),  
    [59](#)  
W4MLogger\_.message\_, [55](#)  
W4MLogger\_[info,warning,error,debug,verbose],  
    [59](#)  
W4MLogger\_add\_out\_paths, [56](#)  
W4MLogger\_finalize, [56](#)  
W4MLogger\_set\_debug, [56](#)  
W4MLogger\_set\_error, [57](#)  
W4MLogger\_set\_info, [57](#)  
W4MLogger\_set\_out\_paths, [57](#)  
W4MLogger\_set\_verbose, [58](#)  
W4MLogger\_set\_warning, [58](#)