

# Package ‘nfc.core.utils’

June 22, 2026

**Version** 0.0.4

**Date** 2026-06-17

**Title** Utilities for 'nf-core' Modules

**Depends** R (>= 4.4.0),

**Imports** utils

**Description** Provides utility functions to facilitate the use of R within nf-core modules.

The package helps parse 'Nextflow' inputs and perform validation checks to ensure correct parameter handling and reproducible execution.

For more details see Ewels (2020) <[doi:10.1038/s41587-020-0439-x](https://doi.org/10.1038/s41587-020-0439-x)>.

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.3.3

**VignetteBuilder** knitr

**Suggests** testthat (>= 3.0.0), rmarkdown, BiocStyle, knitr, withr, yaml

**Config/testthat/edition** 3

**BugReports** <https://github.com/nf-core/r-nf-core-utils/issues>

**URL** <https://github.com/nf-core/r-nf-core-utils>

**NeedsCompilation** no

**Author** nf-core Community [aut],  
Louis Le Nezet [aut, cre, cph] (ORCID:  
<<https://orcid.org/0009-0000-0202-2703>>)

**Maintainer** Louis Le Nezet <louislenezet@gmail.com>

**Repository** CRAN

**Date/Publication** 2026-06-22 15:10:02 UTC

## Contents

nfc.core.utils-package . . . . .	2
create_log_session_info . . . . .	3

create_versions.yml . . . . .	4
is_null_or_invalid . . . . .	5
is_valid_string . . . . .	5
nullify . . . . .	6
parse_arguments . . . . .	6
process_end . . . . .	7
process_inputs . . . . .	8
read_delim_flexible . . . . .	9
validate_boolean . . . . .	10
validate_double . . . . .	11
validate_file . . . . .	11
validate_folder . . . . .	12
validate_integer . . . . .	12
valid_string . . . . .	13

<b>Index</b>	<b>14</b>
--------------	-----------

---

nfcore.utils-package    *The nf-core utils R package for Nextflow processes*

---

## Description

This package aims to ease the link between Nextflow inputs and outputs to the R variable needed in the script.

## Functions

Below are listed some of the main functions available:

`process_inputs()`: Given a list of expected options with their default this function will parse the arguments given and validate their value based on the expected rules.

`process_end()`: Create a versions.yml and R\_sessionInfo.log file in the directory provided. The version file will be populated with the R version, the version of nfcore.utils and the version of the additional packages given.

## Author(s)

**Maintainer:** Louis Le Nezet <louislenezet@gmail.com> ([ORCID](#)) [copyright holder]

Authors:

- nf-core Community

## See Also

Useful links:

- <https://github.com/nf-core/r-nf-core-utils>
- Report bugs at <https://github.com/nf-core/r-nf-core-utils/issues>

**Examples**

```
library(nfcore.utils)
td <- withr::local_tempdir()
test_file_path <- file.path(td, "test_file.txt")
file.create(test_file_path)
my_options <- list(
  input_file = test_file_path,
  output_file = "prefix",
  threshold = 0.5
)
args <- c("--threshold 0.7")
processed_options <- process_inputs(
  my_options, args,
  keys_to_nullify = c("input_file", "output_file"),
  expected_files = c("input_file"),
  expected_double = c("threshold"),
  required_opts = c("input_file", "threshold")
)
## Not run:
process_end(
  packages = list(
    "r-stats" = "stats"
  ),
  task_name = 'MY_PROCESS'
)

## End(Not run)
```

---

create\_log\_session\_info

*Log R session info*

---

**Description**

This function logs the R session info to a file named `R_sessionInfo.log` in the specified output directory.

**Usage**

```
create_log_session_info(log_path = "R_sessionInfo.log")
```

**Arguments**

<code>log_path</code>	Path to the file where the R session info log will be written to. Default is the in the current directory as <code>R_sessionInfo.log</code> .
-----------------------	---

**Value**

R session info log

## Examples

```
td <- withr::local_tempdir()
create_log_session_info(file.path(td, "session.log"))
```

---

create\_versions.yml *Create versions.yml file*

---

## Description

This function allows to automatically create the versions.yml file used to store the packages used. r-base as well as r-nfcore.utils versions will automatically be added.

## Usage

```
create_versions.yml(packages, task_name, versions_path = "versions.yml")
```

## Arguments

packages	Named list of packages to add to the versions.yml. The items names should be the conda package name, while the items value should be the package name used in R.
task_name	Name of the nextflow process. Typically <code>\${task.process}</code>
versions_path	Path to the yml file where the versions will be written to. Default is the in the current directory as versions.yml

## Value

versions.yml file

## Examples

```
td <- withr::local_tempdir()
create_versions.yml(
  list("r-stats" = "stats"), "MY_PROCESS",
  file.path(td, "my_versions.yml")
)
```

---

is\_null\_or\_invalid      *Check if is NULL or invalid*

---

**Description**

Check if is NULL or invalid

**Usage**

```
is_null_or_invalid(x, variable_name = NULL)
```

**Arguments**

x                      String to check  
variable\_name      Name of the variable to print if error occurs

**Value**

TRUE if is null, ERROR if invalid and FALSE otherwise

**Examples**

```
is_null_or_invalid(NULL) # TRUE  
try(is_null_or_invalid(" ")) # ERROR  
is_null_or_invalid("Hello World !") # FALSE
```

---

is\_valid\_string              *Check for Non-Empty, Non-Whitespace String*

---

**Description**

This function checks if the input is non-NULL and contains more than just whitespace. It returns TRUE if the input is a non-empty, non-whitespace string, and FALSE otherwise.

**Usage**

```
is_valid_string(input)
```

**Arguments**

input                      A variable to check.

**Value**

A logical value: TRUE if the input is a valid, non-empty, non-whitespace string; FALSE otherwise.

**Examples**

```
is_valid_string("Hello World") # Returns TRUE
is_valid_string(" Hello World ") # Returns TRUE
is_valid_string("  ") # Returns FALSE
is_valid_string(NULL) # Returns FALSE
```

---

nullify

*Turn "null" or empty strings into actual NULL*


---

**Description**

Turn "null" or empty strings into actual NULL

**Usage**

```
nullify(x)
```

**Arguments**

x                    Input option

**Value**

NULL or x

**Examples**

```
nullify("null") # Returns NULL
nullify("NULL") # Returns NULL
nullify("  ") # Returns NULL
nullify("") # Returns NULL
nullify(NULL) # Returns NULL
nullify(NA) # Returns NULL
nullify("Hello") # Returns "Hello"
```

---

parse\_arguments

*Parse out options from a string without recourse to optparse*


---

**Description**

If only the key is given, the value will be set to TRUE

**Usage**

```
parse_arguments(x)
```

**Arguments**

x Long-form argument list like `--opt1 val1 --opt2 val2`

**Value**

named list of options and values similar to `optparse`

**Examples**

```
parse_arguments("--opt1 val1 --opt2 val2")
parse_arguments(' --opt1-extra "value with space" --opt2 val2 ')
```

---

<code>process_end</code>	<i>Process end of the workflow</i>
--------------------------	------------------------------------

---

**Description**

Create the `versions.yml` file and log session info at the end of the process.

**Usage**

```
process_end(
  packages,
  task_name,
  versions_path = "versions.yml",
  log_path = "R_sessionInfo.log"
)
```

**Arguments**

<code>packages</code>	Named list of packages to add to the <code>versions.yml</code> . The items names should be the conda package name, while the items value should be the package name used in R.
<code>task_name</code>	Name of the nextflow process. Typically <code>\${task.process}</code>
<code>versions_path</code>	Path to the <code>yml</code> file where the versions will be written to. Default is the in the current directory as <code>versions.yml</code>
<code>log_path</code>	Path to the file where the R session info log will be written to. Default is the in the current directory as <code>R_sessionInfo.log</code> .

**Value**

`versions.yml` file and R session info log

**Examples**

```
td <- withr::local_tempdir()
process_end(
  list("r-stats" = "stats"),
  "MY_PROCESS",
  file.path(td, "my_versions.yml"),
  file.path(td, "session.log")
)
```

---

process\_inputs

*Process input parameters and files*


---

**Description**

This function processes input parameters and files, applying parameter overrides, validating expected keys and files, and ensuring that the provided file paths are valid.

**Usage**

```
process_inputs(
  opt,
  args = NULL,
  keys_to_nullify = NULL,
  expected_files = NULL,
  expected_folders = NULL,
  expected_double = NULL,
  expected_integer = NULL,
  expected_boolean = NULL,
  required_opts = NULL
)
```

**Arguments**

opt	A list of default options.
args	A character vector of command-line arguments to override defaults.
keys_to_nullify	A character vector of keys that should be nullified.
expected_files	A character vector of keys in opt that are expected to be file paths.
expected_folders	A character vector of keys in opt that are expected to be folder paths.
expected_double	A character vector of keys in opt that are expected to be float values.
expected_integer	A character vector of keys in opt that are expected to be integer values.
expected_boolean	A character vector of keys in opt that are expected to be boolean values.
required_opts	A character vector of keys in opt that are required and must not be null or empty.

**Value**

A list of processed options with overrides applied and validated.

**Examples**

```
td <- withr::local_tempdir()
test_file_path <- file.path(td, "test_file.txt")
file.create(test_file_path)
my_options <- list(
  input_file = test_file_path,
  output_file = "prefix",
  threshold = 0.5,
  "is-a-test" = NULL
)
args <- c("--threshold 0.7 --is-a-test")
processed_options <- process_inputs(
  my_options, args,
  keys_to_nullify = c("input_file", "output_file"),
  expected_files = c("input_file"),
  expected_double = c("threshold"),
  expected_boolean = c("is-a-test"),
  required_opts = c("input_file", "threshold")
)
```

---

read\_delim\_flexible    *Flexibly read CSV or TSV files*

---

**Description**

The following rules are applied:

- extension == txt or tsv, then separator is set to tabulation
- extension == csv, then separator is set to comma

**Usage**

```
read_delim_flexible(file, ...)
```

**Arguments**

file	Input file
...	Arguments passed on to <code>utils::read.delim</code>
header	a logical value indicating whether the file contains the names of the variables as its first line. If missing, the value is determined from the file format: header is set to TRUE if and only if the first row contains one fewer field than the number of columns.

`quote` the set of quoting characters. To disable quoting altogether, use `quote = ""`. See `scan` for the behaviour on quotes embedded in quotes. Quoting is only considered for columns read as character, which is all of them unless `colClasses` is specified.

`dec` the character used in the file for decimal points.

`fill` logical. If TRUE then in case the rows have unequal length, blank fields are implicitly added. See ‘Details’.

`comment.char` character: a character vector of length one containing a single character or an empty string. Use `""` to turn off the interpretation of comments altogether.

## Value

output Data frame

---

validate_boolean	<i>Check if value is a boolean</i>
------------------	------------------------------------

---

## Description

Convert to TRUE or FALSE with accepted lowered values as:

- TRUE: 1, yes, true
- FALSE: 0, no, false

## Usage

```
validate_boolean(x, variable_name = NULL)
```

## Arguments

`x` String to check and convert

`variable_name` Name of the variable to print if error occurs

## Value

`x` as TRUE/FALSE, or NULL if is null and ERROR if `x` isn't convertible to a boolean

## Examples

```
validate_boolean(NULL) # NULL
try(validate_boolean(" ")) # ERROR
try(validate_boolean("Hello World !")) # ERROR
try(validate_boolean(12.4)) # ERROR
validate_boolean("1") # TRUE
validate_boolean(0) # FALSE
```

---

validate_double	<i>Check if value is a double number</i>
-----------------	--

---

**Description**

Check if value is a double number

**Usage**

```
validate_double(x, variable_name = NULL)
```

**Arguments**

x                      String to check and convert  
variable\_name        Name of the variable to print if error occurs

**Value**

x as double, or NULL if is null and ERROR if x isn't convertible to a double

**Examples**

```
validate_double(NULL) # NULL  
try(validate_double(" ")) # ERROR  
try(validate_double("Hello World !")) # ERROR  
validate_double("12.4") # 12.4  
validate_double("12") # 12.0
```

---

validate_file	<i>Check if value is an existing file</i>
---------------	---

---

**Description**

Check if value is an existing file

**Usage**

```
validate_file(x, variable_name = NULL)
```

**Arguments**

x                      String to check  
variable\_name        Name of the variable to print if error occurs

**Value**

x, or NULL if is null and ERROR if x isn't an existing file

**Examples**

```
try(validate_file("test")) # ERROR
```

---

validate_folder	<i>Check if value is an existing folder</i>
-----------------	---

---

**Description**

Check if value is an existing folder

**Usage**

```
validate_folder(x, variable_name = NULL)
```

**Arguments**

x                      String to check  
variable\_name        Name of the variable to print if error occurs

**Value**

x, or NULL if is null and ERROR if x isn't an existing folder

**Examples**

```
try(validate_folder("test")) # ERROR
```

---

validate_integer	<i>Check if value is an integer</i>
------------------	-------------------------------------

---

**Description**

Check if value is an integer

**Usage**

```
validate_integer(x, variable_name = NULL)
```

**Arguments**

x                      String to check and convert  
variable\_name        Name of the variable to print if error occurs

**Value**

x as integer, or NULL if is null and ERROR if x isn't convertible to an integer

**Examples**

```
validate_integer(NULL) # NULL
try(validate_integer(" ")) # ERROR
try(validate_integer("Hello World !")) # ERROR
try(validate_integer("12.4")) # ERROR
validate_integer("12") # 12
```

---

**valid\_string***Check for Non-Empty, Non-Whitespace String*

---

**Description**

This function checks if the input is non-NULL and contains more than just whitespace.

**Usage**

```
valid_string(input)
```

**Arguments**

input            A variable to check.

**Value**

input with whitespace trimmed or throw error if not a valid string

**Examples**

```
valid_string("Hello World") # Returns "Hello World"
valid_string(" Hello World ") # Returns "Hello World"
try(valid_string(" ")) # Error
try(valid_string(NULL)) # Error
```

# Index

`create_log_session_info`, [3](#)  
`create_versions_yaml`, [4](#)

`is_null_or_invalid`, [5](#)  
`is_valid_string`, [5](#)

`nfcore.utils` (`nfcore.utils`-package), [2](#)  
`nfcore.utils`-package, [2](#)  
`nullify`, [6](#)

`parse_arguments`, [6](#)  
`process_end`, [7](#)  
`process_end()`, [2](#)  
`process_inputs`, [8](#)  
`process_inputs()`, [2](#)

`read_delim_flexible`, [9](#)

`scan`, [10](#)

`utils::read.delim`, [9](#)

`valid_string`, [13](#)  
`validate_boolean`, [10](#)  
`validate_double`, [11](#)  
`validate_file`, [11](#)  
`validate_folder`, [12](#)  
`validate_integer`, [12](#)