Package 'startR'

February 5, 2019

Title Automatically Retrieve Multidimensional Distributed Data Sets

Version 0.1.1

Description Tool to automatically fetch, transform and arrange subsets of multidimensional data sets (collections of files) stored in local and/or remote file systems or servers, using multicore capabilities where possible. The tool provides an interface to perceive a collection of data sets as a single large multidimensional data array, and enables the user to request for automatic retrieval, processing and arrangement of subsets of the large array. Wrapper functions to add support for custom file formats can be plugged in/out, making the tool suitable for any research field where large multidimensional data sets are involved.

Depends R (>= 3.2.0)

Imports abind, bigmemory, future, multiApply (>= 2.1.1), parallel

Suggests easyNCDF, s2dverification

License LGPL-3

URL https://earth.bsc.es/gitlab/es/startR/

BugReports https://earth.bsc.es/gitlab/es/startR/issues

LazyData true

NeedsCompilation no

Author BSC-CNS [aut, cph],
Nicolau Manubens [aut, cre],
Javier Vegas [ctb],
Pierre-Antoine Bretonniere [ctb],
Roberto Serrano [ctb]

Maintainer Nicolau Manubens <nicolau.manubens@bsc.es>

R topics documented:

`CDORemapper` *CDO Remap Data Transformation for 'startR'*

Description

This is a transform function that uses CDO software to remap longitude-latitude data subsets onto a specified target grid, intended for use as parameter `transform` in a call to the function `Start()` in the package 'startR'. This function complies with the input/output interface required by `Start()` defined in the documentation for the parameter `transform` of that function.

This function uses the function `CDORemap()` in the package 's2dverification' to perform the interpolations and hence requires having CDO installed in the machine.

Usage

```
CDORemapper(data_array, variables, file_selectors, ...)
```

Arguments

`data_array`	Input data array to be transformed. See details in the documentation of the parameter `transform` of the function `Start()`.
`variables`	Auxiliary variables required for the transformation, automatically provided by `Start()`. See details in the documentation of the parameter `transform` of the function `Start()`.
`file_selectors`	Information on the path of the file the input data array comes from. See details in the documentation of the parameter `transform` of the function `Start()`.
`...`	Additional parameters to adjust the transform process, as provided in the parameter `transform_params` in a call to the function `Start()`. See details in the documentation of the parameter `transform` of the function `Start()`.

Value

An array with the same amount of dimensions as the input data array, potentially with different sizes, and potentially with the attribute 'variables' with additional auxiliary data. See details in the documentation of the parameter `transform` of the function `Start()`.

Author(s)

History:
0.0 - 2017-04 (N. Manubens, <nicolau.manubens at bsc.es>) - Original code

See Also

`CDORemap`

Examples

```
## Check https://earth.bsc.es/gitlab/es/startR for step-by-step examples
## of Start() that use CDORemapper().
```

`CircularSort` *Circular Sort Dimension Reorder for 'startR'*

Description

This is a function that generates a reorder function intended for use as parameter `<dim_name>_reorder` in a call to the function `Start()` in the package 'startR'. This function complies with the input/output interface required by `Start()` defined in the documentation for the parameter ... of that function.

The applied reordering consists of a circular sort of the coordinate variable values, where any values beyond the limits specified in the parameters `start` and `end` is applied a modulus to fall in the specified range. This is useful for circular dimensions such as the Earth longitudes.

Usage

```
CircularSort(start, end, ...)
```

Arguments

`start`	Numeric lower bound of the circular range.
`end`	Numeric upper bound of the circular range.
`...`	Additional parameters to adjust the reorderig (sent internally to the function `sort()`).

Value

List with the reordered values in the component `$x` and the permutation indices in the component `$ix`. See details in the documentation of the parameter ... of the function `Start()`.

Author(s)

History:
0.0 - 2017-04 (N. Manubens, `<nicolau.manubens at bsc.es>`) - Original code

See Also

`Sort`

Examples

```
## Check https://earth.bsc.es/gitlab/es/startR for step-by-step examples
## of Start() that use CircularSort().
```

indices *Mark Dimension Selectors as Indices*

Description

Helper function intended for use in a call to the function `Start` in the package 'startR', to explicitly mark that a set of provided indices to subset one of the requested dimensions are actually indices and not values to be matched against a coordinate variable. See details in the documentation of the parameter ... of the function `Start()`.

Usage

```
indices(x)
```

Arguments

`x`	Numeric vector or list with two numeric elements.

Value

The same as the input, but with an additional attribute 'indices' with the value TRUE, marking the indices as numeric indices.

Author(s)

See details in the documentation of the parameter `transform` of the function `Start()`.

Examples

```
## Check https://earth.bsc.es/gitlab/es/startR for step-by-step examples
## of Start() that use indices().
```

NcCloser *NetCDF File Closer for 'startR'*

Description

This is a file closer function for NetCDF files, intended for use as parameter `file_closer` in a call to the function `Start()` in the package 'startR'. This function complies with the input/output interface required by `Start()` defined in the documentation for the parameter `file_closer` of that function.

This function uses the function `NcClose()` in the package 'easyNCDF', which in turn uses `nc_close()` in the package 'ncdf4'.

Usage

```
NcCloser(file_object)
```

Arguments

file_object	Open connection to a NetCDF file, optionally with additional header information. See details in the documentation of the parameter file_closer of the function Start().

Value

This function returns NULL.

Author(s)

History:
0.0 - 2017-04 (N. Manubens, <nicolau.manubens at bsc.es>) - Original code

See Also

NcOpener, NcDimReader, NcDataReader, NcVarReader

Examples

```
## Check https://earth.bsc.es/gitlab/es/startR for step-by-step examples
## of Start() that use NcCloser().
```

NcDataReader *NetCDF File Data Reader for 'startR'*

Description

This is a data reader function for NetCDF files, intended for use as parameter file_data_reader in a call to the function Start() in the package 'startR'. This function complies with the input/output interface required by Start() defined in the documentation for the parameter file_data_reader of that function.

This function uses the function NcToArray() in the package 'easyNCDF', which in turn uses nc_var_get() in the package 'ncdf4'.

Usage

```
NcDataReader(file_path = NULL, file_object = NULL, file_selectors = NULL,
             inner_indices = NULL, synonims)
```

Arguments

file_path	Character string with the path to the data file to read. See details in the documentation of the parameter file_data_reader of the function Start().
file_object	Open connection to a NetCDF file, optionally with additional header information. See details in the documentation of the parameter file_data_reader of the function Start().
file_selectors	Information on the path of the file to read data from. See details in the documentation of the parameter file_data_reader of the function Start().

inner_indices
: Named list with the numeric indices to take from each of the inner dimensions in the requested file. See details in the documentation of the parameter file_data_reader of the function Start().

synonims
: Named list with synonims for the dimension names to look for in the requested file, exactly as provided in the parameter synonims in a call to the function Start(). See details in the documentation of the parameter file_data_reader of the function Start().

Value

A multidimensional data array with the named dimensions and indices requested in inner_indices, potentially with the attribute 'variables' with additional auxiliary data. See details in the documentation of the parameter file_data_reader of the function Start().

Author(s)

History:
0.0 - 2017-04 (N. Manubens, <nicolau.manubens at bsc.es>) - Original code

See Also

NcOpener, NcCloser, NcDimReader, NcVarReader

Examples

```
## Check https://earth.bsc.es/gitlab/es/startR for step-by-step examples
## of Start() that use NcDataReader().
```

NcDimReader *NetCDF Dimension Reader for 'startR'*

Description

This is a dimension reader function for NetCDF files, intended for use as parameter file_dim_reader in a call to the function Start() in the package 'startR'. This function complies with the input/output interface required by Start() defined in the documentation for the parameter file_dim_reader of that function.

This function uses the function NcReadDims() in the package 'easyNCDF'.

Usage

```
NcDimReader(file_path = NULL, file_object = NULL, file_selectors = NULL,
            inner_indices = NULL, synonims)
```

Arguments

file_path	Character string with the path to the data file to read the dimensions for. See details in the documentation of the parameter file_dim_reader of the function Start().
file_object	Open connection to a NetCDF file, optionally with additional header information. See details in the documentation of the parameter file_dim_reader of the function Start().
file_selectors	Information on the path of the file to read data from. See details in the documentation of the parameter file_dim_reader of the function Start().
inner_indices	Named list with the numeric indices to take from each of the inner dimensions in the requested file. Used only in advanced configurations. See details in the documentation of the parameter file_dim_reader of the function Start().
synonims	Named list with synonims for the dimension names to look for in the requested file, exactly as provided in the parameter synonims in a call to the function Start(). See details in the documentation of the parameter file_dim_reader of the function Start().

Value

Named numeric vector with the names and sizes of the dimensions of the requested file.

Author(s)

History:
0.0 - 2017-04 (N. Manubens, <nicolau.manubens at bsc.es>) - Original code

See Also

NcOpener, NcCloser, NcDataReader, NcVarReader

Examples

```
## Check https://earth.bsc.es/gitlab/es/startR for step-by-step examples
## of Start() that use NcDimReader().
```

NcOpener *NetCDF File Opener for 'startR'*

Description

This is a file closer function for NetCDF files, intended for use as parameter file_opener in a call to the function Start() in the package 'startR'. This function complies with the input/output interface required by Start() defined in the documentation for the parameter file_opener of that function.

This function uses the function NcOpen() in the package 'easyNCDF', which in turn uses nc_open() in the package 'ncdf4'.

Usage

```
NcOpener(file_path)
```

Arguments

file_path	Character string with the path to the data file to read. See details in the documentation of the parameter file_opener of the function Start().

Value

An open connection to a NetCDF file, with additional header information, as returned by nc_open in the package 'ncdf4'. See details in the documentation of the parameter file_opener of the function Start().

Author(s)

History:
0.0 - 2017-04 (N. Manubens, <nicolau.manubens at bsc.es>) - Original code

See Also

NcCloser, NcDataReader, NcDimReader, NcVarReader

Examples

```
## Check https://earth.bsc.es/gitlab/es/startR for step-by-step examples
## of Start() that use NcOpener().
```

NcVarReader *NetCDF Variable Reader for 'startR'*

Description

This is an auxiliary variable reader function for NetCDF files, intended for use as parameter file_var_reader in a call to the function Start() in the package 'startR'. This function complies with the input/output interface required by Start() defined in the documentation for the parameter file_var_reader of that function.

This function uses the function NcDataReader() in the package 'startR', which in turn uses NcToArray() in the package 'easyNCDF', which in turn uses nc_var_get() in the package 'ncdf4'.

Usage

```
NcVarReader(file_path = NULL, file_object = NULL, file_selectors = NULL,
            var_name = NULL, synonims)
```

Arguments

file_path	Character string with the path to the data file to read the variable from. See details in the documentation of the parameter file_var_reader of the function Start().
file_object	Open connection to a NetCDF file, optionally with additional header information. See details in the documentation of the parameter file_var_reader of the function Start().
file_selectors	Information on the path of the file to read data from. See details in the documentation of the parameter file_var_reader of the function Start().
var_name	Character string with the name of the variable to be read.
synonims	Named list with synonims for the variable names to look for in the requested file, exactly as provided in the parameter synonims in a call to the function Start(). See details in the documentation of the parameter file_var_reader of the function Start().

Value

A multidimensional data array with the named dimensions, potentially with the attribute 'variables' with additional auxiliary data. See details in the documentation of the parameter file_var_reader of the function Start().

Author(s)

History:
0.0 - 2017-04 (N. Manubens, <nicolau.manubens at bsc.es>) - Original code

See Also

NcOpener, NcCloser, NcDataReader, NcDimReader

Examples

```
## Check https://earth.bsc.es/gitlab/es/startR for step-by-step examples
## of Start() that use NcVarReader().
```

SelectorChecker *Default Selector Checker for 'startR'*

Description

This is a selector checker function intended for use as parameter selector_checker in a call to the function Start() in the package 'startR'. This function complies with the input/output interface required by Start() defined in the documentation for the parameter selector_checker of that function.

Usage

```
SelectorChecker(selectors, var = NULL, return_indices = TRUE, tolerance = NULL)
```

Arguments

selectors	Numeric indices or variable values to be retrieved for a dimension, automatically provided by Start(). See details in the documentation of the parameters selector_checker and ... of the function Start(). The indices or values can be provided in the form of a vector or in the form of a list with two elements.
var	Vector of values of a coordinate variable where to search matches with the provided indices or values in the parameter selectors, automatically provided by Start(). See details in the documentation of the parameters selector_checker and ... of the function Start(). The parameter var is optional. When not specified, SelectorChecker simply returns the input indices.
return_indices	Boolean flag, automatically configured by Start(), telling whether to return numeric indices or coordinate variable values after doing the matching.
tolerance	Numeric value with a tolerance value to be used in the matching of the selectors and var. See documentation on <dim_name>_tolerance in ..., in the documentation of the function Start().

Value

A vector of either the indices of the matching values (if return_indices = TRUE) or the matching values themselves (if return_indices = FALSE).

Author(s)

History:
0.0 - 2017-04 (N. Manubens, <nicolau.manubens at bsc.es>) - Original code

Examples

```
## Check https://earth.bsc.es/gitlab/es/startR for step-by-step examples
## of Start() that use SelectorChecker().
```

Sort *Sort Dimension Reorder for 'startR'*

Description

This is a reorder function intended for use as parameter <dim_name>_reorder in a call to the function Start() in the package 'startR'. This function complies with the input/output interface required by Start() defined in the documentation for the parameter ... of that function.

The applied reordering consists of an increasing sort of the coordinate variable values.

Usage

```
Sort(...)
```

Arguments

...	Additional parameters to adjust the reorderig (sent internally to the function sort()).

Value

List with the reordered values in the component `$x` and the permutation indices in the component `$ix`. See details in the documentation of the parameter ... of the function `Start()`.

Author(s)

History:
0.0 - 2017-04 (N. Manubens, <nicolau.manubens at bsc.es>) - Original code

See Also

`CircularSort`

Examples

```
## Check https://earth.bsc.es/gitlab/es/startR for step-by-step examples
## of Start() that use Sort().
```

`Start` *Declare, Discover, Subset and Retrieve Multidimensional Distributed Data Sets*

Description

See the `startR` documentation and tutorial for a step-by-step explanation on how to use `Start()`.

Nowadays in the era of Big Data, large multidimensional data sets from diverse sources need to be combined and processed. Analysis of Big Data in any field is often highly complex and time-consuming. Taking subsets of these datasets (Divide) and processing them efficiently (and Conquer) becomes an indispensable practice. This technique is also known as Domain Decomposition, Map Reduce or, more commonly, 'chunking'.

`startR` (Subset, TrAnsform, ReTrieve, arrange and process large multidimensional data sets in R) is an R project started at BSC with the aim to develop a tool that allows the user to automatically process large multidimensional distributed data sets. It is an open source project that is open to external collaboration and funding, and will continuously evolve to support as many data set formats as possible while maximizing its efficiency.

`startR` provides a framework under which a data set (collection of one or multiple data files, potentially distributed over various remote servers) are perceived as if they all were part of a single large multidimensional array. Once such multidimensional array is declared, any user-defined function can be applied to the data in a `apply`-like fashion, where `startR` transparently implements the Map Reduce paradigm. The steps to follow in order to process a collection of Big Data sets are as follows:

- Declaring the data set, i.e. declaring the distribution of the data files involved, the dimensions and shape of the multidimensional array, and the boundaries of the tatget data. Numeric indices or coordinate values can be used when fixing the boundaries. Once a data set is declared,

a list of involved files, dimension lengths, memory size and other metadata is made available. Optionally, the data set can be retrieved and loaded onto the current R session if it is small enough. This step can be performed with the `Start()` function.

- Declaring the workflow of operations to perform on the involved data set(s). This step can be performed with the `Step()` and `AddStep()` functions.
- Defining the computation settings. The mandatory settings include a) how many subsets to divide the data sets into and along which dimensions; b) which platform to perform the workflow of operations on (local machine, remote machine, remote HPC?), how to communicate with it (unidirectional or bidirectional connection? shared or separate file systems?), which queuing system it uses (slurm, PBS, LSF, none?); and c) how many parallel jobs and execution threads per job to use when running the calculations. This step can be performed when building up the call to the `Compute()` function.
- Running the computation. startR transparently implements the Map Reduce paradigm, according to the settings in the previous steps. The progress can optionally be monitored with the EC-Flow workflow management tool. When the computation ends, a report of performance timings is displayed. This step can be triggered with the `Compute()` function.

`startR` is not bound to a specific file format. Interface functions to custom file formats can be provided for `Start()` to read them. As of April 2017 `startR` includes interface functions to the following file formats:

- NetCDF

Usage

```
Start(...,
      return_vars = NULL,
      synonims = NULL,
      file_opener = NcOpener,
      file_var_reader = NcVarReader,
      file_dim_reader = NcDimReader,
      file_data_reader = NcDataReader,
      file_closer = NcCloser,
      transform = NULL,
      transform_params = NULL,
      transform_vars = NULL,
      transform_extra_cells = 0,
      apply_indices_after_transform = FALSE,
      pattern_dims = NULL,
      metadata_dims = NULL,
      selector_checker = SelectorChecker,
      merge_across_dims = FALSE,
      split_multiselected_dims = FALSE,
      path_glob_permissive = FALSE,
      retrieve = FALSE,
      num_procs = 1,
      silent = FALSE,
      debug = FALSE)
```

Arguments

. . .

When willing to retrieve data from one or a collection of data sets, the involved data can be perceived as belonging to a large multi-dimensional array. For instance, let us consider an example case. We want to retrieve data from a source, which contains data for the number of monthly sales of various items, and also for their retail price each month. The data on source is stored as follows:

```
#  /data/
#    |-> sales/
#    |    |-> electronics
#    |    |    |-> item_a.data
#    |    |    |-> item_b.data
#    |    |    |-> item_c.data
#    |    |-> clothing
#    |         |-> item_d.data
#    |         |-> idem_e.data
#    |         |-> idem_f.data
#    |-> prices/
#         |-> electronics
#         |    |-> item_a.data
#         |    |-> item_b.data
#         |    |-> item_c.data
#         |-> clothing
#              |-> item_d.data
#              |-> item_e.data
#              |-> item_f.data
```

Each item file contains data, stored in whichever format, for the sales or prices over a time period, e.g. for the past 24 months, registered at 100 different stores over the world. Whichever the format it is stored in, each file can be perceived as a container of a data array of 2 dimensions, time and store. Let us assume the '.data' format allows to keep a name for each of these dimensions, and the actual names are 'time' and 'store'.

The different item files for sales or prices can be perceived as belonging to an 'item' dimension of length 3, and the two groups of three items to a 'section' dimension of length 2, and the two groups of two sections (one with the sales and the other with the prices) can be perceived as belonging also to another dimension 'variable' of length 2. Even the source can be perceived as belonging to a dimension 'source' of length 1.

All in all, in this example, the whole data could be perceived as belonging to a multidimensional 'large array' of dimensions

```
#  source variable  section      item    store    month
#       1        2        2         3      100       24
```

The dimensions of this 'large array' can be classified in two types. The ones that group actual files (the file dimensions) and the ones that group data values inside the files (the inner dimensions). In the example, the file dimensions are 'source', 'variable', 'section' and 'item', whereas the inner dimensions are

'store' and 'month'.

Having the dimensions of our target sources in mind, the parameter ... expects to receive information on:

- The names of the expected dimensions of the 'large dataset' we want to retrieve data from
- The indices to take from each dimension (and other constraints)
- How to reorder the dimension if needed
- The location and organization of the files of the data sets

For each dimension, the 3 first information items can be specified with a set of parameters to be provided through .... For a given dimension 'dimname', six parameters can be specified:

```
# dimname = <indices_to_take>,  # 'all' / 'first' / 'last' /
#                               # indices(c(1, 10, 20)) /
#                               # indices(c(1:20)) /
#                               # indices(list(1, 20)) /
#                               # c(1, 10, 20) / c(1:20) /
#                               # list(1, 20)
# dimname_var = <name_of_associated_coordinate_variable>,
# dimname_tolerance = <tolerance_value>,
# dimname_reorder = <reorder_function>,
# dimname_depends = <name_of_another_dimension>,
# dimname_across = <name_of_another_dimension>
```

The **indices to take** can be specified in three possible formats (see code comments above for examples). The first format consists in using character tags, such as 'all' (take all the indices available for that dimension), 'first' (take only the first) and 'last' (only the last). The second format consists in using numeric indices, which have to be wrapped in a call to the `indices()` helper function. For the second format, either a vector of numeric indices can be provided, or a list with two numeric indices can be provided to take all the indices in the range between the two specified indices (both extremes inclusive). The third format consists in providing a vector character strings (for file dimensions) or of values of whichever type (for inner dimensions). For the file dimensions, the provided character strings in the third format will be used as components to build up the final path to the files (read further). For inner dimensions, the provided values in the third format will be compared to the values of an associated coordinate variable (must be specified in `dimname_reorder`, read further), and the indices of the closest values will be retrieved. When using the third format, a list with two values can also be provided to take all the indices of the values within the specified range.

The **name of the associated coordinate variable** must be a character string with the name of an associated coordinate variable to be found in the data files (in all* of them). For this to work, a `file_var_reader` function must be specified when calling `Start()` (see parameter 'file_var_reader'). The coordinate variable must also be requested in the parameter `return_vars` (see its section for details). This feature only works for inner dimensions.

The **tolerance value** is useful when indices for an inner dimension are specified in the third format (values of whichever type). In that case, the indices of

the closest values in the coordinate variable are seeked. However the closest value might be too distant and we would want to consider no real match exists for such provided value. This is possible via the tolerance. which allows to specify a threshold beyond which not to seek for matching values and mark that index as missing value.

The **reorder_function** is useful when indices for an inner dimension are specified in the third fromat, and the retrieved indices need to be reordered in function of their provided associated variable values. A function can be provided, which receives as input a vector of values, and returns as outputs a list with the components `x` with the reordered values, and `ix` with the permutation indices. Two reordering functions are included in `startR`, the `Sort()` and the `CircularSort()`.

The **name of another dimension** to be specified in `dimname_depends`, only available for file dimensions, must be a character string with the name of another requested **file dimension** in ..., and will make `Start()` aware that the path components of a file dimension can vary in function of the path component of another file dimension. For instance, in the example above, specifying `item_depends = 'section'` will make `Start()` aware that the item names vary in function of the section, i.e. section 'electronics' has items 'a', 'b' and 'c' but section 'clothing' has items 'd', 'e', 'f'. Otherwise `Start()` would expect to find the same item names in all the sections.

The **name of another dimension** to be specified in `dimname_across`, only available for inner dimensions, must be a character string with the name of another requested **inner dimension** in ..., and will make `Start()` aware that an inner dimension extends along multiple files. For instance, let us imagine that in the example above, the records for each item are so large that it becomes necessary to split them in multiple files each one containing the registers for a different period of time, e.g. in 10 files with 100 months each ('item_a_period1.data', 'item_a_period2.data', and so on). In that case, the data can be perceived as having an extra file dimension, the 'period' dimension. The inner dimension 'month' would extend across multiple files, and providing the parameter `month = indices(1, 300)` would make `Start()` crash because it would perceive we have made a request out of bounds (each file contains 100 'month' indices, but we requested 1 to 300). This can be solved by specifying the parameter `month_across = period` (along with the full specification of the dimension 'period').

Defining the path pattern

As mentioned above, the parameter ... also expects to receive information with the location of the data files. In order to do this, a special dimension must be defined. In that special dimension, in place of specifying indices to take, a path pattern must be provided. The path pattern is a character string that encodes the way the files are organized in their source. It must be a path to one of the data set files in an accessible local or remote file system, or a URL to one of the files provided by a local or remote server. The regions of this path that vary across files (along the file dimensions) must be replaced by wildcards. The wildcards must match any of the defined file dimensions in the call to `Start()` and must be delimited with heading and trailing '$'. Shell globbing expressions can be used in the path pattern. See the next code snippet for an example of a path

pattern.

All in all, the call to `Start()` to load the entire data set in the example of store item sales, would look as follows:

```
# data <- Start(source = paste0('/data/$variable$/',
#                               '$section$/$item$.data'),
#               variable = 'all',
#               section = 'all',
#               item = 'all',
#               item_depends = 'section',
#               store = 'all',
#               month = 'all')
```

Note that in this example it would still be pending to properly define the parameters `file_opener`, `file_closer`, `file_dim_reader`, `file_var_reader` and `file_data_reader` for the '.data' file format (see the corresponding sections).

The call to `Start()` will return a multidimensional R array with the following dimensions:

```
#  source variable  section      item    store    month
#       1        2        2         3      100       24
```

The dimension specifications in the ... do not have to follow any particular order. The returned array will have the dimensions in the same order as they have been specified in the call. For example, the following call:

```
# data <- Start(source = paste0('/data/$variable$/',
#                               '$section$/$item$.data'),
#               month = 'all',
#               store = 'all',
#               item = 'all',
#               item_depends = 'section',
#               section = 'all',
#               variable = 'all')
```

would return an array with the following dimensions:

```
#  source    month    store      item  section variable
#       1       24      100         3        2        2
```

Next, a more advanced example to retrieve data for only the sales records, for the first section ('electronics'), for the 1st and 3rd items and for the stores located in Barcelona (assuming the files contain the variable 'store_location' with the name of the city each of the 100 stores are located at):

```
# data <- Start(source = paste0('/data/$variable$/',
#                               '$section$/$item$.data'),
#               variable = 'sales',
```

```
#                section = 'first',
#                item = indices(c(1, 3)),
#                item_depends = 'section',
#                store = 'Barcelona',
#                store_var = 'store_location',
#                month = 'all',
#                return_vars = list(store_location = NULL))
```

The defined names for the dimensions do not necessarily have to match the names of the dimensions inside the file. Lists of alternative names to be seeked can be defined in the parameter `synonims`.

If data from multiple sources (not necessarily following the same structure) has to be retrieved, it can be done by providing a vector of character strings with path pattern specifications, or, in the extended form, by providing a list of lists with the components 'name' and 'path', and the name of the dataset and path pattern as values, respectively. For example:

```
# data <- Start(source = list(
#                   list(name = 'sourceA',
#                        path = paste0('/sourceA/$variable$/',
#                                      '$section$/$item$.data')),
#                   list(name = 'sourceB',
#                        path = paste0('/sourceB/$section$/',
#                                      '$variable$/$item$.data'))
#                ),
#                variable = 'sales',
#                section = 'first',
#                item = indices(c(1, 3)),
#                item_depends = 'section',
#                store = 'Barcelona',
#                store_var = 'store_location',
#                month = 'all',
#                return_vars = list(store_location = NULL))
```

`return_vars` Apart from retrieving a multidimensional data array, retrieving auxiliary variables inside the files can also be needed. The parameter `return_vars` allows for requesting such variables, as long as a `file_var_reader` function is also specified in the call to `Start()` (see documentation on the corresponding parameter).

This parameter expects to receive a named list where the names are the names of the variables to be fetched in the files, and the values are vectors of character strings with the names of the file dimension which to retrieve each variable for, or `NULL` if the variable has to be retrieved only once from any (the first) of the involved files. In the case of the the item sales example (see documentation on parameter ...), the store location variable is requested with the parameter `return_vars = list(store_location = NULL)`. This will cause `Start()` to fetch once the variable 'store_location' and return it in the component `$Variables$common$store_location`, and will be an array of charac-

ter strings with the location names, with the dimensions `c('store' = 100)`. Although useless in this example, we could ask `Start()` to fetch and return such variable for each file along the items dimension as follows: `return_vars = list(store_location = c('item'))`. In that case, the variable will be fetched once from a file of each of the items, and will be returned as an array with the dimensions `c('item' = 3, 'store' = 100)`.

If a variable is requested along a file dimension that contains path pattern specifications ('source' in the example), the fetched variable values will be returned in the component `$Variables$<dataset_name>$<variable_name>`. For example:

```
# data <- Start(source = list(
#                 list(name = 'sourceA',
#                      path = paste0('/sourceA/$variable$/',
#                                    '$section$/$item$.data')),
#                 list(name = 'sourceB',
#                      path = paste0('/sourceB/$section$/',
#                                    '$variable$/$item$.data'))
#               ),
#               variable = 'sales',
#               section = 'first',
#               item = indices(c(1, 3)),
#               item_depends = 'section',
#               store = 'Barcelona',
#               store_var = 'store_location',
#               month = 'all',
#               return_vars = list(store_location = c('source',
#                                                     'item')))
# # Checking the structure of the returned variables
# str(found_data$Variables)
# Named list
# ..$common: NULL
# ..$sourceA: Named list
# .. ..$store_location: char[1:18(3d)] 'Barcelona' 'Barcelona' ...
# ..$sourceB: Named list
# .. ..$store_location: char[1:18(3d)] 'Barcelona' 'Barcelona' ...
# # Checking the dimensions of the returned variable
# # for the source A
# dim(found_data$Variables$sourceA)
#     item   store
#        3       3
```

The names of the requested variables do not necessarily have to match the actual variable names inside the files. A list of alternative names to be seeked can be specified via the parameter `synonims`.

`synonims` In some requests, data from different sources may follow different naming conventions for the dimensions or variables, or even files in the same source could have varying names. In order for `Start()` to properly identify the dimensions or variables with different names, the parameter `synonims` can be specified as a named list where the names are requested variable or dimension names, and the values are vectors of character strings with alternative names to seek for such

dimension or variable.

In the example used in parameter `return_vars`, it may be the case that the two involved data sources follow slightly different naming conventions. For example, source A uses 'sect' as name for the sections dimension, whereas source B uses 'section'; source A uses 'store_loc' as variable name for the store locations, whereas source B uses 'store_location'. This can be taken into account as follows:

```
# data <- Start(source = list(
#                 list(name = 'sourceA',
#                      path = paste0('/sourceA/$variable$/',
#                                    '$section$/$item$.data')),
#                 list(name = 'sourceB',
#                      path = paste0('/sourceB/$section$/',
#                                    '$variable$/$item$.data'))
#               ),
#               variable = 'sales',
#               section = 'first',
#               item = indices(c(1, 3)),
#               item_depends = 'section',
#               store = 'Barcelona',
#               store_var = 'store_location',
#               month = 'all',
#               return_vars = list(store_location = c('source',
#                                                     'item')),
#               synonims = list(
#                 section = c('sec', 'section'),
#                 store_location = c('store_loc',
#                                    'store_location')
#               ))
```

`file_opener` A function that receives as a single parameter (`file_path`) a character string with the path to a file to be opened, and returns an object with an open connection to the file (optionally with header information) on success, or returns `NULL` on failure.

This parameter takes by default `NcOpener` (an opener function for NetCDF files).

See `NcOpener` for a template to build a file opener for your own file format.

`file_var_reader`
A function with the header `file_path = NULL`, `file_object = NULL`, `file_selectors = NULL`, `var_name`, `synonims` that returns an array with auxiliary data (i.e. data from a variable) inside a file. `Start()` will provide automatically either a `file_path` or a `file_object` to the `file_var_reader` function (the function has to be ready to work whichever of these two is provided). The parameter `file_selectors` will also be provided automatically to the variable reader, containing a named list where the names are the names of the file dimensions of the queried data set (see documentation on ...) and the values are single character strings with the components used to build the path to the file being read (the one provided in `file_path` or `file_object`). The

parameter var_name will be filled in automatically by Start() also, with the name of one of the variales to be read. The parameter synonims will be filled in with exactly the same value as provided in the parameter synonims in the call to Start(), and has to be used in the code of the variable reader to check for alternative variable names inside the target file. The file_var_reader must return a (multi)dimensional array with named dimensions, and optionally with the attribute 'variales' with other additional metadata on the retrieved variable.

Usually, the file_var_reader should be a degenerate case of the file_data_reader (see documentation on the corresponding parameter), so it is recommended to code the file_data_reder in first place.

This parameter takes by default NcVarReader (a variable reader function for NetCDF files).

See NcVarReader for a template to build a variale reader for your own file format.

file_dim_reader

A function with the header file_path = NULL, file_object = NULL, file_selectors = NULL, synonims that returns a named numeric vector where the names are the names of the dimensions of the multidimensional data array in the file and the values are the sizes of such dimensions. Start() will provide automatically either a file_path or a file_object to the file_dim_reader function (the function has to be ready to work whichever of these two is provided). The parameter file_selectors will also be provided automatically to the dimension reader, containing a named list where the names are the names of the file dimensions of the queried data set (see documentation on ...) and the values are single character strings with the components used to build the path to the file being read (the one provided in file_path or file_object). The parameter synonims will be filled in with exactly the same value as provided in the parameter synonims in the call to Start(), and can optionally be used in advanced configurations.

This parameter takes by default NcDimReader (a dimension reader function for NetCDF files).

See NcDimReader for a(n advanced) template to build a dimension reader for your own file format.

file_data_reader

A function with the header file_path = NULL, file_object = NULL, file_selectors = NULL, inner_indices = NULL, synonims that returns a subset of the multidimensional data array inside a file (even if internally it is not an array). Start() will provide automatically either a file_path or a file_object to the file_data_reader function (the function has to be ready to work whichever of these two is provided). The parameter file_selectors will also be provided automatically to the data reader, containing a named list where the names are the names of the file dimensions of the queried data set (see documentation on ...) and the values are single character strings with the components used to build the path to the file being read (the one provided in file_path or file_object). The parameter inner_indices will be filled in automatically by Start() also, with a named list of numeric vectors,

where the names are the names of all the expected inner dimensions in a file to be read, and the numeric vectors are the indices to be taken from the corresponding dimension (the indices may not be consecutive nor in order). The parameter synonims will be filled in with exactly the same value as provided in the parameter synonims in the call to Start(), and has to be used in the code of the data reader to check for alternative dimension names inside the target file. The file_data_reader must return a (multi)dimensional array with named dimensions, and optionally with the attribute 'variales' with other additional metadata on the retrieved data.

Usually, the file_data_reader should use the file_dim_reader (see documentation on the corresponding parameter), so it is recommended to code the file_dim_reder in first place.

This parameter takes by default NcDataReader (a data reader function for NetCDF files).

See NcDataReader for a template to build a data reader for your own file format.

file_closer A function that receives as a single parameter (file_object) an open connection (as returned by file_opener) to one of the files to be read, optionally with header information, and closes the open connection. Always returns NULL.

This parameter takes by default NcCloser (a closer function for NetCDF files).

See NcCloser for a template to build a file closer for your own file format.

transform A function with the header dara_array, variables, file_selectors = NULL, ... It receives as input, through the parameter data_array, a subset of a multidimensional array (as returned by file_data_reader), applies a transformation to it and returns it, preserving the amount of dimensions but potentially modifying their size. This transformation may require data from other auxiliary variables, automatically provided to transform through the parameter variables, in the form of a named list where the names are the variable names and the values are (multi)dimensional arrays. Which variables need to be sent to transform can be specified with the parameter transform_vars in Start(). The parameter file_selectors will also be provided automatically to transform, containing a named list where the names are the names of the file dimensions of the queried data set (see documentation on ...) and the values are single character strings with the components used to build the path to the file the subset being processed belongs to. The parameter ... will be filled in with other additional parameters to adjust the transformation, exactly as provided in the call to Start() via the parameter transform_params.

transform_params
Named list with additional parameters to be sent to the transform function (if specified). See documentation on transform for details.

transform_vars
Vector of character strings with the names of auxiliary variables to be sent to the transform function (if specified). All the variables to be sent to transform must also have been requested as return variables in the parameter return_vars of Start().

transform_extra_cells
Number of extra indices to retrieve from the data set, beyond the requested in-

dices in ..., in order for `transform` to dispose of additional information to properly apply whichever transformation (if needed). As many as `transform_extra_cells` will be retrieved beyond each of the limits for each of those inner dimensions associated to a coordinate variable and sent to `transform` (i.e. present in `transform_vars`). After `transform` has finished, `Start()` will take again and return a subset of the result, for the returned data to fall within the specified bounds in ....

`apply_indices_after_transform`
: When a `transform` is specified in `Start()` and numeric indices are provided for any of the inner dimensions that depend on coordinate variables, these numeric indices can be made effective (retrieved) before applying the transformation or after. The boolean flab `apply_indices_after_transform` allows to adjust this behaviour. It takes `FALSE` by default (numeric indices are applied before sending data to `transform`).

`pattern_dims` Name of the dimension with path pattern specifications (see ... for details). If not specified, `Start()` assumes the first provided dimension is the pattern dimension, with a warning.

`metadata_dims`
: It expects to receive a vector of character strings with the names of the file dimensions which to return metadata for. As noted in `file_data_reader`, the data reader can optionally return auxiliary data via the attribute 'variables' of the returned array. `Start()` by default returns the auxiliary data read for only the first file of each source (or data set) in the pattern dimension (see ... for info on what the pattern dimension is). However it can be configured to return the metadata for all the files along any set of file dimensions. The parameter `metadata_dims` allows to configure this level of granularity of the returned metadata.

`selector_checker`
: Function used internaly by `Start()` to translate a set of selectors (values for a dimension associated to a coordinate variable) into a set of numeric indices. It takes by default `SelectorChecker` and, in principle, it should not be required to change it for customized file formats. The option to replace it is left open for more versatility. See the code of `SelectorChecker` for details on the inputs, functioning and outputs of a selector checker.

`merge_across_dims`
: Whether to merge dimensions across which another dimension extends (according to the `*_across` parameters). Takes the value `FALSE` by default. For example, if the dimension 'time' extends across the dimension 'chunk' and `merge_across_dims = TRUE`, the resulting data array will only contain only the dimension 'time' as long as all the chunks together.

`split_multiselected_dims`
: Whether to split a dimension that has been selected with a multidimensional array of selectors into as many dimensions as present in the selector array. Takes the value `FALSE` by default.

`path_glob_permissive`
: When specifying a path pattern for a dataset, it might contain shell glob experissions. For each dataset, the first file matching the path pattern is found, and the found file is used to work out fixed values for the glob expressions that will be used for all the files of the dataset. However in some cases the values of the shell glob expressions may not be constant for all files in a dataset, and they need to be worked out for each file involved. In this situation, the

path_glob_permissive can be set to an integer value specifying for how many folder levels in the path pattern, beginning from the end, the shell glob expressions mut be preserved and worked out for each file.
The default value is FALSE, which is equivalent to 0. Setting TRUE is equivalent to 1.
For example, a path pattern could be as follows: '/path/to/dataset/var_*/$date$_*_
Leaving path_glob_permissive = FALSE will trigger automatic seek of the contents to replace the asterisks (e.g. the first asterisk matches with 'bar' and the second with 'baz'. The found contents will be used for all files in the dataset (in the example, the path pattern will be fixed to '/path/to/dataset/var_bar
However, if any of the files in the dataset have other contents in the position of the asterisks, Start() will not find them (in the example, a file like '/path/to/dataset/prec would not be found). Setting path_glob_permissive = 1 would preserve global expressions in the latest level (in the example, the fixed path pattern would be '/path/to/dataset/var_bar/$date$_*_foo.nc', and the problematic file mentioned before would be found), but of course this would slow down the Start() call if the dataset involves a large number of files. Setting path_glob_permissive = 2 would leave the original path pattern with the original glob expressions in the 1st and 2nd levels (in the example, both asterisks would be preserved, thus would allow Start() to recognize files such as '/path/to/dataset/precipitation_zzz/19901101_yyy_foo.nc').

retrieve — Logical value telling whether to retrieve the data defined in the Start call or to explore only its dimension lengths and names, and the values for the file and inner dimensions. Takes FALSE by default.

num_procs — Number of processes to be created for the parallel execution of the retrieval / transformation / arrangement of the multiple involved files in a call to Start(). If set to NULL, takes the number of available cores (as detected by detectCores() in the package 'future'). Takes 1 by default (no parallel execution).

silent — Boolean flag, whether to display progress messages (FALSE; default) or not (TRUE).

debug — Whether to return detailed messages on the progress and operations in a Start call (TRUE) or not (FALSE; default).

Details

Check the startR website for more information.

Value

If retrieve = TRUE the involved data is loaded into RAM memory and an object of the class 'startR_cube' with the following components is returned:

Data — Multidimensional data array with named dimensions, with the data values requested via ... and other parameters. This array can potentially contain metadata in the attribute 'variables'.

Variales — Named list of 1 + N components, containing lists of retrieved variables (as requested in return_vars) common to all the data sources (in the 1st component, $common), and for each of the N dara sources (named after the source name, as specified in ..., or, if not specified, $dat1, $dat2, ..., $datN). Each of the variables are contained in a multidimensional array with named dimensions, and potentially with the attribute 'variables' with additional auxiliary data.

`Files` Multidimensonal character string array with named dimensions. Its dimensions are the file dimensions (as requested in ...). Each cell in this array contains a path to a retrieved file, or `NULL` if the corresponding file was not found.

`NotFoundFiles`
Array with the same shape as `$Files` but with `NULL` in the positions for which the corresponding file was found, and a path to the expected file in the positions for which the corresponding file was not found.

`FileSelectors`
Multidimensional character string array with named dimensions, with the same shape as `$Files` and `$NotFoundFiles`, which contains the components used to build up the paths to each of the files in the data sources.

If `retrieve = FALSE` the involved data is not loaded into RAM memory and an object of the class 'startR_header' with the following components is returned:

`Dimensions` Named vector with the dimension lengths and names of the data involved in the `Start` call.

`Variales` Named list of 1 + N components, containing lists of retrieved variables (as requested in `return_vars`) common to all the data sources (in the 1st component, `$common`), and for each of the N dara sources (named after the source name, as specified in ..., or, if not specified, `$dat1`, `$dat2`, ..., `$datN`). Each of the variables are contained in a multidimensional array with named dimensions, and potentially with the attribute 'variables' with additional auxiliary data.

`Files` Multidimensonal character string array with named dimensions. Its dimensions are the file dimensions (as requested in ...). Each cell in this array contains a path to a file to be retrieved (which may exist or not).

`FileSelectors`
Multidimensional character string array with named dimensions, with the same shape as `$Files` and `$NotFoundFiles`, which contains the components used to build up the paths to each of the files in the data sources.

`StartRCall` List of parameters sent to the `Start` call, with the parameter `retrieve` set to `TRUE`. Intended for calling in order to retrieve the associated data a posteriori with a call to `do.call`.

Author(s)

History:
0.0 - 2017-04 (N. Manubens, <nicolau.manubens at bsc.es>) - Original code

Examples

```
## Check https://earth.bsc.es/gitlab/es/startR for step-by-step examples
## of Start().
```

`Subset` *Subset a Data Array*

Description

This function allows to subset (i.e. slice, take a chunk of) an array, in a similar way as done in the function `take()` in the package plyr. There are two main inprovements:

The input array can have dimension names, either in `names(dim(x))` or in the attribute 'dimensions', and the dimensions to subset along can be specified via the parameter `along` either with integer indices or either by their name.

There are additional ways to adjust which dimensions are dropped in the resulting array: either to drop all, to drop none, to drop only the ones that have been sliced or to drop only the ones that have not been sliced.

If an array is provided without dimension names, dimension names taken from the parameter `dim_names` will be added to the array.

Usage

```
Subset(x, along, indices, drop = FALSE)
```

Arguments

`x`	A multidimensional array to be sliced. It can have dimension names either in `names(dim(x))` or either in the attribute 'dimensions'.
`along`	Vector with references to the dimensions to take the subset from: either integers or dimension names.
`indices`	List of indices to take from each dimension specified in 'along'. If a single dimension is specified in 'along' the indices can be directly provided as a single integer or as a vector.
`drop`	Whether to drop all the dimensions of length 1 in the resulting array, none, only those that are specified in 'along', or only those that are not specified in 'along'. The possible values are, respectively: 'all' or TRUE, 'none' or FALSE, 'selected', and 'non-selected'.

Author(s)

History:
0.0 - 2016-06 (N. Manubens, <nicolau.manubens at bsc.es>) - Original code

Examples

```
sample_array <- array(1:24, dim = c(dataset = 1, sdate = 2, member = 3, ftime = 4))
subset <- Subset(sample_array, c('dataset', 'sdate', 'member'),
                 list(1, 1, 1), drop = 'selected')
```

Index