

# Package ‘thredds’

February 14, 2022

**Title** Crawler for Navigating THREDDS Catalogs

**Description** Provides a crawler for programmatically navigating THREDDS Data Server (<<https://www.unidata.ucar.edu/software/thredds/current/tds/TDS.html>>) catalogs, and access dataset metadata and resources.

**Version** 0.1-1

**Date** 2022-02-14

**Depends** R (>= 3.0)

**Imports** R6, rlang, magrittr, httr, xml2

**Suggests** ncdf4, testthat

**Maintainer** Emmanuel Blondel <emmanuel.blondel1@gmail.com>

**URL** <https://github.com/BigelowLab/thredds>,  
<https://www.unidata.ucar.edu/software/thredds/current/tds/TDS.html>

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.1.2

**NeedsCompilation** no

**Author** Ben Tupper [aut],  
Emmanuel Blondel [aut, cre] (<<https://orcid.org/0000-0002-5870-5762>>),  
Bigelow Laboratory for Ocean Sciences [cph]

**Repository** CRAN

**Date/Publication** 2022-02-14 16:20:08 UTC

## R topics documented:

build_xpath . . . . .	2
CatalogNode . . . . .	2
DatasetNode . . . . .	6
get_catalog . . . . .	8
get_xml_ns . . . . .	8
grepl_it . . . . .	9

is_xmlNode . . . . .	9
parse_node . . . . .	10
ServiceNode . . . . .	10
thredds . . . . .	11
ThreddsNode . . . . .	12
xmlString . . . . .	14
xml_children_names . . . . .	14
xml_id . . . . .	15

## Index 16

---

build_xpath	<i>Build and xpath string, possibly using the user specified namespace prefix.</i>
-------------	--

---

### Description

Build and xpath string, possibly using the user specified namespace prefix.

### Usage

```
build_xpath(x, prefix = "d1", select = ".//")
```

### Arguments

x	character one or more path segments
prefix	character by default "d1" prepended to each of the segments in x. If NA or length is 0 then ignore.
select	character, by default search anywhere in the current node with ".//"

### Value

xpath descriptor

---

CatalogNode	<i>A class for Catalogs (which may contain catalogs references or datasets)</i>
-------------	---

---

### Description

A catalog representation that subclasses from ThreddsNode

### Super class

[thredds::ThreddsNode](#) -> CatalogNode

**Methods****Public methods:**

- `CatalogNode$list_services()`
- `CatalogNode$list_catalogs()`
- `CatalogNode$list_datasets()`
- `CatalogNode$get_catalogs()`
- `CatalogNode$get_datasets()`
- `CatalogNode$get_dataset_names()`
- `CatalogNode$get_catalog_names()`
- `CatalogNode$parse_catalog_node()`
- `CatalogNode$parse_dataset_node()`
- `CatalogNode$print()`
- `CatalogNode$clone()`

**Method** `list_services()`: list available services

*Usage:*

```
CatalogNode$list_services(  
  xpath = build_xpath("service", prefix = self$prefix),  
  form = "list"  
)
```

*Arguments:*

`xpath` character, the xpath specifications  
`form` character, either "list" or "table"

*Returns:* list of zero or more character vectors

**Method** `list_catalogs()`: list available catalogRefs

*Usage:*

```
CatalogNode$list_catalogs(  
  xpath = build_xpath(c("dataset", "catalogRef"), prefix = self$prefix),  
  form = "list"  
)
```

*Arguments:*

`xpath` character, the xpath descriptor  
`form` character, either "list" or "table"

*Returns:* a list with zero or more character vectors

**Method** `list_datasets()`: list available datasets

*Usage:*

```
CatalogNode$list_datasets(  
  xpath = build_xpath(c("dataset", "dataset"), prefix = self$prefix),  
  form = "list"  
)
```

*Arguments:*

xpath character, the xpath descriptor  
 form character, either "list" or "table"

*Returns:* a list with zero or more character vectors

**Method** `get_catalogs()`: Retrieve a list one or more of child catalogs

*Usage:*

```
CatalogNode$get_catalogs(
  index,
  xpath = build_xpath(c("dataset", "catalogRef"), prefix = self$prefix)
)
```

*Arguments:*

index integer index (1,...,nChild), indices or name(s)  
 xpath character xpath representation

*Returns:* a list of Catalog class objects, possibly NULL

**Method** `get_datasets()`: Retrieve list one or more dataset children

*Usage:*

```
CatalogNode$get_datasets(
  index,
  xpath = build_xpath(c("dataset", "dataset"), prefix = self$prefix)
)
```

*Arguments:*

index the integer index (1,...,nChild), indices or name(s)  
 xpath character xpath representation

*Returns:* a list of Dataset objects or NULL

**Method** `get_dataset_names()`: Retrieve list zero or more dataset child names. If unnamed, then we substitute "title", "ID", "urlPath", or "href" in that order of availability.

*Usage:*

```
CatalogNode$get_dataset_names(
  xpath = build_xpath(c("dataset", "dataset"), prefix = self$prefix)
)
```

*Arguments:*

xpath character xpath representation  
 index the integer index (1,...,nChild), indices or name(s)

*Returns:* character vector of zero or more names

**Method** `get_catalog_names()`: Retrieve list zero or more catalog child names. If unnamed, then we substitute "title", "ID", "urlPath" or href" in that order of availability.

*Usage:*

```
CatalogNode$get_catalog_names(
  xpath = build_xpath(c("dataset", "catalogRef"), prefix = self$prefix)
)
```

*Arguments:*

xpath character xpath representation  
 index the integer index (1,...,nChild), indices or name(s)

*Returns:* character vector of zero or more names

**Method** parse\_catalog\_node(): Parse a catalog node

*Usage:*

CatalogNode\$parse\_catalog\_node(x)

*Arguments:*

x xml\_node

*Returns:* Catalog class object

**Method** parse\_dataset\_node(): Parse a dataset node

*Usage:*

CatalogNode\$parse\_dataset\_node(x)

*Arguments:*

x xml\_node

*Returns:* Dataset class object

**Method** print(): print method

*Usage:*

CatalogNode\$print(prefix = "")

*Arguments:*

prefix character, to be printed before each line of output (like spaces)  
 ... other arguments for superclass

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

CatalogNode\$clone(deep = FALSE)

*Arguments:*

deep Whether to make a deep clone.

**Examples**

```
library(thredds)
top_uri <- 'https://oceandata.sci.gsfc.nasa.gov/opendap/catalog.xml'
Top <- thredds::CatalogNode$new(top_uri)
Top$browse()

#go down in 'MODISA' catalog
L3 <- Top$get_catalogs("MODISA")[["MODISA"]$get_catalogs()[[1]]

#see what's available for 2009
```

```

catalog2009 <- L3$get_catalogs("2009")[[1]]

#get catalog for 2009-01-20
doy <- format(as.Date("2009-01-20"), "%j")
catalog20 <- catalog2009$get_catalogs(doy)[[doy]]

#get dataset node
chl <- catalog20$get_datasets("A2009020.L3m_DAY_CHL_chlor_a_9km.nc")[[1]]

#retrieve the relative URL, and add it to the base URL for the service.
#Somewhat awkwardly, the relative URL comes prepended with a path separator, so we
#use straight up `paste0` to append to the base_uri.
#if(require("ncdf4")){
#  base_uri <- "https://oceandata.sci.gsfc.nasa.gov:443/opendap"
#  uri <- paste0(base_uri, chl$url)
#  NC <- ncdf4::nc_open(uri)
#}

```

---

DatasetNode

*A class for a single dataset reference*


---

## Description

A direct Dataset representation that subclasses from ThreddsNode

## Super class

[thredds::ThreddsNode](#) -> DatasetNode

## Public fields

name character, often the filename

dataSize numeric, size in bytes

date character, modification date

## Methods

### Public methods:

- [DatasetNode\\$new\(\)](#)
- [DatasetNode\\$GET\(\)](#)
- [DatasetNode\\$get\\_url\(\)](#)
- [DatasetNode\\$list\\_access\(\)](#)
- [DatasetNode\\$print\(\)](#)
- [DatasetNode\\$clone\(\)](#)

**Method** [new\(\)](#): initialize an instance of ServiceNode

*Usage:*

```
DatasetNode$new(x, ...)
```

*Arguments:*

```
x url or xml2::xml_node
```

```
... arguments for superclass initialization
```

**Method** `GET()`: Overrides the GET method of the superclass. GET is not permitted

*Usage:*

```
DatasetNode$GET()
```

*Returns:* NULL

**Method** `get_url()`: Retrieve the relative URL for a dataset.

*Usage:*

```
DatasetNode$get_url(
  service = c("dap", "opendap", "wms")[1],
  sep = c("/" , "")[2],
  ...
)
```

*Arguments:*

`service` character, the service to use. (default 'dap' equivalent to 'opendap') Ignored if 'url-Path' or 'href' is in the nodes' attributes.

`sep` character, typically "/" or "" (default), used for joined base\_url to relative url

... other arguments for DatasetNode\$list\_access

*Returns:* character

**Method** `list_access()`: list access methods

*Usage:*

```
DatasetNode$list_access(xpath = build_xpath("access", prefix = self$prefix))
```

*Arguments:*

`xpath` character, xpath descriptor

*Returns:* named list of character vectors or NULL

**Method** `print()`: print method

*Usage:*

```
DatasetNode$print(prefix = "")
```

*Arguments:*

`prefix` character, to be printed before each line of output (like spaces)

... other arguments for superclass

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
DatasetNode$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

**Note**

For examples see [CatalogNode](#)

---

get_catalog	<i>Retrieve a catalog</i>
-------------	---------------------------

---

**Description**

Retrieve a catalog

**Usage**

```
get_catalog(uri, ...)
```

**Arguments**

uri	the URI of the catalog
...	further arguments for parse_node

**Value**

ThreddsNodeRefClass or subclass or NULL

---

get_xml_ns	<i>Retrieve the namespaces for a resource</i>
------------	---

---

**Description**

Retrieve the namespaces for a resource

**Usage**

```
get_xml_ns(uri)
```

**Arguments**

uri	the URI of the catalog
-----	------------------------

**Value**

the output of [xml\\_ns](#)



---

grepl_it	<i>Determine if a vector of names match the greplargs</i>
----------	---

---

**Description**

Determine if a vector of names match the greplargs

**Usage**

```
grepl_it(x, greplargs = NULL)
```

**Arguments**

x	a vector of names
greplargs	NULL, vector or list

**Value**

logical vector

---

is_xmlNode	<i>Test if an object inherits from xml2::xml_node</i>
------------	---

---

**Description**

Test if an object inherits from xml2::xml\_node

**Usage**

```
is_xmlNode(x, classname = "xml_node")
```

**Arguments**

x	object to test
classname	character, the class name to test against, by default 'xml_node'

**Value**

logical

---

parse_node	<i>Convert a node to an object inheriting from ThreddsNode</i>
------------	--

---

**Description**

Convert a node to an object inheriting from ThreddsNode

**Usage**

```
parse_node(node, url = NULL, verbose = FALSE, encoding = "UTF-8", ...)
```

**Arguments**

node	xml2::xml_node or an httr::response object
url	character, optional url if a catalog or direct dataset
verbose	logical, by default FALSE
encoding	character, by default UTF-8
...	further arguments for instantiation of classes (such as ns = "foo")

**Value**

ThreddsNode class object or subclass

---

ServiceNode	<i>A simple class for parsing and holdoing service info</i>
-------------	---

---

**Description**

A Service representation that subclasses from ThreddsNode

**Super class**

[thredds::ThreddsNode](#) -> ServiceNode

**Public fields**

name	character
serviceType	character
base	character base url

## Methods

### Public methods:

- [ServiceNode\\$new\(\)](#)
- [ServiceNode#print\(\)](#)
- [ServiceNode\\$clone\(\)](#)

**Method** `new()`: initialize an instance of `ServiceNode`

*Usage:*

```
ServiceNode$new(x, ...)
```

*Arguments:*

x url or `xml2::xml_node`

... arguments for superclass initialization

**Method** `print()`: print method

*Usage:*

```
ServiceNode#print(prefix = "")
```

*Arguments:*

prefix character, to be printed before each line of output (like spaces)

... other arguments for superclass

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
ServiceNode$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

## Note

For examples see [CatalogNode](#)

## Description

A limited crawler for programmatically navigating THREDDDS catalogs.

---

 ThreddsNode

*A base representation that other nodes subclass from*


---

### Description

R6 base class for all other to inherit from

### Public fields

url character - possibly wrong but usually right!

node xml2::xml\_node

verbose logical

prefix xpath namespace prefix, NA or NULL or charcater() to ignore

tries numeric number of requests attempts before failing

encoding character, by default 'UTF-8'

base\_url character, the base URL for the service

### Methods

#### Public methods:

- [ThreddsNode\\$new\(\)](#)
- [ThreddsNode\\$print\(\)](#)
- [ThreddsNode\\$GET\(\)](#)
- [ThreddsNode\\$browse\(\)](#)
- [ThreddsNode\\$children\\_names\(\)](#)
- [ThreddsNode\\$clone\(\)](#)

**Method** `new()`: initialize an instance of ThreddsNode

*Usage:*

```
ThreddsNode$new(
  x,
  verbose = FALSE,
  n_tries = 3,
  prefix = NULL,
  ns_strip = FALSE,
  encoding = "UTF-8",
  base_url = ""
)
```

*Arguments:*

x url or xml2::xml\_node

verbose logical, TRUE to be noisy (default FALSE)

n\_tries numeric, defaults to 3

prefix character, the namespace to examine (default NULL, inherited when initialized)

ns\_strip logical, if TRUE then strip namespace (default FALSE)  
encoding character, by default 'UTF-8'  
base\_url character, the base URL for the service

**Method** print(): print method

*Usage:*

ThreddsNode\$print(prefix = "", ...)

*Arguments:*

prefix character, to be printed before each line of output (like spaces)  
... other arguments (ignored for now)

**Method** GET(): Retrieve a node of the contents at this nodes URL

*Usage:*

ThreddsNode\$GET()

*Returns:* ThreddsNode or subclass or NULL

**Method** browse(): Browse the URL if possible

*Usage:*

ThreddsNode\$browse()

**Method** children\_names(): Retrieve a vector of unique child names

*Usage:*

ThreddsNode\$children\_names(...)

*Arguments:*

... further arguments for [xml\\_children\\_names](#)

*Returns:* a vector of zero or more child names

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

ThreddsNode\$clone(deep = FALSE)

*Arguments:*

deep Whether to make a deep clone.

#### Note

Abstract class. For examples see [CatalogNode](#)

xmlString                      *Convert xml2::xml\_node to character*

---

**Description**

Convert xml2::xml\_node to character

**Usage**

```
xmlString(x)
```

**Arguments**

x                      xmlNode

**Value**

character

---

xml\_children\_names            *Get the names of children*

---

**Description**

Get the names of children

**Usage**

```
xml_children_names(x, unique_only = TRUE)
```

**Arguments**

x                      xml2::xml\_node  
unique\_only            logical if TRUE remove duplicates

**Value**

zero or more child names.

---

`xml_id`*Retrieve an ID value for a node from it's attributes.*

---

**Description**

Retrieve an ID value for a node from it's attributes.

**Usage**

```
xml_id(x, atts = c("name", "title", "ID", "urlPath", "href"))
```

**Arguments**

<code>x</code>	xml node or a named character vector as per <code>xml_attrs</code>
<code>atts</code>	character, ordered vector of attribute names to use as an ID value As the list is stepped through if an attribute is missing or empty character then advance to the next, otherwise return that value

**Value**

character identifier, possibly an empty character (`character()`)

# Index

`build_xpath`, 2

`CatalogNode`, 2, 8, 11, 13

`DatasetNode`, 6

`get_catalog`, 8

`get_xml_ns`, 8

`grepl_it`, 9

`is_xmlNode`, 9

`parse_node`, 10

`ServiceNode`, 10

`thredds`, 11

`thredds::ThreddsNode`, 2, 6, 10

`ThreddsNode`, 12

`xml_children_names`, 13, 14

`xml_id`, 15

`xml_ns`, 8

`xmlString`, 14