

# Package ‘nws’

February 14, 2012

**Title** R functions for NetWorkSpaces and Sleigh

**Version** 1.7.0.1

**Author** REvolution Computing <packages@revolution-computing.com> with support and contributions from Pfizer, Inc.

**Description** Provides coordination and parallel execution facilities,as well as limited cross-language data exchange, using the netWorkSpaces server developed by REvolution Computing

**Maintainer** REvolution Computing <packages@revolution-computing.com>

**License** GPL (>= 2)

**Depends** R (>= 2.1), methods

**LazyLoad** yes

**URL** <http://nws-r.sourceforge.net/>

**Repository** CRAN

**Date/Publication** 2010-04-13 16:23:56

## R topics documented:

nws-package . . . . .	3
batchNodeList . . . . .	3
checkSleigh . . . . .	4
close . . . . .	5
defaultSleighOptions . . . . .	6
eachElem . . . . .	6
eachWorker . . . . .	10
export . . . . .	12
germandata . . . . .	13
isClosure . . . . .	13
lsfcmd . . . . .	14
netWorkSpace . . . . .	15

netWorkspaceObject . . . . .	17
nwsClose . . . . .	17
nwsDeclare . . . . .	18
nwsDeleteVar . . . . .	18
nwsDeleteWs . . . . .	19
nwsFetch . . . . .	20
nwsFetchFile . . . . .	21
nwsFetchTry . . . . .	22
nwsFetchTryFile . . . . .	23
nwsFind . . . . .	24
nwsFindFile . . . . .	24
nwsFindTry . . . . .	25
nwsFindTryFile . . . . .	26
nwsIFetch . . . . .	27
nwsIFetchTry . . . . .	28
nwsIFind . . . . .	29
nwsIFindTry . . . . .	30
nwsListVars . . . . .	31
nwsListWss . . . . .	32
nwsMktempWs . . . . .	33
nwsOpenWs . . . . .	33
nwsPkgInfo . . . . .	34
nwsServer . . . . .	35
nwsServerObject . . . . .	36
nwsStore . . . . .	37
nwsStoreFile . . . . .	38
nwsUseWs . . . . .	39
nwsVariable . . . . .	39
nwsWsName . . . . .	41
rankCount . . . . .	41
rshcmd . . . . .	42
sleigh . . . . .	43
sleighPending . . . . .	45
sshcmd . . . . .	46
status . . . . .	47
stopSleigh . . . . .	48
unexport . . . . .	49
waitSleigh . . . . .	50
workerCount . . . . .	50
workerInfo . . . . .	51

---

nws-package

*The NetWorkSpaces Package*


---

### Description

NetWorkSpaces makes it easy to create and experiment with parallel programs without requiring specialized tools or hardware. You must install both the NetWorkSpaces server (on one machine) and the R NetWorkSpaces package (on all machines involved in the computation). The server is implemented using Python and Twisted (a Python package), which are required. To download and install the server, go to the R NetWorkSpaces project on Source Forge at <http://sourceforge.net/projects/nws-r>.

A commercial version of NetWorkSpaces which has additional fault tolerance features is available as part of REvolution Computing's ParallelR product. More information is available at <http://www.revolution-computing.com/revolution/web/static/products>.

### Details

Further information is available in the following help topics:

sleigh	Create a sleigh for parallel execution
eachWorker	Execute a function on each of the workers in the sleigh
eachElem	Execute a function for a set of arguments
netWorkSpace	Create a NetWorkSpace
nwsStore	Assign a value to a variable in a NetWorkSpace
nwsFetch	Consume a value from a variable in a NetWorkSpace
nwsFind	Retrieve but don't consume a value from a variable in a NetWorkSpace

For a complete list of functions with individual help pages, use `library(help="nws")`.

---

batchNodeList

*NodeList Functions*


---

### Description

Return the list of nodes that SGE, LSF, or PBS has allocated for our process.

### Usage

```
batchNodeList()
sgeNodeList()
lsfNodeList()
pbsNodeList()
```

**Details**

These functions should only be called from an R program that has been submitted as a parallel batch job by SGE, LSF, or PBS/Torque. The `batchNodeList` function calls either `sgeNodeList`, `lsfNodeList`, or `pbsNodeList` depending on what environment variables are defined. The resulting list should be passed to the `sleigh` function via the `nodeList` argument.

**Value**

A character vector to pass to `sleigh` via the `nodeList` argument.

**See Also**

[sleigh](#)

**Examples**

```
Sys.setenv(LSB_HOSTS="node1 node2 node3")
batchNodeList()
```

---

checkSleigh

*Number of Results Pending From Sleigh*

---

**Description**

Return the number of results yet to be generated for the pending `sleigh` job.

**Usage**

```
## S4 method for signature 'sleighPending'
checkSleigh(.Object)
```

**Arguments**

`.Object` a `sleighPending` class object

**Details**

The `sleighPending` class object, `.Object`, is usually obtained through non-blocking `eachElem` or non-blocking `eachWorker`. If the pending job is finished, that is, all results are generated, then 0 is returned.

**See Also**

[eachWorker](#), [eachElem](#)

### Examples

```
## Not run:
eo = list(blocking=0)
s = sleigh()
sp = eachElem(s, function(x) {Sys.sleep(100); x}, list(1:10), eo=eo)
checkSleigh(sp)

## End(Not run)
```

---

close

*Close a Sleigh or netWorkspace*

---

### Description

If the first argument is a sleigh object, then close calls stopSleigh to shut down the sleigh workers and delete the sleigh workspace. If the first argument is a nwsServer object, then the connection to the netWorkspace server is closed.

### Usage

```
## S4 method for signature 'nwsServer'
close(con, ...)
## S4 method for signature 'sleigh'
close(con, ...)
```

### Arguments

con	an object of class sleigh or nwsServer.
...	optional fields

### Details

The optional arguments are not passed to stopSleigh method. They are defined to be compatible with the default, non-generic close method.

### Examples

```
## Not run:
s = sleigh()
close(s)

wss = nwsServer()
close(wss)

## End(Not run)
```

---

defaultSleighOptions    *Default Sleigh Options Environment*

---

### Description

This environment specifies the default options that are used when creating a new sleigh object.

### Details

The default options can be modified by assigning new values to this environment. Those new values will be used when constructing all new sleigh objects. You can also define new entries, which can be used from custom launch functions.

Note that the defaultSleighOptions environment is used to check for illegal options to the sleigh initializer. Therefore, to allow new options to be passed to a custom launch function, you must first define a default value for that option in defaultSleighOptions.

### See Also

[sleigh](#)

### Examples

```
defaultSleighOptions$user
```

---

eachElem                    *Apply a Function in Parallel over a Set of Lists and Vectors*

---

### Description

eachElem executes function fun multiple times in parallel with a varying set of arguments, and returns the results in a list. It is functionally similar to the standard R lapply function, but is more flexible in the way that the function arguments can be specified.

### Usage

```
## S4 method for signature 'sleigh'  
eachElem(.Object, fun, elementArgs=list(), fixedArgs=list(),  
          eo=NULL, DEBUG=FALSE)
```

**Arguments**

.Object	sleigh class object.
fun	the function to be evaluated by the sleigh. In the case of functions like +, %*%, etc., the function name must be quoted.
elementArgs	list of vectors, lists, matrices, and data frames that specify (some of) the arguments to be passed to fun. Each element should correspond to an argument of fun.
fixedArgs	list of additional arguments to be passed to fun. Each element should correspond to an argument of fun.
eo	list specifying environment options. See the section Environment Options below.
DEBUG	logical; should browser function be called upon entry to eachElem? The default is FALSE.

**Details**

The eachElem function forms argument sets from objects passed in via elementArgs and fixedArgs. The elements of elementArgs are used to specify the arguments that are changing, or varying, from task to task, while the elements of fixedArgs are used to specify the arguments that do not vary from task to task. The number of tasks that are executed by a call to eachElem is basically equal to the length of the longest vector (or list, etc) in elementArgs. If any elements of elementArgs are shorter, then their values are recycled, using the standard R rules.

The elements of elementArgs may be vectors, lists, matrices, or data frames. The vectors and lists are always iterated over by element, or "cell", but matrices and data frames can also be iterated over by row or column. This is controlled by the by option, specified via the eo argument. See below for more information.

For example:

```
eachElem(s, '+', elementArgs=list(1:4), fixedArgs=list(100))
```

This will submit four tasks, since the length of 1:4 is four. The four tasks will be to add the arguments 1 and 100, 2 and 100, 3 and 100, and 4 and 100. The result is a list containing the four values 101, 102, 103, and 104.

Another way to do the same thing is with:

```
eachElem(s, '+', elementArgs=list(1:4, 100))
```

Since the second element of elementArgs is length one, it's value is recycled four times, thus specifying the same set of tasks as in the previous example. This method also has the advantage of making it easy to put fixed values before varying values, without the need for the eo\$argPermute option, discussed later. For example:

```
eachElem(s, '-', elementArgs=list(100, 1:4))
```

is similar to the R statement:

100 - 1:4

Note that in simple examples like these, where the results are numeric values, the standard R `unlist` function can be very useful for converting the resulting list into a vector.

## Environment Options

The `eo` argument is a list that can be used to specify various options. The following options are recognized:

**elementFunc** The `eo$elementFunc` option can be used to specify a callback function that provides the varying arguments for `fun` in place of `elementArgs` (that is, you can't specify both `eo$elementFunc` and `elementArgs`). `eachElem` calls the `eo$elementFunc` function to get a list of arguments for one invocation of `fun`, and will keep calling it until `eo$elementFunc` signals that there are no more tasks to execute by calling the `stop` function with no arguments. `eachElem` appends any values specified by `fixedArgs` to the list returned by `eo$elementFunc` just as if `elementArgs` had been specified.

`eachElem` passes the number of the desired task (starting from 1) as the first argument to `eo$elementFunc`, and the value of the `eo$by` option as the second argument. Note that the use of the `eo$elementFunc` function is an advanced feature, but is very useful when executing a large number of tasks, or when the arguments are coming from a database query, for example. For that reason, the `eo$loadFactor` option should usually be used in conjunction with `eo$elementFunc` (see description below).

**accumulator** The `eo$accumulator` option can be used to specify a callback function that will receive the results of the task execution as soon as they are complete, rather than returning all of the task results as a list when `eachElem` completes. In other words, `eachElem` will call the `eo$accumulator` function with task results as soon as it receives them from the sleigh workers, rather than saving them in memory until all the tasks are complete. Note that if the tasks are *chunked* (using the `eo$chunkSize` option described below), then the `eo$accumulator` function will receive multiple task results, which is why the task results are always passed to the `eo$accumulator` function in a list.

The first argument to the `eo$accumulator` function is a list of results, where the length of the list is equal to `eo$chunkSize`. The second argument is a vector of task numbers, starting from 1, where the length of the vector is also equal to `eo$chunkSize`. The task numbers are very important, because the results are not guaranteed to be returned in order. `eo$accumulator` is another advanced feature, and like `eo$elementFunc`, is very useful when executing a large number of tasks. It allows you to process each result as they finish, rather than forcing you to wait until all of the tasks are complete. In conjunction with `eo$elementFunc` and `eo$loadFactor`, you can set up a pipeline, allowing you to process an unlimited number of tasks efficiently. Note that when `eo$accumulator` is specified, `eachElem` returns `NULL`, not the list of results, since `eachElem` doesn't save any of the results after passing them to the `eo$accumulator` function.

**by** The `eo$by` option specifies the iteration scheme to use for matrix and data frame elements in `elementArgs`. The default value is `"row"`, but it can also be set to `"column"` or `"cell"`. Vectors and lists in `elementArgs` are not affected by this option.

**chunkSize** The `eo$chunkSize` option is a tuning parameter that specifies the number of tasks that sleigh workers should allocate at a time. The default value is 1, but if the tasks are small, performance can be improved by specifying a larger value, which decreases the overhead per task.

If the fun function executes very quickly, you may not be able to keep your workers busy, giving you poor performance. In that case, consider setting the `eo$chunkSize` option to a large enough number to increase the effective task execution time.

**loadFactor** The `eo$loadFactor` option is a tuning parameter that specifies the maximum number of tasks per worker that are submitted to the sleigh at the same time. If set, no more than  $(\text{loadFactor} * \text{workerCount})$  tasks will be submitted at the same time. This helps to control the resource demands that are made on the NetWorkSpaces server, which is especially important if there are a large number of tasks. Note that this option is ignored if `blocking` is set to `TRUE`, since the two options are incompatible with each other.

If in doubt, set the `eo$loadFactor` option to 10. That will almost certainly avoid putting a strain on the NetWorkSpaces server, and if that isn't enough to keep your workers busy, then you should really be using the `eo$chunkSize` option to give the workers more to do.

**blocking** The `eo$blocking` option is used to indicate whether to wait for the results, or to return as soon as the tasks have been submitted. If set to `FALSE`, `eachElem` will return a `sleighPending` object that is used to monitor the status of the tasks, and to eventually retrieve the results. You must wait for the results to be complete before executing any further tasks on the sleigh, or an exception will be raised. The default value is `TRUE`.

**argPermute** The `eo$argPermute` option is used to reorder the arguments passed to fun. It is generally only useful if the `fixedArgs` argument has been specified, and some of those arguments need to precede the arguments specified via `elementArgs`. Note that by using recycling of elements in `elementArgs`, the use of `fixedArgs` and `argPermute` can often be avoided entirely.

## Note

If `elementArgs` or `fixedArgs` isn't a list, `eachElem` will automatically wrap it in a list. This is a convenience that only works for passing in a single vector and matrix, however.

If `elementArgs` or `fixedArgs` are named lists, then the names are used to map the values to the appropriate argument of fun. This can be used as another technique to avoid the use of `eo$argPermute`.

The `elementArgs` argument can be specified as a data frame. This works just like a named list, and therefore, the column names of the data frame must all correspond to arguments of fun. Note that if the data frame has many rows, the performance may not be good due to the overhead of subsetting data frames in R.

If you have a huge number of tasks, consider using the `eo$elementFunc`, `eo$accumulator`, and `eo$loadFactor` options.

If `eo$elementFunc` returns a value that isn't a list, `eachElem` will automatically wrap that value in a list.

The `eo$elementFunc` function doesn't have to define a second formal argument (the `by` argument) if it's not needed.

The `eo$accumulator` function doesn't have to define a second formal argument (the `taskVector` argument) if it's not needed. Just remember that the results are not guaranteed to come back in order.

## See Also

[eachWorker](#), [sleighPending](#)

**Examples**

```

## Not run:
# create a sleigh
s <- sleigh()

# compute the list mean for each list element
x <- list(a=1:10, beta=exp(-3:3), logic=c(TRUE,FALSE,FALSE,TRUE))
eachElem(s, mean, list(x))

# median and quartiles for each list element
eachElem(s, quantile, elementArgs=list(x), fixedArgs=list(probs=1:3/4))

# use eo$elementFunc to supply 100 random values and eo$accumulator to
# receive the results
elementFunc <- function(i, by) {
  if (i <= 100) list(i=i, x=runif(1)) else stop()
}
accumulator <- function(resultList, taskVector) {
  if (resultList[[1]][[1]] != taskVector[1]) stop('assertion failure')
  cat(paste(resultList[[1]], collapse=' '), '\n')
}
eo <- list(elementFunc=elementFunc, accumulator=accumulator)
eachElem(s, function(i, x) list(i=i, x=x, xsq=x*x), eo=eo)

## End(Not run)

```

---

eachWorker

*Execute a Function in Parallel on all Workers of a Sleigh*


---

**Description**

eachWorker executes function fun once on each worker in the specified sleigh, and returns the results in a list. This can be useful for initializing each of the workers before starting to execute tasks using eachElem. Loading packages using the library function, loading data sets using the data function, and assigning variables in the global environment are common tasks for eachWorker.

**Usage**

```

## S4 method for signature 'sleigh'
eachWorker(.Object, fun, ..., eo=NULL, DEBUG=FALSE)

```

**Arguments**

.Object	sleigh class object.
fun	the function to be evaluated by each of the sleigh workers. In the case of functions like +, %*%, etc., the function name must be quoted.
...	optional arguments to fun.
eo	additional options, see details
DEBUG	logical; should browser function be called upon entry to eachWorker?

## Details

The `eo` argument is a list that can be used to specify various options. The current options are: `eo$blocking`, and `eo$accumulator`.

The `eo$blocking` option is used to indicate whether to wait for the results, or to return as soon as the tasks have been submitted. If set to `FALSE`, `eachWorker` will return a `sleighPending` object that is used to monitor the status of the tasks, and to eventually retrieve the results. You must wait for the results to be complete before executing any further tasks on the sleigh, or an exception will be raised. The default value is `TRUE`.

The `eo$accumulator` option can be used to specify a function that will receive the results of the task execution. Note that while this can be a very useful feature with `eachElem`, it's not commonly used with `eachWorker`, but is included for consistency. The first argument to `eo$accumulator` function is a list of results, where the length of the list is always equal to 1 (because there is no `eo$chunkSize` option in `eachWorker`). The second argument is a vector of task numbers, starting from 1, where the length of the vector is also always equal to 1. The task numbers are not very important when used with `eachWorker`, because the order of tasks isn't specified, as it is with `eachElem`. Note that when `eo$accumulator` is specified, `eachWorker` returns `NULL`, not the list of results, since `eachWorker` doesn't save any of the results after passing them to the `eo$accumulator` function.

The `DEBUG` argument is used call the browser function upon entering `eachWorker`. The default value is `FALSE`.

## Note

The `eo$blocking` option can be very useful for starting a function on each of the workers, and then allowing the master process to interact with the workers via `NetWorkspace` operations in order to implement sophisticated parallel applications.

## See Also

[eachElem](#), [sleighPending](#)

## Examples

```
## Not run:
# create a sleigh
s <- sleigh()

# assign to global variable x on each worker
eachWorker(s, function() x <<- 1)

# get a listing of each worker's global environment
eachWorker(s, function() ls(globalenv()))

# get system info from each worker
eachWorker(s, Sys.info)

# load MASS package on each worker
eachWorker(s, function() library(MASS))

# non-blocking example using simple NWS operations
```

```

sp <- eachWorker(s, function() nwsFind(SleighNws, 'hello'), eo=list(blocking=FALSE))
nwsStore(s@nws, 'hello', 'world')
waitSleigh(sp)

## End(Not run)

```

---

export

*sleigh Class Method*


---

### Description

Define a variable in the scope of the workers or specified worker.

### Usage

```

## S4 method for signature 'sleigh'
export(.Object, xName, xVal, worker=NULL)

```

### Arguments

<code>.Object</code>	a sleigh class object.
<code>xName</code>	a variable name (given as a quoted string in the function call).
<code>xVal</code>	a value to be assigned to 'x'.
<code>worker</code>	integer rank of the worker where the variable is defined. The rank values range from 0 to <code>workerCount - 1</code> . By default, the variable is defined on all workers in the sleigh.

### See Also

[unexport](#)

### Examples

```

## Not run:
s <- sleigh()
m <- matrix(rnorm(16), 4, 4)
export(s, 'm', m)
eachWorker(s, function() m %% matrix(rnorm(16), 4, 4), eo=list(closure=FALSE))

## End(Not run)

```

---

germandata

*German Credit data*

---

### Description

This dataset produced by Strathclyde University contains numerical attributes converted from the original dataset provided by Prof. Hofmann.

### Usage

germandata

### Format

a table contains 1000 instances and 24 attributes for each instance.

### Source

<http://ftp.ics.uci.edu/pub/machine-learning-databases/statlog/german>

---

isClosure

*Determine if a Worker Function is a Closure*

---

### Description

This is a heuristic function that is used by eachWorker and eachElem to guess if the worker function is a closure.

### Details

If the closure option wasn't specified via the eo argument to eachWorker or eachElem, then this function is used to guess if the worker function is a closure. It can be very useful to use closures with eachWorker and eachElem, but if not used properly, you could accidentally include a lot of unnecessary data in the tasks, thus hurting your performance.

This function is included as a development tool so you can manually test your worker functions. This could be useful if you are getting a warning from eachWorker or eachElem, and are trying to determine how to modify the function or set the closure option.

### Examples

```
# this should return FALSE
isClosure(sqrt)
f <- function(x) function(y) x + y
g <- f(1)
# this should return TRUE
isClosure(g)
```

---

lsfcmd

*Sleigh Auxiliary Function*

---

## Description

This function is used by the sleigh constructor when starting workers on remote nodes using the LSF bsub command. Note that it doesn't actually start any workers directly: it simply returns the program name and arguments to start a worker on the specified node.

## Usage

```
lsfcmd(host, options)
```

## Arguments

host	Name from the nodeList. This is currently ignored.
options	An environment or list. This is currently ignored.

## Details

lsfcmd is not intended to be called by the user. It is called by the sleigh constructor when specified via the sleigh launch argument. You may want to execute it when debugging your sleigh option settings, but that can also be accomplished by setting the sleigh verbose argument to TRUE.

## Value

The character vector `c('bsub')`.

## See Also

[sleigh](#)

## Examples

```
## Not run:  
# Create a sleigh with workers on nodes n1 and n2 started via lsf:  
s <- sleigh(launch=lsfcmd, nodeList=rep('fake', 10))  
  
## End(Not run)
```

---

netWorkspace	<i>NetWorkSpaces Objects</i>
--------------	------------------------------

---

## Description

NetWorkSpaces provides a framework to coordinate R programs. NetWorkSpaces objects represent a generalized workspace or environment. Two or more R programs communicate data by storing it in and retrieving it from the NetWorkspace object.

## Objects from the Class

Objects can be created by calls of the form

```
netWorkspace(wsName, serverHost, port, useUse, serverWrap, ...)
```

or

```
new("netWorkspace", wsName, serverHost, port, useUse, serverWrap, ...).
```

`wsName` name of the netWorkspace to be created.

`serverHost` host name of the server this netWorkspace will be connected to. By default, local machine is used.

`port` port number of the server this netWorkspace will be connected to. Default port number is 8765.

`useUse` a logical value indicating whether ownership will be claimed for this netWorkspace. By default, `useUse=FALSE`, which means ownership will be claimed.

`serverWrap` a netWorkSpaces server object. Reuse an existing server connection, instead of creating a new server connection.

## Slots

`cookieProtocol` Currently not used.

`server` Object of class "nwsServer" representation of the server that this netWorkspace connects to.

`wsName` Object of class "character" representation of this netWorkspace's name.

## Methods

**initialize** signature(.Object = "netWorkspace"): netWorkspace class constructor.

`nwsFetch` signature(.Object = "netWorkspace"): fetch a value of a workspace variable.

`nwsFetchTry` signature(.Object = "netWorkspace"): try to fetch a value of a workspace variable.

`nwsFind` signature(.Object = "netWorkspace"): find a value of a workspace variable.

`nwsFindTry` signature(.Object = "netWorkspace"): try to find a value of a workspace variable.

`nwsStore` signature(.Object = "netWorkspace"): store a value into a workspace variable.

nwsFetchFile signature(.Object = "netWorkspace"): fetch a value of a workspace variable and write it to a file.

nwsFetchTryFile signature(.Object = "netWorkspace"): try to fetch a value of a workspace variable and write it to a file.

nwsFindFile signature(.Object = "netWorkspace"): find a value of a workspace variable and write it to a file.

nwsFindTryFile signature(.Object = "netWorkspace"): try to find a value of a workspace variable and write it to a file.

nwsStoreFile signature(.Object = "netWorkspace"): store data from a file into a workspace variable.

nwsIFetch signature(.Object = "netWorkspace"): create a function that acts as a destructive iterator over the values of the specified variable.

nwsIFetchTry signature(.Object = "netWorkspace"): create a function that acts as a destructive iterator over the values of the specified variable.

nwsIFind signature(.Object = "netWorkspace"): create a function that acts as a non-destructive iterator over the values of the specified variable.

nwsIFindTry signature(.Object = "netWorkspace"): create a function that acts as a non-destructive iterator over the values of the specified variable.

nwsClose signature(.Object = "netWorkspace"): close the connection to the NWS server.

nwsDeclare signature(.Object = "netWorkspace"): declare the mode of a workspace variable.

nwsDeleteVar signature(.Object = "netWorkspace"): delete a variable from a workspace.

nwsListVars signature(.Object = "netWorkspace"): list all variables in a workspace.

ServerObject signature(.Object = "netWorkspace"): return the associated NwsServer object.

nwsVariable signature(.Object = "netWorkspace"): create an Active Binding for a NetWorkspace Variable

nwsWsName signature(.Object = "netWorkspace"): return the name of the workspace.

## Examples

```
## Not run:
# To create a new workspace with the name "my space" use:
ws = netWorkspace('my space')

# To create a new workspace called "my space2" on nws server
# running on port 8245 on machine zeus:
ws2 = netWorkspace(wsName='my space2', serverHost='zeus', port=8245)

## End(Not run)
```

---

netWorkspaceObject      *sleigh Class Method*

---

**Description**

Return the netWorkspace object associated with the sleigh.

**Usage**

```
## S4 method for signature 'sleigh'  
netWorkspaceObject(.Object)
```

**Arguments**

.Object              a sleigh class object

**Examples**

```
## Not run:  
s = sleigh()  
netWorkspaceObject(s)  
  
## End(Not run)
```

---

nwsClose              *Close a netWorkspace*

---

**Description**

Close connection of a shared netWorkspace to the netWorkspace server.

**Usage**

```
## S4 method for signature 'netWorkspace'  
nwsClose(.Object)
```

**Arguments**

.Object              a netWorkspace class object

**Examples**

```
## Not run:  
ws <- netWorkspace('nws example')  
# do some works  
# ...  
nwsClose(ws)  
  
## End(Not run)
```

---

nwsDeclare	<i>Declare a netWorkspace Variable</i>
------------	--

---

### Description

Declare a variable with particular mode in a shared netWorkspace.

### Usage

```
nwsDeclare(.Object, xName, mode)
```

### Arguments

.Object	a netWorkspace class object.
xName	character string giving the name of variable to be declared.
mode	character string specifying the mode of the variable, see Details.

### Details

If xName has not already been declared in the netWorkspace, the behavior of xName will be determined by mode. Possible values are 'fifo' (the default), 'lifo', 'multi', or 'single'. In the first three cases, multiple values can be associated with xName. When a value is retrieved for xName, the oldest value stored will be used in 'fifo' mode, the youngest in 'lifo' mode, and a non-deterministic choice will be made in 'multi' mode. In 'single' mode, only the most recent value is retained.

### Examples

```
## Not run:  
ws <- netWorkspace('nws example')  
nwsDeclare(ws, 'pi', 'single')  
nwsStore(ws, 'pi', 2.171828182)  
nwsStore(ws, 'pi', 3.141592654)  
nwsListVars(ws) # shows that only the most recent value of pi is retained  
  
## End(Not run)
```

---

nwsDeleteVar	<i>Delete a Variable from a netWorkspace</i>
--------------	--

---

### Description

Delete a variable from the shared netWorkspace.

**Usage**

```
## S4 method for signature 'netWorkspace'
nwsDeleteVar(.Object, xName)
```

**Arguments**

```
.Object      a netWorkspace class object.
xName        character string specifying the name of the variable to be deleted.
```

**Examples**

```
## Not run:
ws <- netWorkspace('nws example')
nwsStore(ws, 'x', 10)
nwsDeleteVar(ws, 'x')

## End(Not run)
```

---

nwsDeleteWs	<i>Delete a netWorkspace</i>
-------------	------------------------------

---

**Description**

Delete a shared netWorkspace from the netWorkSpaces server.

**Usage**

```
## S4 method for signature 'nwsServer'
nwsDeleteWs(.Object, wsName)
```

**Arguments**

```
.Object      a nwsServer class object
wsName       name of the netWorkspace to be deleted
```

**Examples**

```
## Not run:
# example 1
nwss <- nwsServer()
ws <- nwsOpenWs(nwss, "nws example")
# do some works
# ...
nwsDeleteWs(nwss, "nws example")

# example 2 illustrates accessing a server object
# from the netWorkspace class object
ws <- netWorkspace("nws example 2")
```

```
# do some works
# ...
nwsDeleteWs(ws@server, "nws example 2")

## End(Not run)
```

---

nwsFetch

*Fetch a Stored Value*


---

## Description

Fetch value associated with a variable from the shared netWorkspace.

## Usage

```
## S4 method for signature 'netWorkspace'
nwsFetch(.Object, xName)
```

## Arguments

.Object	a netWorkspace class object
xName	name of the variable to be fetched

## Details

The nwsFetch method looks in the shared netWorkspace .Object for a value bound to xName; if it finds such a value, nwsFetch returns it and removes it from the variable. If no value is found, nwsFetch blocks until a value for xName becomes available. This operation is atomic. If there are multiple nwsFetch or nwsFetchTry requests for a given xName, any given value from the set of values associated with xName will be returned to just one requester. If there is more than one value associated with xName, the particular value removed depends on xName's behavior. See [nwsDeclare](#) for details.

## See Also

[nwsDeclare](#), [nwsFetchTry](#)

## Examples

```
## Not run:
ws <- netWorkspace('nws example')
nwsStore(ws, 'x', 10)
nwsFetch(ws, 'x')
nwsFetch(ws, 'x') # no value for x; therefore block on fetch

## End(Not run)
```

---

nwsFetchFile	<i>Fetch a Value and Write It to a File</i>
--------------	---

---

## Description

Fetch a value of a netWorkspace variable and write it to a file.

## Usage

```
## S4 method for signature 'netWorkspace'  
nwsFetchFile(.Object, xName, fObj)
```

## Arguments

.Object	a netWorkspace object.
xName	character string giving the name of the variable to find.
fObj	file object or character string specifying file to write data to.

## Details

The nwsFetchFile method looks in the shared netWorkspace .Object for a value bound to xName; if it finds such a value, nwsFetchFile writes the value to the specified file and removes it from the variable. If no value is found, nwsFetchFile blocks until a value for xName becomes available.

## See Also

[nwsFindFile](#), [nwsFetch](#)

## Examples

```
## Not run:  
ws <- netWorkspace('nws example')  
nwsStore(ws, 'x', 'Hello, world\n')  
nwsFetchFile(ws, 'x', 'hello.txt')  
  
## End(Not run)
```

---

nwsFetchTry

*Fetch a Value from a NetWorkspace (Non-Blocking Version)*


---

### Description

Attempt to fetch a value associated with a variable from the shared netWorkspace; a non-blocking version of [nwsFetch](#).

### Usage

```
## S4 method for signature 'netWorkspace'
nwsFetchTry(.Object, xName, defaultVal=NULL)
```

### Arguments

.Object	a netWorkspace class object
xName	name of the variable to be fetched
defaultVal	value to return, if xName is not found in the netWorkspace

### Details

Look in the shared netWorkspace for a value bound to xName. If found, remove a value associated with xName from the shared netWorkspace. This operation is atomic. If there are multiple nwsFetch or nwsFetchTry requests for a given xName, any given value from the set of values associated with xName will be returned to just one requester.

If variable is not found, return immediately rather than block on the operation (as in the case of [nwsFetch](#)). If variable is not found, the value of argument defaultVal is returned. By default, defaultVal is NULL.

### See Also

[nwsDeclare](#), [nwsFetch](#)

### Examples

```
## Not run:
ws <- netWorkspace('nws example')
# If variable 'x' is not found in the shared netWorkspace,
# return default value, NULL.
nwsFetchTry(ws, 'x')
# If variable 'x' is not found in the shared netWorkspace,
# return 10.
nwsFetchTry(ws, 'x', 10)

## End(Not run)
```

---

nwsFetchTryFile	<i>Fetch a Stored Value and Write It to a File</i>
-----------------	--

---

## Description

Fetch a value of a netWorkspace variable and write it to a file.

## Usage

```
## S4 method for signature 'netWorkspace'  
nwsFetchTryFile(.Object, xName, fObj)
```

## Arguments

.Object	a netWorkspace object.
xName	character string specifying name of the variable to find.
fObj	file object or character string specifying file to write data to.

## Details

The nwsFetchTryFile method looks in the shared netWorkspace .Object for a value bound to xName; if it finds such a value, nwsFetchTryFile writes the value to the specified file, removes it from the variable, and the method returns TRUE. If it is not found, nwsFetchTryFile returns FALSE.

## See Also

[nwsFindTryFile](#), [nwsFetchTry](#)

## Examples

```
## Not run:  
ws <- netWorkspace('nws example')  
nwsStore(ws, 'x', 'Hello, world\n')  
if (nwsFetchTryFile(ws, 'x', 'hello.txt')) {  
  cat('success\n')  
} else {  
  cat('failure\n')  
}  
  
## End(Not run)
```

---

`nwsFind`*Find a Stored Value*

---

**Description**

Find a value associated with a variable in a shared netWorkspace.

**Usage**

```
## S4 method for signature 'netWorkspace'  
nwsFind(.Object, xName)
```

**Arguments**

<code>.Object</code>	a netWorkspace class object.
<code>xName</code>	character string specifying the name of the variable to find.

**Details**

The `nwsFind` method looks in the shared netWorkspace `.Object` for a value bound to `xName`; if it finds such a value, `nwsFind` returns it but does not remove it. If no value is found, `nwsFind` blocks until a value for `xName` becomes available. If there is more than one value associated with `xName`, the particular value returned depends on `xName`'s behavior. See [nwsDeclare](#) for details.

**See Also**

[nwsDeclare](#), [nwsFetch](#)

**Examples**

```
## Not run:  
ws <- netWorkspace('nws example')  
nwsStore(ws, 'x', 1)  
x <- nwsFind(ws, 'x')  
  
## End(Not run)
```

---

`nwsFindFile`*Find a Stored Value and Write It to a File*

---

**Description**

Find a value of a workspace variable and write it to a file.

**Usage**

```
## S4 method for signature 'netWorkspace'
nwsFindFile(.Object, xName, fObj)
```

**Arguments**

.Object	a netWorkspace object.
xName	character string specifying the name of the variable to find.
fObj	file object or character string specifying the file to write data to.

**Details**

The nwsFindFile method looks in the shared netWorkspace .Object for a value bound to xName; if it finds such a value, nwsFind writes it to the specified file but does not remove it from the variable. If no value is found, nwsFindFile blocks until a value for xName becomes available. If there is more than one value associated with xName, the particular value returned depends on xName's behavior. See [nwsDeclare](#) for details.

**See Also**

[nwsFetchFile](#), [nwsFind](#)

**Examples**

```
## Not run:
ws <- netWorkspace('nws example')
nwsStore(ws, 'x', 'Hello, world\n')
nwsFindFile(ws, 'x', 'hello.txt')

## End(Not run)
```

---

nwsFindTry

*Find a Stored Value (Non-Blocking Version)*


---

**Description**

Attempt to find a value associated with a variable from the shared netWorkspace; a non-blocking version of [nwsFind](#).

**Usage**

```
## S4 method for signature 'netWorkspace'
nwsFindTry(.Object, xName, defaultVal=NULL)
```

**Arguments**

.Object	a netWorkspace class object
xName	name of variable to be found
defaultVal	value to return if xName is not found

**Details**

The nwsFindTry method looks in the shared netWorkspace .Object for a value bound to xName; if it finds such a value, nwsFindTry returns it but does not remove it. If it does not find a value, nwsFindTry returns immediately, rather than blocking as in the case of [nwsFind](#)), and the value of argument defaultVal is returned. By default, defaultVal is NULL.

If there is more than one value associated with xName, the particular value returned depends on xName's behavior. See [nwsDeclare](#) for details.

**See Also**

[nwsDeclare](#), [nwsFind](#)

**Examples**

```
## Not run:
ws <- netWorkspace('nws example')
x <- nwsFindTry(ws, 'abc', -1)

## End(Not run)
```

---

nwsFindTryFile

*Find a Stored Value and Write It to a File*


---

**Description**

Find a value of a netWorkspace variable and write it to a file.

**Usage**

```
## S4 method for signature 'netWorkspace'
nwsFindTryFile(.Object, xName, fObj)
```

**Arguments**

.Object	a netWorkspace object.
xName	character string specifying the name of the variable to find.
fObj	file object or character string specifying the file to write data to.

**Details**

The `nwsFindTryFile` method looks in the shared `netWorkspace` `.Object` for a value bound to `xName`; if it finds such a value, `nwsFind` writes it to the specified file but does not remove it from the variable, and returns `TRUE`. If no value is found, `nwsFindFile` returns `FALSE`. If there is more than one value associated with `xName`, the particular value returned depends on `xName`'s behavior. See [nwsDeclare](#) for details.

**See Also**

[nwsFetchTryFile](#), [nwsFindTry](#)

**Examples**

```
## Not run:
ws <- netWorkspace('nws example')
nwsStore(ws, 'x', 'Hello, world\n')
if (nwsFindTryFile(ws, 'x', 'hello.txt')) {
  cat('success\n')
} else {
  cat('failure\n')
}

## End(Not run)
```

---

nwsIFetch

*Iterate Through Values in a netWorkspace Variable*


---

**Description**

Create a function that acts as a destructive iterator over the values of the specified variable.

**Usage**

```
## S4 method for signature 'netWorkspace'
nwsIFetch(.Object, xName)
```

**Arguments**

<code>.Object</code>	a <code>netWorkspace</code> class object.
<code>xName</code>	character string specifying name of the variable to be fetched

**Details**

The iterator function returned by the `nwsIFetch` method takes no arguments, and works just like the `nwsFetch` method, specified with the same arguments that were passed to `nwsIFetch` method. Note that the `nwsIFind` and `nwsIFindTry` methods are much more useful, since they provide the only way to iterate over values of a variable non-destructively. The `nwsIFetch` and `nwsIFetchTry` methods are provided for completeness.

**See Also**

[nwsFetch](#), [nwsIFetchTry](#)

**Examples**

```
## Not run:
ws <- netWorkspace('nws example')
nwsStore(ws, 'x', 10)
it <- nwsIFetch(ws, 'x')
it() # returns the value 10
it() # blocks until another process stores a value in the variable

## End(Not run)
```

---

nwsIFetchTry

*Iterate Through Values of a netWorkspace Variable*

---

**Description**

Create a function that acts as a destructive iterator over the values of the specified variable.

**Usage**

```
## S4 method for signature 'netWorkspace'
nwsIFetchTry(.Object, xName, defaultVal=NULL)
```

**Arguments**

<code>.Object</code>	a <code>netWorkspace</code> class object.
<code>xName</code>	character string specifying name of the variable to be fetched.
<code>defaultVal</code>	value to return, if <code>xName</code> is not found in the <code>netWorkspace</code> .

**Details**

The iterator function returned by the `nwsIFetchTry` method takes no arguments, and works just like the `nwsFetchTry` method, specified with the same arguments that were passed to `nwsIFetchTry` method. Note that the `nwsIFind` and `nwsIFindTry` methods are much more useful, since they provide the only way to iterate over values of a variable non-destructively. The `nwsIFetch` and `nwsIFetchTry` methods are provided for completeness.

**See Also**

[nwsFetchTry](#), [nwsIFetch](#)

## Examples

```
## Not run:
ws <- netWorkspace('nws example')
nwsStore(ws, 'x', 10)
it <- nwsIFetchTry(ws, 'x', NA)
it() # returns the value 10
it() # returns NA

## End(Not run)
```

---

nwsIFind

*Iterate Through Stored Values of a netWorkspace Variable*

---

## Description

Create a function that acts as a non-destructive iterator over the values of the specified variable.

## Usage

```
## S4 method for signature 'netWorkspace'
nwsIFind(.Object, xName)
```

## Arguments

.Object	a netWorkspace class object.
xName	character string specifying the name of the variable to be fetched.

## Details

The iterator function returned by the nwsIFind method takes no arguments, and works somewhat like the nwsFind method, specified with the same arguments that were passed to nwsIFind. The difference is that the nwsFind method cannot iterate through the values of a variable; it always returns the same value until the variable is modified. The iterator function, however, maintains some state that allows it to see subsequent values. Each time the iterator function is called, it returns the next value in the variable. Once all values in the variable have been returned, the iterator function blocks, waiting for a new value to be stored in the variable.

## See Also

[nwsFind](#), [nwsIFindTry](#)

## Examples

```
## Not run:
ws <- netWorkspace('nws example')
nwsStore(ws, 'x', 1)
nwsStore(ws, 'x', 2)
it <- nwsIFind(ws, 'x')
```

```

it() # returns the value 1
it() # returns the value 2
it() # blocks until another process stores a value in the variable

## End(Not run)

```

---

nwsIFindTry

*Iterate Through Stored Values of a netWorkspace Variable*


---

### Description

Create a function that acts as a non-destructive iterator over the values of the specified variable.

### Usage

```

## S4 method for signature 'netWorkspace'
nwsIFindTry(.Object, xName, defaultVal=NULL)

```

### Arguments

<code>.Object</code>	a <code>netWorkspace</code> class object.
<code>xName</code>	character string specifying the name of the variable to be fetched.
<code>defaultVal</code>	value to return if <code>xName</code> is not found in the <code>netWorkspace</code> .

### Details

The iterator function returned by the `nwsIFindTry` method takes no arguments, and works somewhat like the `nwsFindTry` method, specified with the same arguments that were passed to `nwsIFindTry`. The difference is that the `nwsFindTry` method cannot iterate through the values of a variable; it always returns the same value until the variable is modified. The iterator function, however, maintains some state that allows it to see subsequent values. Each time the iterator function is called, it returns the next value in the variable. Once all values in the variable have been returned, the iterator function returns `defaultVal`. However, when new values are stored into the variable, the iterator function will return them, picking right up where it left off.

### See Also

[nwsFindTry](#), [nwsIFind](#)

### Examples

```

## Not run:
ws <- netWorkspace('nws example')
nwsStore(ws, 'x', 1)
nwsStore(ws, 'x', 2)
it <- nwsIFindTry(ws, 'x', NA)
it() # returns the value 1
it() # returns the value 2

```

```

it() # returns the value NA
nwsStore(ws, 'x', 3)
it() # returns the value 3
it() # returns the value NA

## End(Not run)

```

---

nwsListVars                      *List Variables in a netWorkspace*

---

## Description

List variables in a netWorkspace.

## Usage

```

## S4 method for signature 'netWorkspace'
nwsListVars(.Object, wsName='', showDataFrame=FALSE)

```

## Arguments

.Object                      a netWorkspace class object.  
wsName                      character string specifying the name of the netWorkspace.  
showDataFrame              logical flag specifying whether to show result in data frame or string.

## Details

Return listing of variables in the netWorkspace with the name passed by wsName argument. If wsName is empty, then return variables in the netWorkspace represented by .Object.

The return values from nwsListVars can be represented in a string or a data frame. If showDataFrame is set to FALSE (the default), then the listing is returned in a string. To see list output clearly, use:  
write(nwsListVars(.Object), stdout())

If showDataFrame is set to TRUE, then the listing is returned in a data frame with these fields: Variables, NumValues, NumFetchers, NumFinders, and Mode.

## Examples

```

## Not run:
# example 1
ws <- netWorkspace('nws example')
write(nwsListVars(ws), stdout())

## End(Not run)

```

---

nwsListWss	<i>List All netWorkSpaces</i>
------------	-------------------------------

---

**Description**

List all netWorkSpaces in the netWorkSpaces server.

**Usage**

```
## S4 method for signature 'nwsServer'  
nwsListWss(.Object, showDataFrame=FALSE)
```

**Arguments**

.Object            a nwsServer class object  
showDataFrame    show result in data frame or string

**Details**

By default, showDataFrame is set to FALSE, which means the return value is a text string containing a list of workspaces in the netWorkSpaces server. To see list output clearly, use:

```
write(nwsListWss(.Object), stdout())
```

If showDataFrame is set to TRUE, then the return value is a data frame with these columns: Owned, Name, Owner, Persistent, NumVariables, and Variables.

**Examples**

```
## Not run:  
# example 1  
nwss <- nwsServer()  
ws1 <- nwsOpenWs(nwss, 'my space')  
ws2 <- nwsOpenWs(nwss, 'other space')  
write(nwsListWss(nwss), stdout())  
  
# example 2  
# retrieve all workspace names  
df <- nwsListWss(nwss, showDataFrame=TRUE)  
df$Name  
$"1"  
[1] "__default"  
  
$"2"  
[1] "my space"  
  
$"3"  
[1] "other space"  
  
## End(Not run)
```

---

nwsMktempWs	<i>Create a Temporary netWorkspace</i>
-------------	--

---

**Description**

Create a unique temporary netWorkspace using the template string.

**Usage**

```
## S4 method for signature 'nwsServer'
nwsMktempWs(.Object, wsNameTemplate)
```

**Arguments**

```
.Object          a nwsServer class object
wsNameTemplate  template for the netWorkspace name
```

**Details**

nwsMktempWs(nwss, wsNameTemplate) returns the name of a temporary space created on the netWorkSpaces server. The template should contain a %d-like construct which will be replaced by a serial counter maintained by the server to generate a unique new netWorkspace name. The user must then invoke nwsOpenWs or nwsUseWs with this name to create an object to access this workspace. WsNameTemplate defaults to \_\_Rws\_\_%010d.

**Examples**

```
## Not run:
s <- nwsServer()
tempWsName <- nwsMktempWs(s, 'temp_%d')
ws <- nwsOpenWs(s, tempWsName)

## End(Not run)
```

---

nwsOpenWs	<i>Create and Own a netWorkspace</i>
-----------	--------------------------------------

---

**Description**

Create and own a netWorkspace, if it does not already exist. If the netWorkspace already exists, but no one owns it, then caller claims ownership of the netWorkspace. If the netWorkspace already exists and someone already claimed the ownership of it, then caller simply makes connection to the netWorkspace.

**Usage**

```
## S4 method for signature 'nwsServer'  
nwsOpenWs(.Object, wsName, space=NULL, ...)
```

**Arguments**

.Object	a nwsServer class object
wsName	name of the netWorkspace to be created
space	a netWorkspace class object
...	optional arguments related to the persistent state of a netWorkspace

**Details**

If the space argument is not provided, nwsOpenWs creates a shared netWorkspace with the name wsName on the server represented by .Object. Otherwise, use the existing netWorkspace object provided by the space argument.

The optional argument persistent is a logical value indicating whether the shared netWorkspace is persistent or not. If the netWorkspace is persistent (persistent=TRUE), then the netWorkspace is not deleted when the owner of the shared netWorkspace exits. Otherwise, the netWorkspace is deleted when its owner exits.

**See Also**

[nwsUseWs](#)

**Examples**

```
## Not run:  
# example 1  
nwss <- nwsServer()  
ws <- nwsOpenWs(nwss, "nws example")  
  
# example 2  
xs <- nwsOpenWs(nwss, wsName='nws example', space=ws)  
  
# example 3  
ys <- nwsOpenWs(nwss, "persistent space", persistent=TRUE)  
  
## End(Not run)
```

**Description**

Reports package name and version information.

**Usage**

```
nwsPkgInfo()
```

**Details**

nwsPkgInfo returns details of the NetWorkSpaces package. The name can be either "nws" or "nwsPro".

**Value**

A character vector with with fields

name	The package name.
version	The version.

**Examples**

```
nwsPkgInfo()
```

---

nwsServer

*Class Representing NetWorkSpaces Server*


---

**Description**

Class representing nwsServer.

**Objects from the Class**

Objects can be created by calls of the form  
nwsServer(serverHost, port)  
or  
new("nwsServer", serverHost, port).

serverHost: server host name. Default value is "localhost".

port: server port number. Default value is 8765.

**Slots**

cookieProtocol: Currently not used.

nwsSocket: Object of class "ANY" representation of the socket connection to the server.

port: Object of class "numeric" representation of the server port number.

serverHost: Object of class "character" representation of the server host name.

**Methods**

**initialize** signature(.Object = "nwsServer"): nwsServer class constructor.

**nwsDeleteWs** signature(.Object = "nwsServer"): delete a netWorkspace from the server.

**nwsListWss** signature(.Object = "nwsServer"): list all netWorkSpaces in the server.

**nwsMktempWs** signature(.Object = "nwsServer"): create a unique temporary workspace using the default or specified template.

**nwsOpenWs** signature(.Object = "nwsServer"): create and owned a netWorkspace.

**nwsUseWs** signature(.Object = "nwsServer"): connect to a netWorkspace but does not claim ownership.

**Examples**

```
## Not run:
# example 1
nwss = nwsServer()
# Or,
nwss = new("nwsServer")

# example 2
nwss = nwsServer(serverHost="node1", port=5555)

## End(Not run)
```

---

nwsServerObject

*Return Server Object Associated with netWorkspace*


---

**Description**

Return the nwsServer object associated with a netWorkspace.

**Usage**

```
## S4 method for signature 'netWorkspace'
nwsServerObject(.Object)
```

**Arguments**

.Object            a netWorkspace class object

**Examples**

```
## Not run:
ws = netWorkspace('nws example')
nwsServerObject(ws)

## End(Not run)
```

---

nwsStore	<i>Associate a Value with a Variable in netWorkspace</i>
----------	--

---

## Description

Store value associated with a variable in the shared netWorkspace.

## Usage

```
## S4 method for signature 'netWorkspace'  
nwsStore(.Object, xName, xVal)
```

## Arguments

.Object	a netWorkspace class object.
xName	character string to be used as the name of the stored variable.
xVal	value to be stored.

## Details

The nwsStore method associates the value xVal with the variable xName in the shared netWorkspace .Object, thereby making the value available to all the distributed R processes. If a mode has not already been set for xName, 'fifo' will be used (see [nwsDeclare](#)).

Note that, by default ('fifo' mode), nwsStore is not idempotent: repeating nwsStore (nws, xName, xVal) will add additional values to the set of values associated with the variable named xName. See the examples below for details.

## See Also

[nwsDeclare](#)

## Examples

```
## Not run:  
ws <- netWorkspace('nws example')  
  
# To store value 5 bound to variable 'x' on the netWorkspace 'ws'  
# (If 'x' was declared, then its mode is inherited,  
# otherwise 'x' uses the default mode 'fifo')  
nwsStore(ws, 'x', 5)  
  
# store 10 values associate with variable y to the netWorkspace  
for (i in 1:10)  
  nwsStore(ws, 'y', i)  
  
# retrieve 10 values associate with variable y from the netWorkspace  
for (i in 1:10)  
  print(nwsFetch(ws, 'y'))
```

```
## End(Not run)
```

---

nwsStoreFile	<i>Read a Value from File and Store in netWorkspace</i>
--------------	---

---

## Description

Store a new value into a variable in the workspace from a file.

## Usage

```
## S4 method for signature 'netWorkspace'  
nwsStoreFile(.Object, xName, fObj, n=0)
```

## Arguments

.Object	a netWorkspace class object.
xName	character string specifying the name of the variable to be stored
fObj	a file object or character string specifying file from which to read data to store in the variable.
n	Number of bytes to write. A value of zero means to write all the data in the file.

## Details

The nwsStoreFile method works like nwsStore, except that the value to store in the netWorkspace variable comes from a file. If fObj is a character string, nwsStoreFile calls file to obtain a file connection which is opened for the duration of the method.

## See Also

[nwsStore](#)

## Examples

```
## Not run:  
ws <- netWorkspace('nws example')  
nwsStore(ws, 'x', 'Hello, world\n')  
nwsFindFile(ws, 'x', 'hello.txt')  
nwsStoreFile(ws, 'hello', 'hello.txt')  
  
## End(Not run)
```

---

nwsUseWs	<i>Connect to a netWorkspace</i>
----------	----------------------------------

---

**Description**

Connects to a netWorkspace but does not claim ownership. If the netWorkspace does not exist, it will be created, but no ownership will be claimed.

**Usage**

```
## S4 method for signature 'nwsServer'
nwsUseWs(.Object, wsName, space=NULL, ...)
```

**Arguments**

.Object	a nwsServer class object
wsName	name of the netWorkspace to open
space	a netWorkspace class object
...	additional arguments create=TRUE to create the workspace if it does not exist and persistent=TRUE to make the workspace persistent.

**See Also**

[nwsOpenWs](#)

**Examples**

```
## Not run:
nwss <- nwsServer()
ws <- nwsUseWs(nwss, "nws example")

## End(Not run)
```

---

nwsVariable	<i>Create an Active Binding for a netWorkspace Variable</i>
-------------	---

---

**Description**

nwsVariable creates a variable in an R workspace that mirrors a variable in a netWorkspace. This allows standard R operations (get, assign) to be used to share data between R programs running on different machines.

**Usage**

```
## S4 method for signature 'netWorkspace'
nwsVariable(.Object, xName, mode=c('fifo','lifo','multi','single'),env=parent.frame(),force=FALSE,qu
```

## Arguments

.Object	a netWorkspace class object.
xName	name of variable to be declared.
mode	mode of the variable, see details.
env	environment in which to define active binding.
force	logical; if TRUE, an existing binding will be overwritten.
quietly	logical; if TRUE, no warnings are issued.

## Details

nwsVariable is built on top of the R makeActiveBinding function. It is experimental, but we have found that it is very useful for introducing people to the concept of netWorkspace variables. It's not clear that this API is ever preferable to nwsStore, nwsFetch, nwsFind for real programs, however.

The mode of the variable controls what happens when a variable is accessed. If the mode is 'single', then all accesses use the nwsFind operation. If the mode is 'fifo', 'lifo', or 'multi', then all accesses use the nwsFetch operation. Assigning a value to an nwsVariable always uses the nwsStore operation.

## Examples

```
## Not run:
# create a netWorkspace
ws = netWorkspace('nws example')

# create a variable in the local R workspace that is linked to
# a netWorkspace variable
nwsVariable(ws, 'x', 'single')

x <- 0
x <- 999 # overwrites the 0
x <- 3.14159 # overwrites the 999
x # returns 3.14159
x # returns 3.14159
x # returns 3.14159

# create a 'fifo' mode variable
nwsVariable(ws, 'message', 'fifo')

message <- 1
message <- 2
message <- 3
message # returns 1
message # returns 2
message # returns 3

## End(Not run)
```

---

nwsWsName	<i>Return Name of a netWorkspace</i>
-----------	--------------------------------------

---

**Description**

Return name of a netWorkspace.

**Usage**

```
## S4 method for signature 'netWorkspace'  
nwsWsName(.Object)
```

**Arguments**

.Object            a netWorkspace class object

**Examples**

```
## Not run:  
ws = netWorkspace('nws example')  
nwsWsName(ws)  
  
## End(Not run)
```

---

rankCount	<i>Return a Rank Count from a Sleigh</i>
-----------	--

---

**Description**

Return the rankCount in sleigh. If rankCount=-1, then the worker group has closed, and no more workers may join the group.

**Usage**

```
## S4 method for signature 'sleigh'  
rankCount(.Object)
```

**Arguments**

.Object            a sleigh class object

**Examples**

```
## Not run:  
# simple hello world  
s = sleigh()  
rankCount(s)  
  
## End(Not run)
```

---

`rshcmd`*Sleigh Auxiliary Function*

---

## Description

This function is used by the sleigh constructor when starting workers on remote nodes using the rsh command. Note that it doesn't actually start any workers directly: it simply returns the program name and arguments to start a worker on the specified node.

## Usage

```
rshcmd(host, options)
```

## Arguments

<code>host</code>	Name of the worker machine to be started.
<code>options</code>	An environment or list that contains information used to construct the command. See <code>defaultSleighOptions</code> for more information.

## Details

`rshcmd` currently uses the following options: `user`, `wrapperDir`, and `python`. The `user` option is the most useful.

## Value

A character mode vector, whose first element is the command that will be executed to start the worker, and whose subsequent elements are the arguments to the command.

## Note

`rshcmd` should only be used if `ssh` is not available, since `ssh` is a much more secure, modern replacement for `rsh`.

## See Also

[sshcmd](#), [sleigh](#)

## Examples

```
## Not run:
# Create a sleigh with workers on nodes n1 and n2 started via rsh:
s <- sleigh(launch=rshcmd, nodeList=c('n1', 'n2'))

# Same as previous, but as user 'frank':
s <- sleigh(launch=rshcmd, nodeList=c('n1', 'n2'), user='frank')

# Specify an alternate python interpreter and see the command
```

```
# and arguments that rshcmd returns:
defaultSleighOptions$python <- '/usr/local/bin/python'
rshcmd('node1', defaultSleighOptions)

## End(Not run)
```

---

sleigh

*Class "sleigh"*


---

## Description

Represents a collection of R processes used to execute tasks.

The sleigh allows R functions to be executed in parallel using the `eachElem` and `eachWorker` methods.

The sleigh workers are started when the sleigh object is constructed. When tasks are submitted to the sleigh, using the `eachWorker` and `eachElem` methods, the workers execute the tasks and return the results. When the `stopSleigh` method is called, the workers are stopped.

A given R program can create multiple sleigh objects, each of which will have its own set of workers. This can be useful if tasks have different requirements. For example, you could create a Linux sleigh and a Windows sleigh, and submit Windows-specific tasks only to your Windows sleigh.

## Objects from the Class

Objects can be created by calls of the form

```
sleigh(...)
```

or

```
new("sleigh",...)
```

where ... can be one or more of the following named arguments:

**nodeList:** a list of hosts on which workers will be created. This argument is ignored when `launch='local'`. The default value is `c('localhost', 'localhost', 'localhost')`.

**workerCount:** number of workers that will be created. This argument is only relevant when `launch='local'`. The default value is 3.

**launch:** method to launch remote workers. This can be set to the strings `'local'` (the default) or `'web'`, or to a function object.

The function is called once for each worker listed in `nodeList`. It is passed two arguments: a name from `nodeList`, and an environment object that was constructed by merging `defaultSleighOptions` with the arguments that were passed to the sleigh constructor. The function should return a character vector, where the first element is the command to execute, and the subsequent elements are the command arguments. For example, the function could return the vector `c('ssh', '-f', 'host')`, where `'host'` is the first argument to the launch function. This isn't the complete command to be executed; it's the 'remote execution' portion of the command. The sleigh constructor will add the rest of the command based on the `scriptExec` argument. Note that the command is expected to return after launching the worker. That is why the `ssh -f` option is used in the example.

**nwsHost:** host name of the netWorkSpaces server. Default is the machine where sleigh starts up.

**nwsPort:** port number of the netWorkSpaces server. Default is 8765.

**scriptExec:** command to execute worker script. Default uses `scriptcmd` function on Windows, and uses `envcmd` function on other platforms.

**scriptDir:** location of the sleigh worker script. Default is the bin directory under the nws package location. If library cannot be found, then use current working directory.

**scriptName:** worker script file name. Default is `RNWSSleighWorker.py` on Windows, and `RNWSSleighWorker.sh` on other platforms.

**workingDir:** worker's working directory. Default is master's current working directory.

**logDir:** location where log files will be stored. Default is NULL.

**outfile:** remote workers' standard errors will be redirected to this file. Default is NULL.

**wsNameTemplate:** template name to create sleigh workspace. Default is `'sleigh_ride_%010d'`.

**user:** user name used for remote execution. Default is NULL.

**verbose:** a logical value indicating whether to print out debug messages. Default is FALSE.

## Methods

**initialize** signature(`.Object = "sleigh"`): sleigh class constructor.

**eachElem** signature(`.Object = "sleigh"`): evaluate the given function with multiple argument sets using the workers in sleigh.

**eachWorker** signature(`.Object = "sleigh"`): evaluate the given function exactly once for each worker in sleigh.

**rankCount** signature(`.Object = "sleigh"`): get sleigh's rankCount.

**status** signature(`.Object = "sleigh"`): return the status of the sleigh.

**stopSleigh** signature(`.Object = "sleigh"`): shut down workers and remove sleigh workspace.

**workerCount** signature(`.Object = "sleigh"`): get number of workers started in sleigh.

## Details

There are six different standard launch types (`'local'`, `sshcmd`, `rshcmd`, `lsfcmd`, `sshforwardcmd`, and `'web'`) to tailor the client's working environment. This is done by setting launch variable to a function (`sshcmd`, `rshcmd`, `sshforwardcmd` or `lsfcmd`) or a string (`'local'` and `'web'`). See the examples section.

## Examples

```
## Not run:
# Default option: create three sleigh workers on local host:
s <- sleigh()
# which is equivalent to:
s <- sleigh(launch='local')

# Create sleigh workers on multiple machines using SSH:
s <- sleigh(nodeList=c('n1', 'n2', 'n3'), launch=sshcmd)
```

```

# Use the LSF bsub command to launch ten workers:
s <- sleigh(nodeList=rep('fake', 10), launch=lsfcmd)

# Use web launch:
s <- sleigh(launch='web')

## End(Not run)

```

---

sleighPending	<i>Class "sleighPending"</i>
---------------	------------------------------

---

## Description

Class representing sleighPending.

## Details

This class object is usually obtained from non-blocking eachElem or non-blocking eachWorker.

## Objects from the Class

Objects can be created by calls of the form  
sleighPending(nws, numTasks, bn, ss)  
or  
new("sleighPending", nws, numTasks, bn, ss).

**nws:** netWorkSpace class object.

**numTasks:** number of submitted tasks.

**bn:** barrier names.

**ss:** sleigh state.

## Slots

**nws:** Object of class "netWorkSpace".

**numTasks:** The number of pending tasks in sleigh.

**numSubmitted:** The number of tasks submitted.

**accumulator:** Function used as accumulator.

**barrierName:** Character string giving the barrier name.

**sleighState:** Object of class "environment" representing the sleigh state.

**state:** Object of class "environment" representing the sleighPending state.

## Methods

**initialize** signature(.Object = "sleighPending"): sleighPending class constructor.

**checkSleigh** signature(.Object = "sleighPending"): returns the number of results yet to be generated for the pending sleigh job.

**waitSleigh** signature(.Object = "sleighPending"): wait and block for the results to be generated for the pending sleigh job.

**See Also**

[eachWorker](#), [eachElem](#)

---

 sshcmd

*Sleigh Auxiliary Function*


---

**Description**

This function is used by the sleigh constructor when starting workers on remote nodes using the ssh command. Note that it doesn't actually start any workers directly: it simply returns the program name and arguments to start a worker on the specified node.

**Usage**

```
sshcmd(host, options)
sshforwardcmd(host, options)
```

**Arguments**

host	Name of the worker machine to be started.
options	An environment or list that contains information used to construct the command. See defaultSleighOptions for more information.

**Details**

sshcmd currently uses the following options: user, wrapperDir, workerWrapper, and python. The user option is the most useful.

sshforward works like sshcmd, but it makes use of the ssh -R argument to tunnel the NWS server connection back to the master machine. It uses the options nwsHostRemote and nwsPortRemote to determine what bind address and port should be forwarded back to the master machine.

Note that when using sshforwardcmd, you must specify a different value of nwsHostRemote, usually 'localhost'.

**Value**

A character vector, whose first element is the command that will be executed to start the worker, and whose subsequent elements are the arguments to the command.

**Note**

sshcmd and sshforwardcmd are not intended to be called directly by the user. They are called by the sleigh constructor when specified via the launch argument. You may want to execute it when debugging your sleigh option settings, but that can also be accomplished by setting the verbose argument to TRUE.

**See Also**

[sleigh, defaultSleighOptions](#)

**Examples**

```
## Not run:
# Create a sleigh with workers on nodes n1 and n2 started via ssh:
s <- sleigh(launch=sshcmd, nodeList=c('n1', 'n2'))

# Same as previous, but as user 'frank':
s <- sleigh(launch=sshcmd, nodeList=c('n1', 'n2'), user='frank')

# Create two tunnels from workers n1 and n2 back the nws server
# on the local machine:
s <- sleigh(launch=sshforwardcmd, nodeList=c('n1', 'n2'),
            nwsHostRemote='localhost')

# Same as the previous example, but use port 9876 in case either
# worker machine already has an nws server bound to port 8765:
s <- sleigh(launch=sshforwardcmd, nodeList=c('n1', 'n2'),
            nwsHostRemote='localhost', nwsPortRemote=9876)

## End(Not run)
```

---

status

*Check Status of Sleigh Workers*

---

**Description**

Check status of sleigh workers.

**Usage**

```
## S4 method for signature 'sleigh'
status(.Object, closeGroup=FALSE, timeout=0)
```

**Arguments**

.Object	a sleigh class object
closeGroup	a logical value indicating whether to close the worker group
timeout	timeout value measure in seconds

**Details**

The argument `closeGroup` is set to `FALSE` by default. If `closeGroup` is set to `FALSE`, then all workers must wait for the others to start before they can start working on tasks. If `closeGroup` is set to `TRUE`, no new workers may join the group after the timeout value has expired. Once the group is closed, all launched workers can start working on tasks.

The `timeout` argument indicates how long to wait and check on the status of workers.

**Value**

a list that contains two values:

numWorkers	the number of workers started
closed	If closed is zero, then some workers failed to start. If closed is one, then either all workers have started or closeGroup is set to TRUE.

**Examples**

```
## Not run:
# example 1
# one available machine and one non-existent machine
s <- sleigh(c('localhost', 'noname'))
slist <- status(s)
# slist$numWorkers = the number of worker started
# slist$closed = whether the worker group is closed or not.

# example 2
# check the status of worker group after 20 seconds
slist <- status(s, timeout=20)

# example 3
# close the group after 10 seconds, regardless of whether
# all workers have started up successfully.
slist <- status(s, closeGroup=TRUE, timeout=10)

## End(Not run)
```

---

stopSleigh

*Stop a Sleigh*


---

**Description**

Shut down sleigh workers and delete sleigh workspace.

**Usage**

```
## S4 method for signature 'sleigh'
stopSleigh(.Object)
```

**Arguments**

.Object            a sleigh class object

**Details**

This function is called by `close` method.

**See Also**[close](#)**Examples**

```
## Not run:  
s = sleigh()  
stopSleigh(s)  
  
## End(Not run)
```

---

unexport

*sleigh Class Method*

---

**Description**

Remove a variable from the scope of the workers or specified worker.

**Usage**

```
## S4 method for signature 'sleigh'  
unexport(.Object, xName, worker=NULL)
```

**Arguments**

<code>.Object</code>	a sleigh class object.
<code>xName</code>	a variable name (given as a quoted string in the function call).
<code>worker</code>	integer rank of the worker where the variable is defined. The rank values range from 0 to <code>workerCount - 1</code> . By default, the variable is removed from all workers in the sleigh.

**See Also**[export](#)**Examples**

```
## Not run:  
s <- sleigh()  
x <- 6  
export(s, 'x', x)  
unexport(s, 'x')  
  
## End(Not run)
```

---

waitSleigh	<i>Wait for Results from Sleigh</i>
------------	-------------------------------------

---

**Description**

Wait and block for the results to be generated for the pending sleigh job. If the job has completed, then waitSleigh returns immediately with the generated results.

**Usage**

```
## S4 method for signature 'sleighPending'
waitSleigh(.Object)
```

**Arguments**

.Object            a sleighPending class object

**Details**

The sleighPending class object, .Object, is usually obtained from the return value of non-blocking eachElem or non-blocking eachWorker.

**See Also**

[eachWorker](#), [eachElem](#)

**Examples**

```
## Not run:
s = sleigh()
eo = list(blocking=0)
sp = eachWorker(s, function() {Sys.sleep(100)}, eo=eo)
waitSleigh(sp) # wait on workers. Each worker sleeps for 100 seconds

## End(Not run)
```

---

workerCount	<i>Find the Number of Workers in a Sleigh</i>
-------------	---

---

**Description**

Return number of Sleigh workers that have successfully started.

**Usage**

```
## S4 method for signature 'sleigh'
workerCount(.Object)
```

**Arguments**

.Object            a sleigh class object

**Examples**

```
## Not run:  
# simple hello world  
s = sleigh()  
workerCount(s)  
  
## End(Not run)
```

---

workerInfo	<i>sleigh Class Method</i>
------------	----------------------------

---

**Description**

Return information about sleigh workers.

**Usage**

```
## S4 method for signature 'sleigh'  
workerInfo(.Object)
```

**Arguments**

.Object            a sleigh class object.

**Value**

A data frame with a row for each worker, with column names 'host', 'os', 'pid', 'R', 'nws', 'rank', and 'logfile'.

**Examples**

```
## Not run:  
s <- sleigh()  
status(s, TRUE, 20)  
workerInfo(s)  
  
## End(Not run)
```

# Index

- \*Topic **classes**
    - netWorkspace, 15
    - nwsServer, 35
    - sleigh, 43
    - sleighPending, 45
  - \*Topic **datasets**
    - germandata, 13
  - \*Topic **debugging**
    - isClosure, 13
  - \*Topic **environment**
    - defaultSleighOptions, 6
  - \*Topic **methods**
    - checkSleigh, 4
    - close, 5
    - eachElem, 6
    - eachWorker, 10
    - export, 12
    - netWorkspace, 15
    - netWorkspaceObject, 17
    - nwsClose, 17
    - nwsDeclare, 18
    - nwsDeleteVar, 18
    - nwsDeleteWs, 19
    - nwsFetch, 20
    - nwsFetchFile, 21
    - nwsFetchTry, 22
    - nwsFetchTryFile, 23
    - nwsFind, 24
    - nwsFindFile, 24
    - nwsFindTry, 25
    - nwsFindTryFile, 26
    - nwsIFetch, 27
    - nwsIFetchTry, 28
    - nwsIFind, 29
    - nwsIFindTry, 30
    - nwsListVars, 31
    - nwsListWss, 32
    - nwsMktempWs, 33
    - nwsOpenWs, 33
    - nwsServerObject, 36
    - nwsStore, 37
    - nwsStoreFile, 38
    - nwsUseWs, 39
    - nwsVariable, 39
    - nwsWsName, 41
    - rankCount, 41
    - sleighPending, 45
    - status, 47
    - stopSleigh, 48
    - unexport, 49
    - waitSleigh, 50
    - workerCount, 50
    - workerInfo, 51
  - \*Topic **package**
    - nws-package, 3
  - \*Topic **utilities**
    - batchNodeList, 3
    - isClosure, 13
    - lsfcmd, 14
    - nwsPkgInfo, 34
    - rshcmd, 42
    - sshcmd, 46
- batchNodeList, 3
- checkSleigh, 4
- checkSleigh, sleighPending-method (checkSleigh), 4
- checkSleigh-methods (checkSleigh), 4
- close, 5, 48, 49
- close, ANY-method (close), 5
- close, nwsServer-method (close), 5
- close, sleigh-method (close), 5
- close-methods (close), 5
- defaultSleighOptions, 6, 47
- eachElem, 4, 6, 11, 46, 50
- eachElem, sleigh-method (eachElem), 6

- eachElem-methods (eachElem), [6](#)
- eachWorker, [4](#), [9](#), [10](#), [46](#), [50](#)
- eachWorker, sleigh-method (eachWorker), [10](#)
- eachWorker-methods (eachWorker), [10](#)
- export, [12](#), [49](#)
- export, sleigh-method (export), [12](#)
- export-methods (export), [12](#)
  
- germandata, [13](#)
  
- initialize (sleigh), [43](#)
- initialize, netWorkspace-method (netWorkspace), [15](#)
- initialize, nwsServer-method (nwsServer), [35](#)
- initialize, sleigh-method (sleigh), [43](#)
- initialize, sleighPending-method (sleighPending), [45](#)
- isClosure, [13](#)
  
- lsfcmd, [14](#)
- lsfNodeList (batchNodeList), [3](#)
  
- netWorkspace, [15](#)
- netWorkspace-class (netWorkspace), [15](#)
- netWorkspaceObject, [17](#)
- netWorkspaceObject, sleigh-method (netWorkspaceObject), [17](#)
- netWorkspaceObject-methods (netWorkspaceObject), [17](#)
- nws (nws-package), [3](#)
- nws-package, [3](#)
- nwsClose, [17](#)
- nwsClose, netWorkspace-method (nwsClose), [17](#)
- nwsClose-methods (nwsClose), [17](#)
- nwsDeclare, [18](#), [20](#), [22](#), [24–27](#), [37](#)
- nwsDeclare, netWorkspace-method (nwsDeclare), [18](#)
- nwsDeclare-methods (nwsDeclare), [18](#)
- nwsDeleteVar, [18](#)
- nwsDeleteVar, netWorkspace-method (nwsDeleteVar), [18](#)
- nwsDeleteVar-methods (nwsDeleteVar), [18](#)
- nwsDeleteWs, [19](#)
- nwsDeleteWs, nwsServer-method (nwsDeleteWs), [19](#)
- nwsDeleteWs-methods (nwsDeleteWs), [19](#)
  
- nwsFetch, [20](#), [21](#), [22](#), [24](#), [28](#)
- nwsFetch, netWorkspace-method (nwsFetch), [20](#)
- nwsFetch-methods (nwsFetch), [20](#)
- nwsFetchFile, [21](#), [25](#)
- nwsFetchFile, netWorkspace-method (nwsFetchFile), [21](#)
- nwsFetchFile-methods (nwsFetchFile), [21](#)
- nwsFetchTry, [20](#), [22](#), [23](#), [28](#)
- nwsFetchTry, netWorkspace-method (nwsFetchTry), [22](#)
- nwsFetchTry-methods (nwsFetchTry), [22](#)
- nwsFetchTryFile, [23](#), [27](#)
- nwsFetchTryFile, netWorkspace-method (nwsFetchTryFile), [23](#)
- nwsFetchTryFile-methods (nwsFetchTryFile), [23](#)
- nwsFind, [24](#), [25](#), [26](#), [29](#)
- nwsFind, netWorkspace-method (nwsFind), [24](#)
- nwsFind-methods (nwsFind), [24](#)
- nwsFindFile, [21](#), [24](#)
- nwsFindFile, netWorkspace-method (nwsFindFile), [24](#)
- nwsFindFile-methods (nwsFindFile), [24](#)
- nwsFindTry, [25](#), [27](#), [30](#)
- nwsFindTry, netWorkspace-method (nwsFindTry), [25](#)
- nwsFindTry-methods (nwsFindTry), [25](#)
- nwsFindTryFile, [23](#), [26](#)
- nwsFindTryFile, netWorkspace-method (nwsFindTryFile), [26](#)
- nwsFindTryFile-methods (nwsFindTryFile), [26](#)
- nwsIFetch, [27](#), [28](#)
- nwsIFetch, netWorkspace-method (nwsIFetch), [27](#)
- nwsIFetch-methods (nwsIFetch), [27](#)
- nwsIFetchTry, [28](#), [28](#)
- nwsIFetchTry, netWorkspace-method (nwsIFetchTry), [28](#)
- nwsIFetchTry-methods (nwsIFetchTry), [28](#)
- nwsIFind, [29](#), [30](#)
- nwsIFind, netWorkspace-method (nwsIFind), [29](#)
- nwsIFind-methods (nwsIFind), [29](#)
- nwsIFindTry, [29](#), [30](#)
- nwsIFindTry, netWorkspace-method

- (nwsIFindTry), 30
- nwsIFindTry-methods (nwsIFindTry), 30
- nwsListVars, 31
- nwsListVars, netWorkspace-method (nwsListVars), 31
- nwsListVars-methods (nwsListVars), 31
- nwsListWss, 32
- nwsListWss, nwsServer-method (nwsListWss), 32
- nwsListWss-methods (nwsListWss), 32
- nwsMktempWs, 33
- nwsMktempWs, nwsServer-method (nwsMktempWs), 33
- nwsMktempWs-methods (nwsMktempWs), 33
- nwsOpenWs, 33, 39
- nwsOpenWs, nwsServer-method (nwsOpenWs), 33
- nwsOpenWs-methods (nwsOpenWs), 33
- nwsPkgInfo, 34
- nwsPro (nws-package), 3
- nwsPro-package (nws-package), 3
- nwsServer, 35
- nwsServer-class (nwsServer), 35
- nwsServerObject, 36
- nwsServerObject, netWorkspace-method (nwsServerObject), 36
- nwsServerObject-methods (nwsServerObject), 36
- nwsStore, 37, 38
- nwsStore, netWorkspace-method (nwsStore), 37
- nwsStore-methods (nwsStore), 37
- nwsStoreFile, 38
- nwsStoreFile, netWorkspace-method (nwsStoreFile), 38
- nwsStoreFile-methods (nwsStoreFile), 38
- nwsUseWs, 34, 39
- nwsUseWs, nwsServer-method (nwsUseWs), 39
- nwsUseWs-methods (nwsUseWs), 39
- nwsVariable, 39
- nwsVariable, netWorkspace-method (nwsVariable), 39
- nwsVariable-methods (nwsVariable), 39
- nwsWsName, 41
- nwsWsName, netWorkspace-method (nwsWsName), 41
- nwsWsName-methods (nwsWsName), 41
- pbsNodeList (batchNodeList), 3
- rankCount, 41
- rankCount, sleigh-method (rankCount), 41
- rankCount-methods (rankCount), 41
- rshcmd, 42
- sgeNodeList (batchNodeList), 3
- sleigh, 4, 6, 14, 42, 43, 47
- sleigh-class (sleigh), 43
- sleighPending, 9, 11, 45
- sleighPending-class (sleighPending), 45
- sleighPro (sleigh), 43
- sshcmd, 42, 46
- sshforwardcmd (sshcmd), 46
- status, 47
- status, sleigh-method (status), 47
- status-methods (status), 47
- stopSleigh, 48
- stopSleigh, sleigh-method (stopSleigh), 48
- unexport, 12, 49
- unexport, sleigh-method (unexport), 49
- unexport-methods (unexport), 49
- waitSleigh, 50
- waitSleigh, sleighPending-method (waitSleigh), 50
- waitSleigh-methods (waitSleigh), 50
- workerCount, 50
- workerCount, sleigh-method (workerCount), 50
- workerCount-methods (workerCount), 50
- workerInfo, 51
- workerInfo, sleigh-method (workerInfo), 51
- workerInfo-methods (workerInfo), 51