Package ‘RNeXML’

January 24, 2019

Type    Package
Title   Semantically Rich I/O for the 'NeXML' Format
Version 2.3.0
Description Provides access to phyloinformatic data in 'NeXML' format. The package should add new functionality to R such as the possibility to manipulate 'NeXML' objects in more various and refined way and compatibility with 'ape' objects.

URL    https://github.com/ropensci/RNeXML
BugReports https://github.com/ropensci/RNeXML/issues
License BSD_3_clause + file LICENSE
VignetteBuilder knitr

Additional_repositories http://packages.ropensci.org

Suggests spelling, rdflib, geiger (>= 2.0), phytools (>= 0.3.93),
        knitr (>= 1.5), rfigshare (>= 0.3.0), knitrations (>= 1.0.1),
        testthat (>= 0.10.0), rmarkdown (>= 0.3.3), rlang, xslt, covr,
        taxadb

Depends R (>= 3.0.0), ape (>= 3.1), methods (>= 3.0.0)
Imports XML (>= 3.95), plyr (>= 1.8), reshape2 (>= 1.2.2), httr (>= 0.3), uuid (>= 0.1-1), dplyr (>= 0.5.0), lazyeval (>= 0.1.0),
        tidyr (>= 0.3.1), stringr (>= 1.0), stringi, xml2

Collate 'S4-utils.R' 'classes.R' 'add_basic_meta.R' 'add_characters.R'
        'add_meta.R' 'add_namespaces.R' 'nexmlTree.R' 'add_trees.R'
        'character_classes.R' 'concatenate_nexml.R' 'constructors.R'
        'deprecated.R' 'get_basic_metadata.R' 'get_characters.R'
        'get_level.R' 'get_metadata.R' 'get_namespaces.R' 'get_rdf.R'
        'get_taxa.R' 'get_taxa_meta.R' 'get_trees.R'
        'internal_get_node_maps.R' 'internal_isEmpty.R'
        'internal_name_by_id.R' 'internal_nexml_id.R' 'meta.R'
        'nexml_add.R' 'nexml_get.R' 'nexml_methods.R' 'nexml_publish.R'
        'nexml_read.R' 'nexml_validate.R' 'nexml_write.R'
        'prefixed-uris.R' 'simmap.R' 'taxize_nexml.R' 'tbl_df.R'
        'utils.R'
RoxygenNote 6.1.1
X-schema.org-applicationCategory Data Publication
X-schema.org-keywords metadata, nexml, phylogenetics, linked-data
X-schema.org-isPartOf https://ropensci.org
Language en-US
Encoding UTF-8
NeedsCompilation no
Author Carl Boettiger [cre, aut] (<https://orcid.org/0000-0002-1642-628X>),
Scott Chamberlain [aut] (<https://orcid.org/0000-0003-1444-9135>),
Hilmar Lapp [aut] (<https://orcid.org/0000-0001-9107-0714>),
Kseniia Shumelchyk [aut],
Rutger Vos [aut] (<https://orcid.org/0000-0001-9254-7318>)
Maintainer Carl Boettiger <cboettig@gmail.com>
Repository CRAN
Date/Publication 2019-01-24 09:10:08 UTC

R topics documented:

.. callGeneric .................................................. 3
.. methodWithNext ............................................. 4
.. sigLabel ..................................................... 5
add_basic_meta ................................................. 5
add_characters ................................................ 7
add_meta .......................................................... 7
add_namespaces .................................................. 8
add_trees ......................................................... 10
Annotated-class .................................................. 10
c.meta-method .................................................. 11
c.nexml-method ................................................. 11
coaalesce_ ........................................................ 12
findNextMethod .................................................. 13
flatten_prefix ................................................... 14
findNextMethod .................................................. 14
flatten_multiphylo ............................................... 15
get_all_meta .................................................... 15
get_characters ................................................... 16
get_characters_list ............................................. 17
get_citation ...................................................... 18
get_flat_trees ................................................... 19
get_level .......................................................... 19
get_license ....................................................... 20
get_meta .......................................................... 21
get_metadata ...................................................... 22
get_metadata_values ............................................ 23
get_namespaces ................................................... 23
get_rdf ............................................................ 24
.callGeneric

Calls the given generic with the given arguments

Description

Calls the given generic with the given arguments, using the method whose signature matches the arguments.

Usage

.callGeneric(f, ..., .package = NULL)

Arguments

f the generic, as a character string or a standardGeneric object

... the arguments (named and/or unnamed) with which to call the matching method

.package the package name for finding the generic (if f is a character string); by default the package is determined from the calling environment
Details

Uses `methods::selectMethod()` to find the matching method. In theory, this is at best wholly redundant with what standard S4 generics already do by themselves. However, the generics dispatch for S4 seems (at least currently) broken at least if the first argument in the signature is a class that name-clashes with a class defined in another package. In that case, whether the standard dispatch works correctly or not can depend on search order, and can change within a session depending on the order in which packages are loaded.

Value

the value returned by the method

---

`.methodWithNext` Saves the next method in the method meta data

Description

Promotes the given method definition to an instance of `MethodWithNext`, thereby recording the next method in the `nextMethod` slot.

Usage

`.methodWithNext(method, nextMethod, .cache = FALSE)`

Arguments

- `method` the `MethodDefinition` object to promote
- `nextMethod` the `MethodDefinition` object to record as the next method
- `.cache` whether to cache the promoted method definition object (using `methods::cacheMethod()`)

Value

an instance of `MethodWithNext`, which has the next method in the `nextMethod` slot

Note

`MethodWithNext` objects are normally returned by `methods::addNextMethod()`, but a constructor function for the class seems missing (or is undocumented?). This provides one.
.sigLabel  

Create a label for a method signature

Description

Creates a label for a signature mirroring the result of .sigLabel() in the methods package, which unfortunately does not export the function. This is needed, for example, for the excluded slot in the MethodWithNext class.

Usage

.sigLabel(signature)

Arguments

signature  the signature for which to create a label, as a vector or list of strings, or as an instance of signature.

Value

a character string

add_basic_meta  

Add basic metadata

Description

adds Dublin Core metadata elements to (top-level) nexml

Usage

add_basic_meta(title = NULL, description = NULL, creator = Sys.getenv("USER"), pubdate = NULL, rights = "CC0", publisher = NULL, citation = NULL, nexml = new("nexml"))

Arguments

title  A title for the dataset
description  a description of the dataset
creator  name of the data creator. Can be a string or R person object
pubdate  publication date. Default is current date.
add_basic_meta

rights  the intellectual property rights associated with the data. The default is Creative Commons Zero (CC0) public domain declaration, compatible with all other licenses and appropriate for deposition into the Dryad or figshare repositories. CC0 is also recommended by the Panton Principles. Alternatively, any other plain text string can be added and will be provided as the content attribute to the dc:rights property.

publisher  the publisher of the dataset. Usually where a user may go to find the canonical copy of the dataset: could be a repository, journal, or academic institution.

citation  a citation associated with the data. Usually an academic journal article that indicates how the data should be cited in an academic context. Multiple citations can be included here. citation can be a plain text object, but is preferably an R citation or bibentry object (which can include multiple citations). See examples

nexml  a nexml object to which metadata should be added. A new nexml object will be created if none exists.

Details

add_basic_meta() is just a wrapper for add_meta to make it easy to provide generic metadata without explicitly providing the namespace. For instance, add_basic_meta(title="My title", description="a description", creator="Carl Boettiger <cboettig@gmail.com>", publisher="unpublished data", pubdate="2012-04-01") is identical to: add_meta(list(meta("dc:title", "My title"), meta("dc:description", "a description")))

Most function arguments are mapped directly to the Dublin Core terms of the same name, with the exception of rights, which by default maps to the Creative Commons namespace when using CC0 license.

Value

an updated nexml object

See Also

add_trees add_characters add_meta

Examples

nex <- add_basic_meta(title = "My test title",
                      description = "A description of my test",
                      creator = "Carl Boettiger <cboettig@gmail.com>",
                      publisher = "unpublished data",
                      pubdate = "2012-04-01")

## Adding citation to an R package:
nexml <- add_basic_meta(citation = citation("ape"))

## Not run:
## Use knitr citations package to add a citation by DOI:
library(knitr)
nexml <- add_basic_meta(citation = bib_metadata("10.2307/2408428"))

## End(Not run)
add_characters

Description
Add character data to a nexml object

Usage
add_characters(x, nexml = new("nexml"),
               append_to_existing_otus = FALSE)

Arguments
x character data, in which character traits labels are column names and taxon labels are row names. x can be in matrix or data.frame format.
nexml a nexml object, if appending character table to an existing nexml object. If omitted will initiate a new nexml object.
append_to_existing_otus logical. If TRUE, will add any new taxa (taxa not matching any existing otus block) to the existing (first) otus block. Otherwise (default), a new otus block is created, even though it may contain duplicate taxa to those already present. While FALSE is the safe option, TRUE may be appropriate when building nexml files from scratch with both characters and trees.

Examples
library("geiger")
data(geospiza)
geiger_nex <- add_characters(geospiza$dat)

add_meta

Description
Add metadata to a nexml file

Usage
add_meta(meta, nexml = new("nexml"), level = c("nexml", "otus", "trees", "characters"), namespaces = NULL, i = 1, at_id = NULL)
Arguments

`meta` a meta S4 object, e.g. output of the function `meta`, or a list of these meta objects
`nexml` (S4) object
`level` the level at which the metadata annotation should be added.
`namespaces` named character string for any additional namespaces that should be defined.
`
for otus, trees, characters: if there are multiple such blocks, which one should be annotated? Default is first/only block.
`at_id` the id of the element to be annotated. Optional, advanced use only.

Value

the updated nexml object

See Also

`meta add_trees add_characters add_basic_meta`

Examples

```r
## Create a new nexml object with a single metadata element:
modified <- meta(property = "prism:modificationDate", content = "2013-10-04")
nex <- add_meta(modified) # Note: 'prism' is defined in nexml_namespaces by default.

## Write multiple metadata elements, including a new namespace:
website <- meta(href = "http://carlboettiger.info",
rel = "foaf:homepage") # meta can be link-style metadata
nex <- add_meta(list(modified, website),
namespaces = c(foaf = "http://xmlns.com/foaf/0.1/"))

## Append more metadata, and specify a level:
history <- meta(property = "skos:historyNote",
content = "Mapped from the bird.orders data in the ape package using RNexML")
data(bird.orders)
nex <- add_trees(bird.orders) # need to have created a trees block first
nex <- add_meta(history,
nexml = nex,
level = "trees",
namespaces = c(skos = "http://www.w3.org/2004/02/skos/core#"))
```

Description

Add namespaces and their prefixes as a named vector of URIs, with the names being the prefixes. Namespaces have most relevance for meta objects’ rel and property, and for embedded XML literals.
add_namespaces

Usage

add_namespaces(namespaces, nexml = new("nexml"))

Arguments

namespaces     a named character vector of namespaces
nexml          a nexml object. will create a new one if none is given.

Details

The implementation attempts to avoid duplication, currently using the prefix. I.e., namespaces with prefixes already defined will not get added. Namespaces needed by the NeXML format, and for commonly used metadata terms, are already included by default, see `get_namespaces()`.

Value

a nexml object with updated namespaces

Note

Often a user won’t call this directly, but instead provide the namespace(s) through `add_meta()`.

See Also

`meta()` `add_meta()` `get_namespaces()`

Examples

```r
## Write multiple metadata elements, including a new namespace:
website <- meta(href = "http://carlboettiger.info",
               rel = "foaf:homepage")  # meta can be link-style metadata
modified <- meta(property = "prism:modificationDate",
               content = "2013-10-04")
nex <- add_meta(list(modified, website),
               namespaces = c(foaf = "http://xmlns.com/foaf/0.1/"))  # prism prefix already included by default

## Add namespace "by hand" before adding meta:
nex <- add_namespaces(c(skos = "http://www.w3.org/2004/02/skos/core"),
                       nexml = nex)
history <- meta(property = "skos:historyNote",
                content = "Mapped from the bird.orders data in the ape package using RNeXML")
nex <- add_meta(history, nexml = nex)
```
**add_trees**

**Description**

Add trees to an existing nexml file.

**Usage**

```r
add_trees(phy, nexml = new("nexml"), append_to_existing_otus = FALSE)
```

**Arguments**

- `phy`: A phylo object, multiPhylo object, or list of multiPhylo to be added to the nexml.
- `nexml`: A nexml object to which we should append this phylo. By default, a new nexml object will be created.
- `append_to_existing_otus`: Logical, indicating if we should make a new OTU block (default) or append to the existing one.

**Value**

A nexml object containing the phy in nexml format.

**Examples**

```r
library("geiger")
data(geospiza)
geiger_nex <- add_trees(geospiza$phy)
```

---

**Annotated-class**

*Class of objects that have metadata as lists of meta elements*

**Description**

Class of objects that have metadata as lists of meta elements

**Slots**

- `meta`: List of meta objects
- `about`: For RDF extraction, the identifier for the resource that this object is about
**c.meta-method**

*Concatenate meta elements into a ListOfmeta*

**Description**

Concatenate meta elements into a ListOfmeta

Concatenate ListOfmeta elements into a flat ListOfmeta

**Usage**

```r
## S4 method for signature 'meta'
c(x, ..., recursive = TRUE)
```

```r
## S4 method for signature 'ListOfmeta'
c(x, ..., recursive = TRUE)
```

**Arguments**

- `x, ...`: meta and ListOfmeta elements to be concatenated, see `meta`
- `recursive`: logical, if 'recursive=TRUE', the function recursively descends through lists and combines their elements into a flat vector. This method does not support `recursive=FALSE`, use `list` instead.

**Value**

a ListOfmeta object containing a flat list of meta elements.

**Examples**

```r
c(meta(content="example", property="dc:title"),
   meta(content="Carl", property="dc:creator"))
metalist <- c(meta(content="example", property="dc:title"),
              meta(content="Carl", property="dc:creator"))
out <- c(metalist, metalist)
out <- c(metalist, meta(content="a", property="b"))
```

---

**c.nexml-method**

*Concatenate nexml files*

**Description**

Concatenate nexml files

**Usage**

```r
## S4 method for signature 'nexml'
c(x, ..., recursive = FALSE)
```
Arguments

- `x, ...`: nexml objects to be concatenated, e.g. from `write.nexml` or `read.nexml`. Must have unique ids on all elements.
- `recursive`: logical. If `recursive = TRUE`, the function recursively descends through lists (and pairlists) combining all their elements into a vector. (Not implemented).

Value

- a concatenated nexml file

Examples

```r
## Not run:
f1 <- system.file("examples", "trees.xml", package="RNeXML")
f2 <- system.file("examples", "comp_analysis.xml", package="RNeXML")
nex1 <- read.nexml(f1)
nex2 <- read.nexml(f2)
nex <- c(nex1, nex2)

## End(Not run)
```

Description

Replaces any NULL argument with a vector of NA, and casts every vector to the same type as the last vector. After that, calls `dplyr::coalesce()`.

Usage

```r
coalessce_(...)```

Arguments

- `...`: the vectors to coalesce on NA

Value

- a vector of the same type and length as the last argument

See Also

`dplyr::coalesce()`
**Expand namespace-prefixed string**

**Description**
Substitutes the namespace prefix in the input vector of strings with the corresponding namespaces.

**Usage**
```r
expand_prefix(x, namespaces = NULL)
```

**Arguments**
- `x`: a character vector of potentially namespace-prefixed strings
- `namespaces`: a named vector of namespaces, with namespace prefixes being the names. A "base" namespace with an empty name can be included. If not provided, or if empty, the input vector is returned as is.

**Details**
Namespace prefixes are expected to be separated by one or more semicolons. Prefixes that cannot be matched to the vector of namespaces will be left as is. For strings that do not have a namespace prefix, the vector of namespaces can contain a base namespace, identified as not having a name, with which these strings will be expanded.

**Value**
a character vector, of the same length as the input vector

**Examples**
```r
uris <- c("cc:license", "dc:title")
ns <- c(dc = "http://purl.org/dc/elements/1.1/",
        dcterms = "http://purl.org/dc/terms/",
        dct = "http://purl.org/dc/terms/",
        cc = "http://creativecommons.org/ns#")
# expansion is vectorized
expand_prefix(uris, ns)

# strings with non-matching or no prefix are left as is
uris <- c(uris, "my:title", "title")
expand_prefix(uris, ns)

# NAs in the input list become NA in the output
uris <- c(uris, NA)
expand_prefix(uris, ns)

# can include a "base" (unnamed) namespace for expanding unprefixed strings
ns <- c(ns, "//local/")
```
findNextMethod

findNextMethod <- expand_prefix(uris, ns)
xuris
xuris[uris == "title"] = paste0("/\local/", uris[uris == "title"])

# different prefixes may expand to the same result
expand_prefix("dcterms:modified", ns) == expand_prefix("dct:modified", ns)

# or they may result in different expansions
expand_prefix("dc:title", ns) != expand_prefix("dcterms:title", ns)

findNextMethod(method, f = NULL, envir = topenv())

Arguments

method MethodDefinition, the method for which to find the next method
f standardGeneric, the standard generic for which to find the next method. By
default this will be obtained from method.
envir the environment in which to find the method

Details

The next method will be determined by the S4 inheritance chain. However, this function will walk
only the inheritance chain of those arguments in the signature that are defined in the package of the
generic method from which this function was invoked (directly or indirectly). If there are no such
parameters in the signature, or if there is more than one, finding the next method is handed off to
methods::addNextMethod().

Value

a MethodDefinition object that is the next method in the chain by inheritance
Note

In theory a class name clash between packages shouldn’t be a problem because class names can
be namespaced, and the MethodDefinition object passed to addNextMethod() has all the necessary
namespace information. Hopefully, at some point this gets fixed in R, and then we don’t need this
anymore.

flatten_multiphylo

Flatten a multiphylo object

Description

Flatten a multiphylo object

Usage

flatten_multiphylo(object)

Arguments

object a list of multiphylo objects

Details

NeXML has the concept of multiple <trees> nodes, each with multiple child <tree> nodes. This
maps naturally to a list of multiphylo objects. Sometimes this hierarchy conveys important structural
information, so it is not discarded by default. Occasionally it is useful to flatten the structure though,
and hence this function. Note that this discards the original structure, and the nexml file must be parsed
again to recover it.

get_all_meta

Get flattened list of meta annotations

Description

Collects recursively (in the case of nested meta annotations) all meta object annotations for the
given object, and returns the result as a flat list.

Usage

get_all_meta(annotated)

Arguments

annotated the object from which to extract meta object annotations
get_characters

Details

Does not check that the input object can actually have meta annotations. An invalid slot error will be generated if it can’t.

Value

a flat list of meta objects

get_characters HnexL rownames_as_col \] falseL otu_id \] falseL otus_id \] falseL include_state_types \] falseI

Arguments

nex a nexml object
rownames_as_col option to return character matrix rownames (with taxon ids) as it’s own column in the data.frame. Default is FALSE for compatibility with geiger and similar packages.

otu_id logical, default FALSE. return a column with the otu id (for joining with otu metadata, etc)

otus_id logical, default FALSE. return a column with the otus block id (for joining with otu metadata, etc)

include_state_types logical, default FALSE. whether to also return a matrix of state types (with values standard, polymorphic, and uncertain)

Details

RNeXML will attempt to return the matrix using the NeXML taxon (otu) labels to name the rows and the NeXML char labels to name the traits (columns). If these are unavailable or not unique, the NeXML id values for the otus or traits will be used instead.

Value

the character matrix as a data.frame, or if include_state_types is TRUE a list of two elements, characters as the character matrix, and state_types as a matrix of state types. Both matrices will be in the same ordering of rows and columns.
get_characters_list

Examples

## Not run:
# A simple example with a discrete and a continuos trait
f <- system.file("examples", "comp_analysis.xml", package="RNeXML")
nex <- read.nexml(f)
get_characters(nex)

# A more complex example -- currently ignores sequence-type characters
f <- system.file("examples", "characters.xml", package="RNeXML")
nex <- read.nexml(f)
get_characters(nex)

# if polymorphic or uncertain states need special treatment, request state
# types to be returned as well:
f <- system.file("examples", "ontotrace-result.xml", package="RNeXML")
nex <- read.nexml(f)
res <- get_characters(nex, include_state_types = TRUE)
row.has.p <- apply(res$state_types, 1,
    function(x) any(x == "polymorphic", na.rm = TRUE))
col.has.p <- apply(res$state_types, 2,
    function(x) any(x == "polymorphic", na.rm = TRUE))
res$characters[row.has.p, col.has.p, drop=FALSE] # polymorphic rows and cols
res$characters[row.has.p, drop=FALSE] # drop taxa with polymorphic states
# replace polymorphic state symbols in matrix with '?'
m1 <- mapply(function(s, s.t) ifelse(s.t == "standard", s, "?")
    , res$characters, res$state_types)
row.names(m1) <- row.names(res$characters)
m1

## End(Not run)

---

get_characters_list  

*Extract the character matrix*

Description

Extract the character matrix

Usage

get_characters_list(nexml, rownames_as_col = FALSE)

Arguments

nexml  
nexml object (e.g. from read.nexml)

rownames_as_col  
option to return character matrix rownames (with taxon ids) as it’s own column
in the data.frame. Default is FALSE for compatibility with geiger and similar packages.
Value

the list of taxa

Examples

comp_analysis <- system.file("examples", "comp_analysis.xml", package="RNeXML")
nex <- nxml_read(comp_analysis)
get_characters_list(nex)

---

get_citation  Get citation from metadata

Description

Extracts the citation annotation from the metadata annotation of the nexml object, and returns its value.

Usage

gcrt_citation(nexml)

Arguments

nexml a nexml object

Details

Currently the implementation looks for dcterms:bibliographicCitation annotations. (Note that these may be given with any prefix in the metadata so long as they expand to the same full property URIs.)

Value

the citation if the metadata provides one that is non-empty, and NA otherwise. If multiple non-empty annotations are found, only the first one is returned.

See Also

gcrt_metadata_values()
**get_flat_trees**

Description

extract a single multiPhylo object containing all trees in the nexml

Usage

get_flat_trees(nexml)

Arguments

nexml a representation of the nexml object from which the data is to be retrieved

Details

Note that this method collapses any hierarchical structure that may have been present as multiple trees nodes in the original nexml (though such a feature is rarely used). To preserve that structure, use get_trees instead.

Value

a multiPhylo object (list of ape::phylo objects). See details.

See Also

get_trees get_trees get_item

Examples

```r
comp_analysis <- system.file("examples", "comp_analysis.xml", package="RNeXML")
nex <- nexml_read(comp_analysis)
get_flat_trees(nex)
```

**get_level**

Description

get a data.frame of attribute values of a given node

Usage

get_level(nex, level)"
get_license

Arguments

nex      a nexus object
level    a character vector indicating the class of node, see details

Details

level should be a character vector giving the path to the specified node group. For instance, otus, characters, and trees are top-level blocks (e.g. child nodes of the root nexus block), and can be specified directly. To get metadata for all "char" elements from all characters blocks, you must specify that char nodes are child nodes to character nodes: e.g. get_level(nex, "characters/char"), or similarly for states: get_level(nex, characters/states).

The return object is a data frame whose columns are the attribute names of the elements specified. The column names match the attribute names except for "id" attribute, for which the column is renamed using the node itself. (Thus <otus id="os2"> would be rendered in a data.frame with column called "otus" instead of "id"). Additional columns are added for each parent element in the path; e.g. get_level(nex, "otus/otu") would include a column named "otus" with the id of each otus block. Even though the method always returns the data frame for all matching nodes in all blocks, these ids let you see which otu values came from which otus block. This is identical to the function call get_taxa(). Similarly, get_level(nex, "otus/otu/meta") would return additional columns 'otus' and also a column, 'otu', with the otu parent ids of each metadata block. (This is identical to a function call to get_metadata). This makes it easier to join data.frames as well, see examples

Value

Returns the attributes of specified class of nodes as a data.frame

get_license

Get license from metadata

Description

Extracts the license annotation from the metadata annotation of the nexus object, and returns its value.

Usage

generate_license(nexml)

Arguments

nexml      a nexus object

Details

Currently the implementation looks for cc:license and dc:-rights annotations. (Note that these may be given with any prefix in the metadata so long as they expand to the same full property URIs.)
Value

the license if the metadata asserts one that is non-empty, and NA otherwise. If multiple non-empty annotations are found, only the first one is returned.

See Also

get_metadata_values()

get_meta

Extracts meta objects matching properties

Description

Extracts the metadata annotations for the given property or properties, and returns the result as a list of meta objects.

Usage

get_meta(nexml, annotated = NULL, props)

Arguments

nexml
  a nexml object
annotated
  the nexml component object from which to obtain metadata annotations, or a list of such objects. Defaults to the nexml object itself.
props
  a character vector of property names for which to extract metadata annotations

Details

For matching property identifiers (i.e., URIs), prefixes in the input list as well as in the annotated object will be expanded using the namespaces of the nexml object. Names in the returned list are mapped to the (possibly prefixed) form in the input list. The resulting list is flat, and hence does not retain the nesting hierarchy in the object’s annotation.

Value

a named list of the matching meta objects
get_metadata

Description

get_metadata

Usage

get_metadata(nexml, level = "nexml", simplify = TRUE)

Arguments

nexml
  a nexml object
level
  the name of the level of element desired, see details
simplify
  logical, see Details

Details

'level' should be either the name of a child element of a NeXML document (e.g. "otu", "characters"), or a path to the desired element, e.g. 'trees/tree' will return the metadata for all phylogenies in all trees blocks.

If a metadata element has other metadata elements nested within it, the nested metadata are returned as well. A column "Meta" will contain the IDs consolidated from the type-specific LiteralMeta and ResourceMeta columns, and IDs are generated for meta elements that have nested elements but do not have an ID ("blank nodes"). A column "meta" contains the IDs of the parent meta elements for nested ones. This means that the resulting table can be self-joined on those columns.

If simplify is FALSE, the type-specific "LiteralMeta" and "ResourceMeta" columns will be retained even if a consolidated "Meta" column is present. Otherwise, only the consolidated column will be included in the result. Also, if simplify is TRUE the values for "property" (LiteralMeta) and "rel" (ResourceMeta) will be consolidated to "property", and "rel" will be removed from the result.

Value

the requested metadata as a data.frame. Additional columns indicate the parent element of the return value.

Examples

```r
## Not run:
comp_analysis <- system.file("examples", "primates.xml", package="RNeXML")
nex <- nexml_read(comp_analysis)
get_metadata(nex)
get_metadata(nex, "otus/otu")

## End(Not run)
```
get_metadata_values

### Description

Extracts the values from the metadata annotations for the given property or properties, and returns the result.

### Usage

```
get_metadata_values(nexml, annotated = NULL, props)
```

### Arguments

- **nexml**: a `nexml` object
- **annotated**: the `nexml` component object from which to obtain metadata annotations, defaults to the `nexml` object itself
- **props**: a character vector of property names for which to extract metadata annotations

### Details

For matching property identifiers (i.e., URIs), prefixes in the input list as well as in the `annotated` object will be expanded using the namespaces of the `nexml` object. Names in the returned vector are mapped to the (possibly prefixed) form in the input list.

### Value

A named character vector, giving the values and names being the property names

---

get_namespaces

### Description

get namespaces

### Usage

```
get_namespaces(nexml)
```

### Arguments

- **nexml**: a `nexml` object
Value

a named character vector providing the URLs defining each of the namespaces used in the nexml file. Names correspond to the prefix abbreviations of the namespaces.

Examples

```r
comp_analysis <- system.file("examples", "comp_analysis.xml", package="RNeXML")
nex <- nexml_read(comp_analysis)
get_namespaces(nex)
```

---

get_rdf

Extract rdf-xml from a NeXML file

Description

Extract rdf-xml from a NeXML file

Usage

`get_rdf(file)`

Arguments

- **file**: the name of a nexml file, or otherwise a nexml object.

Value

an RDF-XML object (XMLInternalDocument). This can be manipulated with tools from the XML R package, or converted into a triplestore for use with SPARQL queries from the rdflib R package.

---

get_taxa

Retrieve names of all species/otus (operational taxonomic units) included in the nexml

Description

Retrieve names of all species/otus (operational taxonomic units) included in the nexml

Usage

`get_taxa(nexml)`

Arguments

- **nexml**: a nexml object
**get_taxa_list**

**Value**

the list of taxa

**See Also**

`get_item`

**Examples**

```r
comp_analysis <- system.file("examples", "comp_analysis.xml", package="RNeXML")
nex <- nxml_read(comp_analysis)
get_taxa(nex)
```

**Description**

Retrieve names of all species/otus (operational taxonomic units) included in the nxml

**Usage**

```
get_taxa_list(nexml)
```

**Arguments**

- **nexml**
  - a nxml object

**Value**

the list of taxa

**See Also**

`get_item`
get_trees

extract a phylogenetic tree from the nexml

Description

extract a phylogenetic tree from the nexml

Usage

get_trees(nexml)

Arguments

nexml  a representation of the nexml object from which the data is to be retrieved

Value

an ape::phylo tree, if only one tree is represented. Otherwise returns a list of lists of multiphylo trees. To consistently receive the list of lists format (preserving the hierarchical nature of the nexml), use get_trees_list instead.

See Also

get_trees get_flat_trees get_item

Examples

comp_analysis <- system.file("examples", "comp_analysis.xml", package="RNeXML")
nex <- nxml_read(comp_analysis)
get_trees(nex)

get_trees_list

extract all phylogenetic trees in ape format

Description

extract all phylogenetic trees in ape format

Usage

get_trees_list(nexml)

Arguments

nexml  a representation of the nexml object from which the data is to be retrieved
**lcapply**

*Compact list then lapply*

**Value**

returns a list of lists of multiphylo trees, even if all trees are in the same `trees` node (and hence the outer list will be of length 1. or if there is only a single tree (and hence the inner list will also be of length 1. This ensures a consistent return type regardless of the number of trees present in the nexml file, and also preserves any grouping of trees.

**See Also**

`get_trees` `get_flat_trees` `get_item`

**Examples**

```r
comp_analysis <- system.file("examples", "comp_analysis.xml", package="RNeXML")
nex <- nxml_read(comp_analysis)
get_trees_list(nex)
```

**Description**

Compacts the list (i.e., removes NULL objects), then calls `lapply()` on the result with the remaining parameters.

**Usage**

`lcapply(x, ...)`

**Arguments**

- `x` the list object
- `...` remaining arguments to `lapply()`
Constructor function for metadata nodes

Description

Constructor function for metadata nodes

Usage

meta(property = NULL, content = NULL, rel = NULL, href = NULL,
   datatype = NULL, id = NULL, type = NULL, children = list())

Arguments

property  specify the ontological definition together with it's namespace, e.g. dc:title
content   content of the metadata field
rel       Ontological definition of the reference provided in href
href      A link to some reference
datatype  optional RDFa field
id        optional id element (otherwise id will be automatically generated).
type      optional xsi:type. If not given, will use either "LiteralMeta" or "ResourceMeta"
   as determined by the presence of either a property or a href value.
children  Optional element containing any valid XML block (XMLInternalElementNode
   class, see the XML package for details).

Details

User must either provide property+content or rel+href. Mixing these will result in potential garbage.
The datatype attribute will be detected automatically from the class of the content argument. Maps
from R class to schema datatypes are as follows: character - xs:string, Date - xs:date, integer -
xs:integer, numeric - xs:decimal, logical - xs:boolean

See Also

nexml_write

Examples

meta(content="example", property="dc:title")
new with namespaced class name

Description
Convenience function for `methods::new()` that ensures that the provided class name is namespaced with a package name.

Usage
```
New(Class, ...)
```

Arguments
- **Class** the name of the S4 class to be instantiated
- **...** additional parameters for `methods::new()`

Details
If the provided class name is not already namespaced (see `methods::packageSlot()`), it will be namespaced with this package. This mechanism is used by `new()` to disambiguate if the class name clashes with a class defined in another package.

Note
This may not completely eliminate messages on standard error about classes with the same name having been found in different packages. If they appear, they will most likely have come from the call to the `methods::initialize()` generic that `new()` issues at the end.

nexml-class

Class representing a NeXML document

Description
The `nexml` class represents a NeXML document, and is the top of the class hierarchy defined in this package, corresponding to the root node of the corresponding XML document.

Details
Normally objects of this type are created by the package as a result of reading a NeXML file, or of converting from another type, such as `ape::phylo`. Also, interacting directly with the slots of the class is normally not necessary. Instead, use the `get Xxx()` and `add Xxx()` functions in the API.
Slots

trees list, corresponding to the list of `<trees/>` elements in NeXML. Elements will be of class `trees`.

characters list, corresponding to the list of `<characters/>` elements in NeXML. Elements will be of class `characters`.

otus list, corresponding to the list of `<otus/>` elements in NeXML. Elements will be of class `otus`.

about inherited, see `Annotated`

meta inherited, see `Annotated`

`xsi:type` for internal use

version NeXML schema version, do not change

generator name of software generating the XML

`xsi:schemaLocation` for internal use, do not change

namespaces named character vector giving the XML namespaces

See Also

`read.nexml()`

Examples

```r
nex <- nexml() # a `nexml` object with no further content
nex <- new("nexml") # accomplishes the same thing
nex@generator
length(nex@trees)

data(bird.orders)
nex <- as(bird.orders, "nexml")
summary(nex)
length(nex@trees)
```

---

**nexml.tree**

Constructor for the respective class

Description

Creates an instance of the class corresponding to the respective NeXML element, and initializes its slots with the provided parameters, if any.
Usage

nexml.tree(...)
nexml.trees(...)
nexml.node(...)
nexml.edge(...)
nexml.otu(...)
nexml.otus(...)
nexml.char(...)
nexml.characters(...)
nexml.format(...)
nexml.state(...)
nexml.uncertain_state(...)
nexml.states(...)
nexml.uncertain_states(...)
nexml.polymorphic_states(...)
nexml.member(...)
nexml.matrix(...)
nexml.row(...)
nexml.seq(...)
nexml.cell(...)

Arguments

... optionally, parameters passed on to new()

Details

Usually, users won’t need to invoke this directly.
**nexml_add**

*add elements to a new or existing nexml object*

**Description**

add elements to a new or existing nexml object

**Usage**

nexml_add(x, nexml = new("nexml"), type = c("trees", "characters", "meta", "namespaces"), ...)

**Arguments**

- **x**: the object to be added
- **nexml**: an existing nexml object onto which the object should be appended
- **type**: the type of object being provided.
- **...**: additional optional arguments to the add functions

**Value**

a nexml object with the additional data

**See Also**

`add_trees add_characters add_meta add_namespaces`

**Examples**

```r
library("geiger")
data(geospiza)
geiger_nex <- nexml_add(geospiza$phy, type="trees")
geiger_nex <- nexml_add(geospiza$dat, nexml = geiger_nex, type="characters")
```
Description

publish nexml to figshare

Usage

```r
description = "publish nexml to figshare"

tags = list("phylogeny", "NeXML")
visibility = c("public", "private", "draft")
```

Arguments

- **nexml**: a nexml object (or file path to a nexml file)
- **file**: The filename desired for the object, if nexml is not already a file. If the first argument is already a path, this value is ignored.
- **categories**: The figshare categories, must match available set. See `fs_add_categories`
- **tags**: Any keyword tags you want to add to the data.
- **visibility**: Whether the results should be published (public), or kept private, or kept as a draft for further editing before publication. (New versions can be updated, but any former versions that was once made public will always be archived and cannot be removed).
- **id**: An existing figshare id (e.g. from `fs_create`), to which this file can be appended.
- **...**: Additional arguments

Value

The figshare id of the object

Examples

```r
## Not run:
data(bird.orders)
birds <- add_trees(bird.orders)
doi <- nexml_figshare(birds, visibility = "public", repository="figshare")
```

## End(Not run)
nexml_get

Get the desired element from the nexml object

Description

Get the desired element from the nexml object.

Usage

nexml_get(nexml, element = c("trees", "trees_list", "flat_trees", "metadata", "otu", "taxa", "characters", "characters_list", "namespaces"), ...)

Arguments

- `nexml`: a nexml object (from read_nexml)
- `element`: the kind of object desired, see details.
- `...`: additional arguments, if applicable to certain elements

Details

- "tree" an ape::phylo tree, if only one tree is represented. Otherwise returns a list of lists of multiphylo trees. To consistently receive the list of lists format (preserving the hierarchical nature of the nexml), use `trees` instead.
- "trees" returns a list of lists of multiphylo trees, even if all trees are in the same `trees` node (and hence the outer list will be of length 1) or if there is only a single tree (and hence the inner list will also be of length 1. This ensures a consistent return type regardless of the number of trees present in the nexml file, and also preserves any hierarchy/grouping of trees.
- "flat_trees" a multiPhylo object (list of ape::phylo objects) Note that this method collapses any hierarchical structure that may have been present as multiple trees nodes in the original nexml (though such a feature is rarely used). To preserve that structure, use `trees` instead.
- "metadata"Get metadata from the specified level (default is top/nexml level)
- "otu" returns a named character vector containing all available metadata. names indicate property (or rel in the case of links/resourceMeta), while values indicate the content (or href for links).
- "taxa" alias for `otu`

For a slightly cleaner interface, each of these elements is also defined as an S4 method for a nexml object. So in place of `get_item(nexml, "tree")`, one could use `get_tree(nexml)`, and so forth for each element type.

Value

return type depends on the element requested. See details.
nexml_publish

See Also
get_trees

Examples

```r
comp_analysis <- system.file("examples", "comp_analysis.xml", package="RNeXML")
nex <- nexml_read(comp_analysis)
nexml_get(nex, "trees")
nexml_get(nex, "characters_list")
```

---

```
library(nexml)
library(doi)
library(phylo)
library(phytools)
library(figshare)
library(RNeXML)
```

```
# load the bird orders data

# add the trees

# publish the data

doib <- nexml_publish(birds, visibility = "public", repository="figshare")
```

## Not run:
```
data(bird.orders)
birds <- add_trees(bird.orders)
doi <- nexml_publish(birds, visibility = "public", repository="figshare")

## End(Not run)
```

---

**nexml_publish**

*publish nexml files to the web and receive a DOI*

**Description**

publish nexml files to the web and receive a DOI

**Usage**

```
nexml_publish(nexml, ..., repository = "figshare")
```

**Arguments**

- `nexml`: a nexml object (or file path)
- `...`: additional arguments, depending on repository. See examples.
- `repository`: destination repository

**Value**

a digital object identifier to the published data

**Examples**

```
## Not run:
data(bird.orders)
birds <- add_trees(bird.orders)
doi <- nexml_publish(birds, visibility = "public", repository="figshare")

## End(Not run)
```
nexml_read

Read NeXML files into various R formats

Description

Read NeXML files into various R formats

Usage

nexml_read(x, ...)

## S3 method for class 'character'
nexml_read(x, ...)

## S3 method for class 'XMLInternalDocument'
nexml_read(x, ...)

## S3 method for class 'XMLInternalNode'
nexml_read(x, ...)

Arguments

x Path to the file to be read in. Or an XMLInternalDocument-class or XMLInternalNode-class
... Further arguments passed on to xmlParse

Examples

# file
f <- system.file("examples", "trees.xml", package="RNeXML")
nexml_read(f)
## Not run: # may take > 5 s
# url
url <- "https://raw.githubusercontent.com/ropensci/RNeXML/master/inst/examples/trees.xml"
nexml_read(url)
# character string of XML
str <- paste0(readLines(f), collapse = "")
nexml_read(str)
# XMLInternalDocument
library("httr")
library("XML")
x <- xmlParse(content(GET(url)))
nexml_read(x)
# XMLInternalNode
nexml_read(xmlRoot(x))

## End(Not run)
nexml_validate

validate nexml using the online validator tool

**Description**

validate nexml using the online validator tool

**Usage**

nexml_validate(file, schema = CANONICAL_SCHEMA)

**Arguments**

- **file**: path to the nexml file to validate
- **schema**: URL of schema (for fallback method only, set by default).

**Details**

Requires an internet connection. see http://www.nexml.org/nexml/phylows/validator for more information in debugging invalid files

**Value**

TRUE if the file is valid, FALSE or error message otherwise

**Examples**

```r
## Not run:
data(bird.orders)
birds <- nexml_write(bird.orders, "birds_orders.xml")
nexml_validate("birds_orders.xml")
unlink("birds_orders.xml") # delete file to clean up

## End(Not run)
```

nexml_write

Write nexml files

**Description**

Write nexml files

**Usage**

nexml_write(x = nexml(), file = NULL, trees = NULL,
characters = NULL, meta = NULL, ...)
Arguments

- **x**: a `nexml` object, or any phylogeny object (e.g., `phylo`, `phylo4`) that can be coerced into one. Can also be omitted, in which case a new `nexml` object will be constructed with the additional parameters specified.

- **file**: the name of the file to write out

- **trees**: phylogenetic trees to add to the `nexml` file (if not already given in `x`) see `add_trees` for details.

- **characters**: additional characters

- **meta**: A meta element or list of meta elements, see `add_meta` for details.

- **...**: additional arguments to add__basic_meta, such as the title. See `add_basic_meta`.

Value

Writes out a `nexml` file

See Also

- `add_trees`
- `add_characters`
- `add_meta`
- `nexml_read`

Examples

```r
## Write an ape tree to nexml, analogous to write.nexus:
library(ape); data(bird.orders)
write.nexml(bird.orders, file="example.xml")

## Not run: # takes > 5s
## Assemble a nexml section by section and then write to file:
library(geiger)
data(geospiza)
	nexml <- add_trees(geospiza$phy) # creates new nexml
tnexml <- add_characters(geospiza$dat, nexml = nexml) # pass the nexml obj to append character data
tnexml <- add_basic_meta(title="my title", creator = "Carl Boettiger", nexml = nexml)
tnexml <- add_meta(meta("prism:modificationDate", format(Sys.Date())), nexml = nexml)

write.nexml(nexml, file="example.xml")

## As above, but in one call (except for add_meta() call).
write.nexml(trees = geospiza$phy,
characters = geospiza$dat,
title = "My title",
creator = "Carl Boettiger",
file = "example.xml")

## Mix and match: identical to the section by section:
nexml <- add_meta(meta("prism:modificationDate", format(Sys.Date())))
write.nexml(x = nexml,
trees = geospiza$phy,
characters = geospiza$dat,
title = "My title",
...)
```
Description
reset the id counter

Usage
reset_id_counter()

simmap_to_nexml  Convert phylo with attached simmap to nexml object

Description
Convert phylo with attached simmap to nexml object
Convert nexml object with simmap to phylo

Usage
simmap_to_nexml(phy, state_ids = NULL)
nexml_to_simmap(nexml)

Arguments
phy  a phylo object containing simmap phy$maps element, from the phytools package
state_ids  a named character vector giving the state names corresponding to the ids used to refer to each state in nexml. If null ids will be generated and states taken from the phy$states names.
nexml  a nexml object

Value
a nexml representation of the simmap
a simmap object (phylo object with a $maps element for use in phytools functions).
slot,ResourceMeta-method

Access or set slot of S4 object

Description

See `methods::slot()`. This version allows using "property" consistently for both LiteralMeta and ResourceMeta (which internally uses "rel" because RDFa does), which is easier to program. It also allows using "meta" as an alias for "children" for ResourceMeta, to be consistent with the corresponding slot for instances of Annotated.

Usage

```r
## S4 method for signature 'ResourceMeta'
slot(object, name)

## S4 replacement method for signature 'ResourceMeta'
slot(object, name) <- value
```

Arguments

- `object`: the object
- `name`: name of the slot
- `value`: the new value

See Also

`methods::slot()`
**taxize_nexml**

---

### Description

Check taxonomic names against the specified service and add appropriate semantic metadata to the `nexml` OTU unit containing the corresponding identifier.

### Usage

```r
taxize_nexml(nexml, type = c("ncbi", "itis", "col", "tpl", "gbif", "wd"),
             warnings = TRUE, ...)
```

### Arguments

- `nexml`: a `nexml` object
- `type`: the name of the identifier to use
- `warnings`: should we show warning messages if no match can be found?
- `...`: additional arguments to `taxadb::get_ids()`

### Examples

```r
## Not run:
data(bird.orders)
birds <- add_trees(bird.orders)
birds <- taxize_nexml(birds, "NCBI")
## End(Not run)
```

---

**toPhylo**

---

### Description

`nexml` to `phylo` coercion

### Usage

```r
toPhylo(tree, otus)
```

### Arguments

- `tree`: an `nexml` tree element
- `otus`: a character string of taxonomic labels, named by the `otu` ids. e.g. (from `get_otu_maps` for the `otus` set matching the relevant `trees` node.)
Value

phylo object. If a "reconstructions" annotation is found on the edges, return simmap maps slot as well.
Index

.getGeneric, 3
.methodWithNext, 4
.sigLabel, 5
add_basic_meta, 5, 8, 38
add_characters, 6, 7, 8, 32, 38
add_meta, 6, 7, 32, 38
add_namespaces, 8, 32
add_trees, 6, 8, 10, 32, 38
Annotated, 30
Annotated-class, 10
c, ListOfmeta-method (c, meta-method), 11
c, meta-method, 11
c, nexml-method, 11
c,ListOfmeta (c, meta-method), 11
c-meta (c, meta-method), 11
coalesce, 12
dplyr::coalesce(), 12
expand_prefix, 13
findNextMethod, 14
flatten_multiphylo, 15
get_all_meta, 15
get_characters, 16
get_characters_list, 17
get_citation, 18
get_flat_trees, 19, 26, 27
get_item, 19, 25–27
get_item (nexml_get), 34
get_level, 19
get_license, 20
get_meta, 21
get_metadata, 22
get_metadata_values, 23
get_metadata_values(), 18, 21
get_namespaces, 23
get_namespaces(), 9
get_otu (get_taxa), 24
get_otas_list (get_taxa_list), 25
get_rdf, 24
get_taxa, 24
get_taxa_list, 25
get_trees, 19, 26, 26, 27, 35
get_trees_list, 26, 26
lapply(), 27
lapply(), 27
list, 11
meta, 8, 11, 28
meta(), 9
MethodDefinition, 4, 14, 15
methods::addNextMethod(), 4, 14
methods::cacheMethod(), 4
methods::callNextMethod(), 14
methods::initialize(), 29
methods::new(), 29
methods::packageSlot(), 29
methods::selectMethod(), 4
methods::slot(), 40
MethodWithNext, 4, 5, 14
New, 29
new(), 31
nexml (nexml-class), 29
nexml-class, 29
nexml.cell (nexml.tree), 30
nexml.char (nexml.tree), 30
nexml.characters (nexml.tree), 30
nexml.edge (nexml.tree), 30
nexml.format (nexml.tree), 30
nexml.matrix (nexml.tree), 30
nexml.member (nexml.tree), 30
nexml.meta (meta), 28
nexml.meta(), 32
nexml.meta_ (nexml.tree), 30
nexml.node (nexml.tree), 30
nexml.otu (nexml.tree), 30
nexml.otus (nexml.tree), 30
nexml.polymorphic_states (nexml.tree), 30
nexml.row (nexml.tree), 30
nexml.seq (nexml.tree), 30
nexml.state (nexml.tree), 30
nexml.states (nexml.tree), 30
nexml.tree, 30
nexml.trees (nexml.tree), 30
nexml.uncertain_state (nexml.tree), 30
nexml.uncertain_states (nexml.tree), 30
nexml_add, 32
nexml_figshare, 33
nexml_get, 34
nexml_publish, 35
nexml_read, 36, 38
nexml_to_simmap (simmap_to_nexml), 39
nexml_validate, 37
nexml_write, 28, 37

read.nexml, 12
read.nexml (nexml_read), 36
read.nexml(), 30
reset_id_counter, 39

search order, 4
signature, 5
simmap_to_nexml, 39
slot, ResourceMeta-method, 40
slot<-, ResourceMeta-method
  (slot, ResourceMeta-method), 40
standardGeneric, 3, 14

taxize_nexml, 41
toPhylo, 41

write.nexml, 12
write.nexml (nexml_write), 37

xmlParse, 36