Package: marshal (via r-universe)

November 5, 2026

Version 0.0.1-9006

Title Framework to Marshal Objects to be Used in Another R Process

Depends R (>= 3.2.0)

Suggests knitr, DT, jsonlite, bigmemory, bundle, caret, data.table, DBI, RSQLite, digest, h2o, inline, keras, tensorflow, magick, ncdf4, parsnip, raster, reticulate, rJava, rstan, BH, RcppEigen, sparklyr, stats, terra, tools, torch, luz, udpipe, xgboost, XML, xml2

VignetteBuilder knitr

Description Some types of R objects can be used only in the R session they were created. If used as-is in another R process, such objects often result in an immediate error or in obscure and hard-to-troubleshoot outcomes. Because of this, they cannot be saved to file and re-used at a later time. They can also not be exported to a worker in parallel processing. These objects are sometimes referred to as non-exportable or non-serializable objects. One solution to this problem is to use ``marshalling'' to encode the R object into an exportable representation that then can be used to re-create a copy of that object in another R process. This package provides a framework for marshalling and unmarshalling R objects such that they can be transferred using functions such as serialize() and unserialize() of base R.

License MIT + file LICENSE

Encoding UTF-8

LazyLoad TRUE

ByteCompile TRUE

URL https://marshal.futureverse.org, https://github.com/futureverse/marshal

BugReports https://github.com/futureverse/marshal/issues

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Repository https://futureverse-devel.r-universe.dev

Date/Publication 2024-06-06 21:22:26 UTC

RemoteUrl https://github.com/futureverse/marshal

RemoteRef develop

RemoteSha 7fefeee14158ac37bd95f5eae80cde11c3f40589

Contents

marshal-package . . . 3
marshal . . . 3
marshal.big.matrix . . . 4
marshal.CFunc . . . 5
marshal.connection . . . 6
marshal.data.table . . . 8
marshal.DBIConnection . . . 9
marshal.default . . . 9
marshal.H2OAutoML . . . 10
marshal.jclassName . . . 11
marshal.keras.engine.base_layer.Layer . . . 12
marshal.luz_module_fitted . . . 13
marshal.magick-image . . . 14
marshal.model_fit . . . 14
marshal.ncdf4 . . . 15
marshal.python.builtin.object . . . 17
marshal.RasterLayer . . . 17
marshal.SOCKcluster . . . 18
marshal.SpatVector . . . 19
marshal.stanfit . . . 20
marshal.tbl_spark . . . 21
marshal.train . . . 21
marshal.udpipe_model . . . 22
marshal.xgb.DMatrix . . . 22
marshal.xml_document . . . 23
marshal.XMLAbstractNode . . . 24
marshallable . . . 25

Index **27**

`marshal-package` *Package marshal - Marshal Objects to be Used in Another R Process*

Description

Some types of R objects can be used only in the R session they were created. If used as-is in another R process, such objects often result in an immediate error or in obscure and hard-to-troubleshoot outcomes. Because of this, they cannot be saved to file and re-used at a later time. They can also not be exported to a worker in parallel processing. These objects are sometimes referred to as non-exportable or non-serializable objects. One solution to this problem is to use "marshalling" to encode the R object into an exportable representation that then can be used to re-create a copy of that object in another R process. This package provides a framework for marshalling and unmarshalling R objects such that they can be transferred using functions such as `serialize()` and `unserialize()` of base R.

Author(s)

Maintainer: Henrik Bengtsson <henrikb@braju.com> [copyright holder]

See Also

Useful links:

- https://marshal.futureverse.org
- https://github.com/futureverse/marshal
- Report bugs at https://github.com/futureverse/marshal/issues

`marshal` *Marshalling of R objects*

Description

Marshalling of R objects

Usage

```
marshal(...)

unmarshal(...)
```

Arguments

`...`	The object to be marshalled, or unmarshalled, followed by additional arguments passed to the specific S3 method.

Value

`marshal()` returns a `marshalled` object, which is a list with components:

- `marshalled`: marshalled version of the original object
- `unmarshal`: function that takes the `marshalled` object as input and returns an unmarshalled version of the original object.

`unmarshal()` returns an unmarshalled version of the original object.

`marshal.big.matrix` *Marshalling of 'bigmemory' objects*

Description

Marshalling of 'bigmemory' objects

Usage

```
## S3 method for class 'big.matrix'
marshal(x, ...)

## S3 method for class 'big.memory'
marshallable(...)
```

Arguments

`x`	A bigmemory::big.matrix object.
`...`	Not used.

Details

`bigmemory::write.big.matrix()` is used to produce a marshalled version of the original object. `bigmemory::read.big.matrix()` is used to reconstruct a version of the original object from the marshalled object.

Value

A `marshalled` object as described in `marshal()`.

Examples

```
if (requireNamespace("bigmemory", quietly = TRUE)) {
  library(bigmemory)

  x <- big.matrix(nrow = 3, ncol = 2, type = "double")
  x[] <- seq_along(x)

  ## Marshal
```

```
  x_ <- marshal(x)

  ## Unmarshal
  x2 <- unmarshal(x_)

  stopifnot(identical(x2[], x[]))
}
```

`marshal.CFunc` *Marshalling of 'inline' objects*

Description

WARNING: This is very limited proof of concept!

Usage

```
## S3 method for class 'CFunc'
marshal(inline, ...)

## S3 method for class 'CFunc'
marshallable(...)
```

Arguments

`inline`	A CFunc function.
`...`	Not used.

Details

Currently, it is only possible to marshal a function:

- of class `CFunc` that was created *without* "includes" or "otherdefs"

Value

A `marshalled` object as described in `marshal()`.

Examples

```
if (requireNamespace("inline", quietly = TRUE)) {
  code <- "
    int i;
    for (i = 0; i < *n; i++) x[0] = x[0] + (i+1);
  "
  sum_1_to_n <- inline::cfunction(
    signature(n = "integer", x = "numeric"),
    code,
```

```
  language = "C", convention = ".C"
)

## Marshal CFunc function
sum_1_to_n_ <- marshal(sum_1_to_n)

## Unarshal CFunc function
sum_1_to_n2 <- unmarshal(sum_1_to_n_)

y <- sum_1_to_n(10, 0)$x
print(y)

y2 <- sum_1_to_n2(10, 0)$x
print(y2)
}
```

`marshal.connection` *Marshalling of R connections*

Description

Marshalling of R connections

Usage

```
## S3 method for class 'connection'
marshal(con, ...)

## S3 method for class 'connection'
marshallable(con, ...)
```

Arguments

`con`	A connection.
`...`	Not used.

Value

A `marshalled` object as described in `marshal()`.

Limitations

Not all connections can be marshalled, specifically we:

- cannot marshal connections stdin (0), stdout (1), and stderr (2)
- can only marshal *read-only* connections
- can only marshal *unopened* or *seekable* connections

Examples

```
tf <- tempfile()
cat(file = tf, letters, sep = "")

## Read-only connection
con <- file(tf, open = "rb")

bfr <- readChar(con, nchars = 4L)
print(bfr)  ## "abcd"

## Marshal read-only connection, which records the
## current state, including the current file position.
con_ <- marshal(con)

## Unmarshal connection, which restores the state
## of the original connection, including the current
## file position
con2 <- unmarshal(con_)
stopifnot(
  all.equal(summary(con2), summary(con)),
  identical(seek(con2), seek(con))
)

bfr <- readChar(con, nchars = 4L)
print(bfr)
bfr2 <- readChar(con2, nchars = 4L)
print(bfr2)
stopifnot(identical(bfr2, bfr))

## Cleanup
close(con)
close(con2)
file.remove(tf)
con <- url("https://www.r-project.org")
print(con)

## Marshal read-only connection, which records the
## current state, including the current file position.
con_ <- marshal(con)

## Unmarshal connection, which restores the state
## of the original connection
con2 <- unmarshal(con_)
print(con2)

stopifnot(all.equal(summary(con2), summary(con)))

bfr <- readChar(con, nchars = 100L)
print(bfr)
bfr2 <- readChar(con2, nchars = 100L)
print(bfr2)
stopifnot(identical(bfr2, bfr))
```

```
## Cleanup
close(con)
close(con2)
```

`marshal.data.table` *Marshalling of 'data.table' objects*

Description

Marshalling of 'data.table' objects

Usage

```
## S3 method for class 'data.table'
marshal(x, ...)

## S3 method for class 'data.table'
marshallable(...)
```

Arguments

`x`	A data.table::data.table object.
`...`	Not used.

Value

A `marshalled` object as described in `marshal()`.

Examples

```
if (requireNamespace("data.table", quietly = TRUE)) {
  dt <- data.table::data.table(a = 1:3, b = letters[1:3])

  ## Marshal
  dt_ <- marshal(dt)

  ## Unmarshal
  dt2 <- unmarshal(dt_)

  stopifnot(identical(dt2, dt))

  ## Marshal and unmarshal again
  dt2_ <- marshal(dt2)
  dt3 <- unmarshal(dt2_)

  stopifnot(
    identical(dt2_, dt_),
```

```
    identical(dt3, dt2)
  )
}
```

`marshal.DBIConnection` *Marshalling of 'DBI:DBIConnection' objects (not supported)*

Description

Warning: Objects of this class are not possible to marshal. If attempted, an error is produced.

Usage

```
## S3 method for class 'DBIConnection'
marshal(x, ...)

## S3 method for class 'DBIConnection'
marshallable(...)
```

Arguments

`x`	A DBI::DBIConnection object.
`...`	Not used.

Value

A `marshalled` object as described in `marshal()`.

`marshal.default` *Marshalling of an R objects*

Description

Marshalling of an R objects

Usage

```
## Default S3 method:
marshal(x, ...)
```

Arguments

`x`	An R object.
`...`	Not used.

Details

This is the default, fallback method for marshalling of an R object. It assumes that the object do not need specific marshalling, but can be serialized as-is. All this default method does is wrapping the original object up in a `marshalled` object, so that the original object can be recovered via `unmarshalled()`.

Value

A `marshalled` object as described in `marshal()`.

Examples

```
x <- data.frame(a = 1:3, b = letters[1:3])
x_ <- marshal(x)
x2 <- unmarshal(x_)
stopifnot(identical(x2, x))
```

`marshal.H2OAutoML` *Marshalling of 'h2o' objects*

Description

Marshalling of 'h2o' objects

Usage

```
## S3 method for class 'H2OAutoML'
marshal(x, ...)

## S3 method for class 'H2OMultinomialModel'
marshal(x, ...)

## S3 method for class 'H2OBinomialModel'
marshal(x, ...)

## S3 method for class 'H2ORegressionModel'
marshal(x, ...)

## S3 method for class 'H2OAutoML'
marshallable(...)

## S3 method for class 'H2OMultinomialModel'
marshallable(...)

## S3 method for class 'H2OBinomialModel'
```

```
marshallable(...)

## S3 method for class 'H2ORegressionModel'
marshallable(...)
```

Arguments

x	An "h2o" object.
...	Not used.

Details

h2o::h2o.save_mojo() and h2o::h2o.saveModel() are used to produce a marshalled version of the original object. h2o::h2o.import_mojo() and h2o::h2o.loadModel() are used to reconstruct a version of the original object from the marshalled object.

Value

A marshalled object as described in marshal().

---

marshal.jclassName *Marshalling of 'rJava' objects (not supported)*

Description

Warning: Objects of this class are not possible to marshal. If attempted, an error is produced.

Usage

```
## S3 method for class 'jclassName'
marshal(x, ...)

## S3 method for class 'jobjRef'
marshal(x, ...)

## S3 method for class 'jclassName'
marshallable(...)

## S3 method for class 'jobjRef'
marshallable(...)
```

Arguments

x	"rJava" object.
...	Not used.

Value

A marshalled object as described in marshal().

`marshal.keras.engine.base_layer.Layer` *Marshalling of 'keras' objects*

Description

Marshalling of 'keras' objects

Usage

```
## S3 method for class 'keras.engine.base_layer.Layer'
marshal(model, ...)

## S3 method for class 'keras.engine.base_layer.Layer'
marshallable(...)
```

Arguments

`model`	A keras:keras.engine.base_layer.Layer object.
`...`	Not used.

Details

`keras::serialize_model()` is used to produce a marshalled version of the original object. `keras::unserialize_model()` is used to reconstruct a version of the original object from the marshalled object.

Value

A `marshalled` object as described in `marshal()`.

Examples

```
## Run only in interactive mode, because example takes > 5 seconds,
## which is longer than what is allowed on CRAN
if (interactive() && requireNamespace("keras", quietly = TRUE)) {
  library(keras)

  ## Create a keras model (adopted from {keras} vignette)
  inputs <- layer_input(shape = shape(32))
  outputs <- layer_dense(inputs, units = 1L)
  model <- keras_model(inputs, outputs)
  model <- compile(model, optimizer = "adam", loss = "mean_squared_error")
  print(model)

  ## Not needed anymore
  rm(list = c("inputs", "outputs"))

  ## Marshal
  model_ <- marshal(model)
```

```
  ## Unmarshal
  model2 <- unmarshal(model_)

  stopifnot(
    identical(summary(model2), summary(model))
  )


  ## Fitted keras model (adopted from {keras} vignette)
  test_input <- array(runif(128 * 32), dim = c(128, 32))
  test_target <- array(runif(128), dim = c(128, 1))
  hist <- fit(model, test_input, test_target)
  print(hist)
  print(model)

  ## Not needed anymore
  rm(list = "test_target")

  ## Marshal
  model_ <- marshal(model)

  ## Unmarshal
  model2 <- unmarshal(model_)

  stopifnot(
    identical(summary(model2), summary(model)),
    identical(stats::predict(model2, test_input), stats::predict(model, test_input))
  )
}
```

`marshal.luz_module_fitted` *Marshalling of 'torch' objects*

Description

Marshalling of 'torch' objects

Usage

```
## S3 method for class 'luz_module_fitted'
marshal(model, ...)

## S3 method for class 'luz_module_fitted'
marshallable(...)
```

Arguments

`model`	A `luz_module_fitted` object.
`...`	Not used.

Details

luz::luz_save() is used to produce a marshalled version of the original object. luz::luz_load() is used to reconstruct a version of the original object from the marshalled object.

Value

A marshalled object as described in marshal().

---

marshal.magick-image *Marshalling of 'magick' objects (not supported)*

Description

Warning: Objects of this class are not possible to marshal. If attempted, an error is produced.

Usage

```
## S3 method for class '`magick-image`'
marshal(x, ...)

## S3 method for class '`magick-image`'
marshallable(...)
```

Arguments

x	A magick:magick-image object.
...	Not used.

Value

A marshalled object as described in marshal().

---

marshal.model_fit *Marshalling of 'parsnip' objects*

Description

Marshalling of 'parsnip' objects

Usage

```
## S3 method for class 'model_fit'
marshal(x, ...)

## S3 method for class 'model_fit'
marshallable(...)
```

Arguments

x	A parnsip:model_fit object.
...	Not used.

Details

The fit element of the model_fit object is marshalled.

Value

A marshalled object as described in marshal().

Examples

```
if (requireNamespace("parsnip", quietly = TRUE) && requireNamespace("xgboost", quietly = TRUE)) {
  library(parsnip)

  ## Adopted from example("boost_tree", package = "parsnip")
  model <- boost_tree(mode = "classification", trees = 20L, engine = "xgboost")
  model <- set_mode(model, "regression")
  fit <- fit(model, mpg ~ ., data = datasets::mtcars)

  ## Marshal
  fit_ <- marshal(fit)

  ## Unmarshal
  fit2 <- unmarshal(fit_)

  ## Marshal again
  fit2_ <- marshal(fit2)

  ## Assert identity
  stopifnot(
    all.equal(fit2_, fit_)
  )

  fit3 <- unmarshal(fit2_)

  ## Assert identity
  stopifnot(
    all.equal(fit3, fit2)
  )
}
```

---

marshal.ncdf4 *Marshalling of 'ncdf4' objects*

Description

Marshalling of 'ncdf4' objects

Usage

```
## S3 method for class 'ncdf4'
marshal(x, ...)

## S3 method for class 'ncdf4'
marshallable(...)
```

Arguments

x	A ncdf4:ncdf4 object.
...	Not used.

Details

base::readBin() is used to produce a marshalled version of the original object. base::writeBin() and ncdf4::nc_open() are used to reconstruct a version of the original object from the marshalled object.

Value

A marshalled object as described in marshal().

Examples

```
if (requireNamespace("ncdf4", quietly = TRUE)) {
  library(ncdf4)

  ## Create a temporary NCDF file
  tf <- tempfile(fileext = ".nc")

  x <- ncvar_def("x", units = "count", dim = list())
  nf <- nc_create(tf, x)
  ncvar_put(nf, x, 42)
  print(nf)
  nc_close(nf)
  rm(list = c("nf", "x"))

  ## Open NCDF file
  nf <- nc_open(tf)
  y <- ncvar_get(nf, "x")

  ## Marshal
  nf_ <- marshal(nf)

  ## Unmarshal
  nf2 <- unmarshal(nf_)

  y2 <- ncvar_get(nf2, "x")
  stopifnot(identical(y2, y))
}
```

marshal.python.builtin.object *Marshalling of 'reticulate' objects (not supported)*

Description

Warning: Objects of this class are not possible to marshal. If attempted, an error is produced.

Usage

```
## S3 method for class 'python.builtin.object'
marshal(x, ...)

## S3 method for class 'python.builtin.object'
marshallable(...)
```

Arguments

x	"reticulate" object.
...	Not used.

Value

A marshalled object as described in marshal().

marshal.RasterLayer *Marshalling of 'raster' objects*

Description

Marshalling of 'raster' objects

Usage

```
## S3 method for class 'RasterLayer'
marshal(x, ...)

## S3 method for class 'RasterLayer'
marshallable(x, ...)
```

Arguments

x	A raster:RasterLayer object.
...	Not used.

Details

raster::writeRaster() is used to produce a marshalled version of the original object. raster::raster() is used to reconstruct a version of the original object from the marshalled object.

Value

A marshalled object as described in marshal().

WARNING - Numerical identical results cannot be guaranteed

Marshalling of RasterLayer objects is *leaky*. More precisely, the method *cannot* guarantee that multiple rounds of marshalling and unmarshalling produce numerically identical objects. For further information, see (1).

References

1. Issue #311 - 'Incorrect large values saved to disk with writeRaster()', **raster** GitHub repostory, 2023-05-17, https://github.com/rspatial/raster/issues/311.

Examples

```
if (requireNamespace("raster", quietly = TRUE)) {
  r <- raster::raster(system.file("external/test.grd", package = "raster"))
  print(r)

  ## Marshal
  r_ <- marshal(r)

  ## Unmarshal
  r2 <- unmarshal(r_)
  print(r2)
}
```

---

marshal.SOCKcluster *Marshalling of 'parallel' objects (not supported)*

Description

Warning: Objects of this class are not possible to marshal. If attempted, an error is produced.

Usage

```
## S3 method for class 'SOCKcluster'
marshal(x, ...)

## S3 method for class 'SOCKnode'
marshal(x, ...)
```

```
## S3 method for class 'SOCK0node'
marshal(x, ...)

## S3 method for class 'SOCKcluster'
marshallable(...)

## S3 method for class 'SOCKnode'
marshallable(...)

## S3 method for class 'SOCK0node'
marshallable(...)
```

Arguments

x	A magick:magick-image object.
...	Not used.

Value

A marshalled object as described in marshal().

---

marshal.SpatVector *Marshalling of 'terra' objects*

Description

Marshalling of 'terra' objects

Usage

```
## S3 method for class 'SpatVector'
marshal(terra, ...)

## S3 method for class 'SpatVector'
marshallable(...)
```

Arguments

terra	An terra::SpatVector.
...	Not used.

Details

terra::wrap() is used to produce a marshalled version of the original object. terra::vect() is used to reconstruct a version of the original object from the marshalled object.

Value

A marshalled object as described in marshal().

Examples

```
if (requireNamespace("terra", quietly = TRUE)) {
  file <- system.file("ex/lux.shp", package = "terra")
  v <- terra::vect(file)

  ## Marshal SpatVector object
  v_ <- marshal(v)

  ## Unmarshal SpatVector object
  v2 <- unmarshal(v_)

  stopifnot(all.equal(v2, v, check.attributes = FALSE))
}
```

---

marshal.stanfit *Marshalling of 'rstan' objects*

Description

Marshalling of 'rstan' objects

Usage

```
## S3 method for class 'stanfit'
marshal(x, ...)

## S3 method for class 'stanfit'
marshallable(...)
```

Arguments

x	A rstan:stanfit object.
...	Not used.

Value

A marshalled object as described in marshal().

marshal.tbl_spark *Marshalling of 'sparklyr' objects (not supported)*

Description

Warning: Objects of this class are not possible to marshal. If attempted, an error is produced.

Usage

```
## S3 method for class 'tbl_spark'
marshal(x, ...)

## S3 method for class 'tbl_spark'
marshallable(...)
```

Arguments

x	"sparklyr" object.
...	Not used.

Value

A marshalled object as described in marshal().

marshal.train *Marshalling of 'caret:train' objects*

Description

Marshalling of 'caret:train' objects

Usage

```
## S3 method for class 'train'
marshal(train, ...)

## S3 method for class 'train'
marshallable(...)
```

Arguments

train	A caret::train object.
...	Not used.

Details

`bundle::bundle()` is used to produce a marshalled version of the original object. `bundle::unbundle()` is used to reconstruct a version of the original object from the marshalled object.

Value

A `marshalled` object as described in `marshal()`.

`marshal.udpipe_model` *Marshalling of 'udpipe' objects (not supported)*

Description

Warning: Objects of this class are not possible to marshal. If attempted, an error is produced.

Usage

```
## S3 method for class 'udpipe_model'
marshal(x, ...)

## S3 method for class 'udpipe_model'
marshallable(...)
```

Arguments

`x`	"udpipe" object.
`...`	Not used.

Value

A `marshalled` object as described in `marshal()`.

`marshal.xgb.DMatrix` *Marshalling of 'xgboost' objects*

Description

Marshalling of 'xgboost' objects

Usage

```
## S3 method for class 'xgb.DMatrix'
marshal(x, ...)

## S3 method for class 'xgb.Booster'
marshal(x, ...)

## S3 method for class 'xgb.Booster'
marshallable(...)

## S3 method for class 'xgb.DMatrix'
marshallable(...)
```

Arguments

x	An xgboost::xgb.DMatrix or an xgboost:xgb.Booster object.
...	Not used.

Details

xgboost::xgb.DMatrix.save() is used to produce a marshalled version of the original object. xgboost::xgb.DMatrix() is used to reconstruct a version of the original object from the marshalled object.

Value

A marshalled object as described in marshal().

---

marshal.xml_document *Marshalling of 'xml2' objects*

Description

Marshalling of 'xml2' objects

Usage

```
## S3 method for class 'xml_document'
marshal(xml2, ...)

## S3 method for class 'xml_nodeset'
marshal(xml2, ...)

## S3 method for class 'xml_document'
marshallable(...)

## S3 method for class 'xml_nodeset'
marshallable(...)
```

Arguments

xml2	A xml2::xml_document or similar.
...	Not used.

Details

xml2::xml_serialize() is used to produce a marshalled version of the original object. xml2::xml_unserialize() is used to reconstruct a version of the original object from the marshalled object.

Value

A marshalled object as described in marshal().

Examples

```
if (requireNamespace("xml2", quietly = TRUE)) {
  doc <- xml2::read_xml("<body></body>")

  ## Marshal 'xml_document' object
  doc_ <- marshal(doc)

  ## Unmarshal 'xml_document' object
  doc2 <- unmarshal(doc_)

  stopifnot(all.equal(doc2, doc))
}
```

---

marshal.XMLAbstractNode

Marshalling of 'XML' objects

Description

Marshalling of 'XML' objects

Usage

```
## S3 method for class 'XMLAbstractNode'
marshal(xml, ...)

## S3 method for class 'XMLAbstractDocument'
marshal(xml, ...)

## S3 method for class 'XMLAbstractDocument'
marshallable(...)

## S3 method for class 'XMLAbstractNode'
marshallable(...)
```

Arguments

xml	An XML::XMLAbstractNode or XML::XMLAbstractDocument.
...	Not used.

Details

XML::xmlSerializeHook() is used to produce a marshalled version of the original object. XML::xmlDeserializeHook() is used to reconstruct a version of the original object from the marshalled object.

Value

A marshalled object as described in marshal().

Examples

```
if (requireNamespace("XML", quietly = TRUE)) {
  node <- XML::xmlParseString("<a><b>c</b></a>")
  print(node)

  ## Marshal XMLAbstractNode object
  node_ <- marshal(node)
  ## Unmarshal XMLAbstractNode object
  node2 <- unmarshal(node_)
  print(node2)

  stopifnot(all.equal(node2, node))


  doc <- XML::xmlParse(system.file("exampleData", "tides.xml", package = "XML"))

  ## Marshal XMLAbstractDocument object
  doc_ <- marshal(doc)
  ## Unmarshal XMLAbstractDocument object
  doc2 <- unmarshal(doc_)

  stopifnot(all.equal(doc2, doc))
}
```

---

marshallable *Check if R object can be marshalled*

Description

Check if R object can be marshalled

Usage

```
marshallable(...)
```

Arguments

...	The object to be checked, followed by additional arguments passed to the specific S3 method.

Value

TRUE if the object can be marshalled, FALSE if it cannot be marshalled, and NA if it is not known whether the object can be marshalled.

Index

base::readBin(), *16*
base::writeBin(), *16*
bigmemory::big.matrix, *4*
bigmemory::read.big.matrix(), *4*
bigmemory::write.big.matrix(), *4*
bundle::bundle(), *22*
bundle::unbundle(), *22*

caret::train, *21*
CFunc, *5*
connection, *6*

data.table::data.table, *8*
DBI::DBIConnection, *9*

h2o::h2o.import_mojo(), *11*
h2o::h2o.loadModel(), *11*
h2o::h2o.save_mojo(), *11*
h2o::h2o.saveModel(), *11*

keras::serialize_model(), *12*
keras::unserialize_model(), *12*
keras:keras.engine.base_layer.Layer, *12*

luz::luz_load(), *14*
luz::luz_save(), *14*

magick:magick-image, *14*, *19*
marshal, 3
marshal(), *4–6*, *8–12*, *14–25*
marshal-package, 3
marshal.big.matrix, 4
marshal.bigmemory *(*marshal.big.matrix*)*, 4
marshal.CFunc, 5
marshal.connection, 6
marshal.data.table, 8
marshal.DBIConnection, 9
marshal.default, 9
marshal.H2OAutoML, 10
marshal.H2OBinomialModel *(*marshal.H2OAutoML*)*, 10
marshal.H2OMultinomialModel *(*marshal.H2OAutoML*)*, 10
marshal.H2ORegressionModel *(*marshal.H2OAutoML*)*, 10
marshal.inline *(*marshal.CFunc*)*, 5
marshal.jclassName, 11
marshal.jobjRef *(*marshal.jclassName*)*, 11
marshal.keras.engine.base_layer.Layer, 12
marshal.luz_module_fitted, 13
marshal.magick-image, 14
marshal.model_fit, 14
marshal.ncdf4, 15
marshal.python.builtin.object, 17
marshal.RasterLayer, 17
marshal.SOCK0node *(*marshal.SOCKcluster*)*, 18
marshal.SOCKcluster, 18
marshal.SOCKnode *(*marshal.SOCKcluster*)*, 18
marshal.SpatVector, 19
marshal.stanfit, 20
marshal.tbl_spark, 21
marshal.terra *(*marshal.SpatVector*)*, 19
marshal.train, 21
marshal.udpipe_model, 22
marshal.xgb.Booster *(*marshal.xgb.DMatrix*)*, 22
marshal.xgb.DMatrix, 22
marshal.XML *(*marshal.XMLAbstractNode*)*, 24
marshal.xml2 *(*marshal.xml_document*)*, 23
marshal.xml_document, 23
marshal.xml_nodeset *(*marshal.xml_document*)*, 23
marshal.XMLAbstractDocument *(*marshal.XMLAbstractNode*)*, 24

marshal.XMLAbstractNode, 24
marshallable, 25
marshallable.big.memory (marshal.big.matrix), 4
marshallable.CFunc (marshal.CFunc), 5
marshallable.connection (marshal.connection), 6
marshallable.data.table (marshal.data.table), 8
marshallable.DBIConnection (marshal.DBIConnection), 9
marshallable.H2OAutoML (marshal.H2OAutoML), 10
marshallable.H2OBinomialModel (marshal.H2OAutoML), 10
marshallable.H2OMultinomialModel (marshal.H2OAutoML), 10
marshallable.H2ORegressionModel (marshal.H2OAutoML), 10
marshallable.jclassName (marshal.jclassName), 11
marshallable.jobjRef (marshal.jclassName), 11
marshallable.keras.engine.base_layer.Layer (marshal.keras.engine.base_layer.Layer), 12
marshallable.luz_module_fitted (marshal.luz_module_fitted), 13
marshallable.magick-image (marshal.magick-image), 14
marshallable.model_fit (marshal.model_fit), 14
marshallable.ncdf4 (marshal.ncdf4), 15
marshallable.python.builtin.object (marshal.python.builtin.object), 17
marshallable.RasterLayer (marshal.RasterLayer), 17
marshallable.SOCK0node (marshal.SOCKcluster), 18
marshallable.SOCKcluster (marshal.SOCKcluster), 18
marshallable.SOCKnode (marshal.SOCKcluster), 18
marshallable.SpatVector (marshal.SpatVector), 19
marshallable.stanfit (marshal.stanfit), 20
marshallable.tbl_spark (marshal.tbl_spark), 21
marshallable.train (marshal.train), 21
marshallable.udpipe_model (marshal.udpipe_model), 22
marshallable.xgb.Booster (marshal.xgb.DMatrix), 22
marshallable.xgb.DMatrix (marshal.xgb.DMatrix), 22
marshallable.xml_document (marshal.xml_document), 23
marshallable.xml_nodeset (marshal.xml_document), 23
marshallable.XMLAbstractDocument (marshal.XMLAbstractNode), 24
marshallable.XMLAbstractNode (marshal.XMLAbstractNode), 24

ncdf4::nc_open(), *16*
ncdf4:ncdf4, *16*

raster::raster(), *18*
raster::writeRaster(), *18*
raster:RasterLayer, *17*

terra::SpatVector, *19*
terra::vect(), *19*
terra::wrap(), *19*

unmarshal (marshal), 3

xgboost::xgb.DMatrix, *23*
xgboost::xgb.DMatrix(), *23*
xgboost::xgb.DMatrix.save(), *23*
xml2::xml_document, *24*
xml2::xml_serialize(), *24*
xml2::xml_unserialize(), *24*
XML::XMLAbstractDocument, *25*
XML::XMLAbstractNode, *25*
XML::xmlDeserializeHook(), *25*
XML::xmlSerializeHook(), *25*