Package 'import'

May 8, 2026

Type Package

Title An Import Mechanism for R

Version 1.3.4

Description Alternative mechanism for importing objects from packages and R modules. The syntax allows for importing multiple objects with a single command in an expressive way. The import package bridges some of the gap between using library (or require) and direct (single-object) imports. Furthermore the imported objects are not placed in the current environment.

License MIT + file LICENSE

ByteCompile TRUE

URL https://rticulate.github.io/import/, https://github.com/rticulate/import

BugReports https://github.com/rticulate/import/issues

Suggests knitr, rmarkdown, magrittr, testthat

Language en-US

VignetteBuilder knitr

RoxygenNote 7.3.3

Encoding UTF-8

NeedsCompilation no

Author Stefan Milton Bache [aut], Magnus Thor Torfason [aut, cre]

Maintainer Magnus Thor Torfason <m@zulutime.net>

Repository CRAN

Date/Publication 2025-10-19 12:30:02 UTC

Contents

`from` *Import Objects From a Package.*

Description

The `import::from()`, `import::into()`, and `import::here()` functions provide an alternative way to import objects (e.g. functions) from packages or modules (see below). It is sometimes preferred over using `library` (or `require`) which will import all objects exported by the package. The benefit over `obj <- pkg::obj` is that the imported objects will (by default) be placed in a separate entry in the search path (which can be specified), rather in the global/current environment. Also, it is a more succinct way of importing several objects.

`import::from()` and `import::into()` are symmetric, and usage is a matter of preference and whether specifying the `.into` argument is desired. `import::here()` is a shorthand that always imports into the current environment, and `import::what()` provides a way to quickly list all objects in a package or module that are available for import by the other functions.

Usage

```
from(
  .from,
  ...,
  .into = "imports",
  .library = .libPaths(),
  .directory = ".",
  .all = (length(.except) > 0),
  .except = character(),
  .chdir = TRUE,
  .character_only = FALSE,
  .S3 = FALSE
)

here(
  .from,
  ...,
  .library = .libPaths()[1L],
  .directory = ".",
  .all = (length(.except) > 0),
  .except = character(),
  .chdir = TRUE,
  .character_only = FALSE,
  .S3 = FALSE
)

into(
  .into,
  ...,
```

```
  .from,
  .library = .libPaths()[1L],
  .directory = ".",
  .all = (length(.except) > 0),
  .except = character(),
  .chdir = TRUE,
  .character_only = FALSE,
  .S3 = FALSE
)

what(
  .from,
  ...,
  .library = .libPaths()[1L],
  .directory = ".",
  .chdir = TRUE,
  .character_only = FALSE,
  .S3 = FALSE
)
```

Arguments

`.from`	The package from which to import.
`...`	Names or name-value pairs specifying objects to import. If arguments are named, then the imported object will have this new name.
`.into`	The environment into which the imported objects should be assigned. If the value is of mode `character`, it is treated as referring to a named environment on the search path. If it is of mode `environment`, the objects are assigned directly to that environment. Using `.into=environment()` causes imports to be made into the current environment; `.into=""` is an equivalent shorthand value.
`.library`	character specifying the library to use when importing from packages. Defaults to the current set of library paths (note that the default value was different in versions up to and including `1.3.0`).
`.directory`	character specifying the directory to use when importing from modules. Defaults to the current working directory. If .from is a module specified using an absolute path (i.e. starting with `/`), this parameter is ignored.
`.all`	logical specifying whether all available objects in a package or module should be imported. It defaults to FALSE unless .exclude is being used to omit particular functions.
`.except`	character vector specifying any objects that should not be imported. Any values specified here override both values provided in `...` and objects included because of the `.all` parameter
`.chdir`	logical specifying whether to change directories before sourcing a module (this parameter is ignored for libraries)
`.character_only`	A logical indicating whether `.from` and `...` can be assumed to be character

strings. (Note that this parameter does not apply to how the `.into` parameter is handled).

`.S3` **[Experimental]** A logical indicating whether an automatic detection and registration of S3 methods should be performed. The S3 methods are assumed to be in the standard form `generic.class`. Methods can also be registered manually instead using be registered manually instead using the `.S3method(generic, class, method)` call. *This is an experimental feature. We think it should work well and you are encouraged to use it and report back – but the syntax and semantics may change in the future to improve the feature.*

Details

The function arguments can be quoted or unquoted as with e.g. `library`. In any case, the character representation is used when unquoted arguments are provided (and not the value of objects with matching names). The period in the argument names `.into` and `.from` are there to avoid name clash with package objects. However, while importing of hidden objects (those with names prefixed by a period) is supported, care should be taken not to conflict with the argument names. The double-colon syntax `import::from` allows for imports of exported objects (and lazy data) only. To import objects that are not exported, use triple-colon syntax, e.g. `import:::from`. The two ways of calling the `import` functions analogue the `::` and `:::` operators themselves.

Note that the `import` functions usually have the (intended) side-effect of altering the search path, as they (by default) import objects into the "imports" search path entry rather than the global environment.

The `import` package is not meant to be loaded with `library` (and will output a message about this if attached), but rather it is named to make the function calls expressive without the need to loading before use, i.e. it is designed to be used explicitly with the `::` syntax, e.g. `import::from(pkg, x, y)`.

Value

a reference to the environment containing the imported objects.

Packages vs. modules

`import` can either be used to import objects either from R packages or from `R` source files. If the `.from` parameter ends with '.R' or '.r', `import` will look for a source file to import from. A source file in this context is referred to as a `module` in the documentation.

Package Versions

With `import` you can specify package version requirements. To do this add a requirement in parentheses to the package name (which then needs to be quoted), e.g `import::from("parallel (>= 3.2.0)", ...)`. You can use the operators <, >, <=, >=, ==, !=. Whitespace in the specification is irrelevant.

See Also

Helpful links:

- https://rticulate.github.io/import/
- https://github.com/rticulate/import/
- https://github.com/rticulate/import/issues/

Examples

```
import::from(parallel, makeCluster, parLapply)
import::into("imports:parallel", makeCluster, parLapply, .from = parallel)
import::here(parallel, detectCores)
import::what(parallel)
```

`import` *An Import Mechanism for R*

Description

This is an alternative mechanism for importing objects from packages. The syntax allows for importing multiple objects from a package with a single command in an expressive way. The `import` package bridges some of the gap between using `library` (or `require`) and direct (single-object) imports. Furthermore the imported objects are not placed in the current environment (although possible), but in a named entry in the search path.

Details

This package is not intended for use with `library`. It is named to make calls like `import::from(pkg, fun1, fun2)` expressive. Using the `import` functions complements the standard use of `library(pkg)`(when most objects are needed, and context is clear) and `obj <- pkg::obj` (when only a single object is needed).

Author(s)

Stefan Milton Bache

See Also

For usage instructions and examples, see `from`, `into`, or `here`.

Helpful links:

- https://rticulate.github.io/import/
- https://github.com/rticulate/import/
- https://github.com/rticulate/import/issues/

Index