

Package ‘porter’

June 30, 2026

Title Generate Port Files for C Libraries

Version 0.1.0

Description Generate port files for foreign function interfaces to C libraries by parsing C-family header files with 'CastXML'.

License MIT + file LICENSE

URL <https://github.com/hongyuanjia/porter>

BugReports <https://github.com/hongyuanjia/porter/issues>

SystemRequirements CastXML

Encoding UTF-8

RoxygenNote 7.3.3

Depends R (>= 3.6.0)

Imports xml2

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

Collate 'utils.R' 'cache.R' 'castxml.R' 'check.R' 'field.R' 'port.R' 'porter-package.R' 'signature.R' 'write.R' 'zzz.R'

NeedsCompilation no

Author Hongyuan Jia [aut, cre, cph] (ORCID: <https://orcid.org/0000-0002-0075-8183>)

Maintainer Hongyuan Jia <hongyuanjia@cqust.edu.cn>

Repository CRAN

Date/Publication 2026-06-30 10:30:08 UTC

Contents

locate_castxml	2
port	3
port_fields	4
port_report	5

port_validate_symbols	6
port_write	7
print.dynport	7

Index	9
--------------	----------

locate_castxml	<i>Locate a CastXML executable</i>
----------------	------------------------------------

Description

locate_castxml() finds a CastXML executable installed by the user. porter does not download or install CastXML; install it with your system package manager before calling port().

Usage

```
locate_castxml(path = getOption("porter.castxml"))
```

Arguments

path	Optional path to either the CastXML executable or a directory containing it. Defaults to getOption("porter.castxml").
------	-----------------------------------------------------------------------------------------------------------------------

Details

When path is supplied, only that executable or directory is checked. With the default NULL path, the search order is:

- Sys.which("castxml").
- Common package-manager locations that may not be on PATH, including Homebrew, Linuxbrew, Scoop, Chocolatey, and conda prefixes.

Package-manager installations are usually found by Sys.which("castxml") when their shims or binary directories are on PATH.

Value

A single named character vector. The value is the normalized path to the CastXML executable, and the name is the detected version. Returns NULL if no usable CastXML executable is found.

Examples

```
locate_castxml()
```

port	<i>Parse C-family header files using CastXML and gather data from the generated XML.</i>
------	------------------------------------------------------------------------------------------

Description

Parse C-family header files using CastXML and gather data from the generated XML.

Usage

```
port(
  header,
  limit = TRUE,
  keep = FALSE,
  cflags = NULL,
  castxml = locate_castxml()
)
```

Arguments

header	A single string indicating the path of C header file.
limit	A flag (TRUE/FALSE) a string, or a character vector. If TRUE, only functions/enums/structs/unions from the directory of the input header are kept. If FALSE, no subsetting is performed. If a single string, it will be used as a regular expression to extract only functions/enums/structs/unions that match the pattern. If a character vector, it should be paths of directories used to constrain the scope. Default: TRUE.
keep	Either TRUE or FALSE. If TRUE, the XML file generated by CastXML is saved into the same directory of input header file with the same name. Default: FALSE.
cflags	A character vector of C compile flags passed to CastXML. Default: NULL.
castxml	A single string indicating the file path of the CastXML executable. Default uses locate_castxml() .

Value

An dynport object.

Note

Please note that function-like macros are not converted to callable entries. They are reported by `port_report()` as unsupported macros. See: <https://github.com/CastXML/CastXML/issues/21>

Examples

```
header <- tempfile(fileext = ".h")
writeLines(c(
  "typedef struct Point { int x; double y; } Point;",
  "int add(int a, int b);"
```

```
), header)
port(header)
```

port_fields *Query and update dynport fields*

Description

Query and update dynport fields

Usage

```
port_fields(port)
port_has(port, fields)
port_get(port, fields)
port_set(port, ...)
```

Arguments

port	A dynport object created using port .
fields	A character vector of valid field names in input dynport object.
...	Pairs of field names and values. See Details.

Details

port_fields() lists the fields in the object.

port_has() checks if one or multiple fields are present in the object.

port_get() returns the current values of input fields in the object. If multiple fields are given, a named list is returned. Note that port_get() errors if non-existing fields are queried.

port_set() updates one or multiple field values in the object. It accepts two forms of arguments:

- Two arguments with the first element being the name of a single field, and the second being the field value. Use NULL value to remove the field.
- Multiple named arguments with names being names of field to set, and values being the new values. Use NULL value to remove the field.

For port_set(), you can use NULL to remove values of input fields. For example, the following two lines of code both remove the value of the Version field.

```
port_set(p, "Version", NULL)
port_set(p, Version = NULL)
```

Note that, following fields are always required for a dynport object and will be kept as NULL instead of removal: Package, Version, Library, Function, FuncPtr, Enum, Struct and Union. Other fields are treated as user customize fields and will be removed when setting to NULL.'

Value

port_fields() returns a character vector.

port_has() returns a logical vector.

port_get() returns a character vector or a data.frame, depending on the field value types.

port_set() returns a new dynport object, invisibly.

Examples

```
header <- tempfile(fileext = ".h")
writeLines("int add(int a, int b);", header)

p <- port(header)
port_fields(p)
port_has(p, c("Package", "Function"))
port_get(p, "Function")
port_set(p, Package = "Example", Version = "1.0", Library = "example")
```

port_report

Inspect diagnostics recorded while generating a dynport

Description

Inspect diagnostics recorded while generating a dynport

Usage

```
port_report(port, kind = NULL)
```

Arguments

port	A dynport object created by <code>port()</code> .
kind	Optional diagnostic kind to filter by.

Value

A data frame with diagnostic entries, including source file and line when available.

Examples

```
header <- tempfile(fileext = ".h")
writeLines(c(
  "#define ADD_ONE(x) ((x) + 1)",
  "int add(int a, int b);"
), header)

p <- port(header)
port_report(p)
port_report(p, "unsupported_macro")
```

port_validate_symbols *Validate generated function symbols against an exported symbol list*

Description

Validate generated function symbols against an exported symbol list

Usage

```
port_validate_symbols(port, symbols, ...)
```

Arguments

port	A dynport object created by <code>port()</code> .
symbols	A character vector of exported symbol names to compare against the generated Function and Variadic entries.
...	Reserved for future extensions.

Value

A data frame with symbol validation status.

Examples

```
header <- tempfile(fileext = ".h")
writeLines("int add(int a, int b);", header)

p <- port(header)
port_validate_symbols(p, c("add", "other_symbol"))
```

port_write	<i>Write dynport file</i>
------------	---------------------------

Description

Write dynport file

Usage

```
port_write(port, file, reorder = TRUE)
```

Arguments

port	A dynport object
file	A single string or a connection object.
reorder	Either TRUE or FALSE. If TRUE, required dynport fields are always written before all other customized fields. Default: TRUE.

Value

A single string of the file path, invisibly.

Examples

```
header <- tempfile(fileext = ".h")
writeLines("int add(int a, int b);", header)

p <- port(header)
p <- port_set(p, Package = "Example", Version = "1.0", Library = "example")
port_write(p, file.path(tempdir(), "Example.dynport"))
```

print.dynport	<i>Print summary of an dynport object</i>
---------------	-------------------------------------------

Description

Print summary of an dynport object

Usage

```
## S3 method for class 'dynport'
print(x, n = 5L, ...)
```

Arguments

x	A dynport.
n	The number of items to print. Default: 5.
...	Other arguments to pass. Reserved for future use.

Value

x invisibly.

Examples

```
header <- tempfile(fileext = ".h")
writeLines("int add(int a, int b);", header)
```

```
p <- port(header)
print(p)
```

Index

locate_castxml, 2
locate_castxml(), 3

port, 3, 4
port(), 2, 5, 6
port_fields, 4
port_get (port_fields), 4
port_has (port_fields), 4
port_report, 5
port_set (port_fields), 4
port_validate_symbols, 6
port_write, 7
print.dynport, 7