Package 'xml2'

April 23, 2020

Title Parse XML

Version 1.3.2

Description Work with XML files using a simple, consistent interface. Built on top of the 'libxml2' C library.

License GPL (>=2)

URL https://xml2.r-lib.org/, https://github.com/r-lib/xml2

BugReports https://github.com/r-lib/xml2/issues

Depends R (>= 3.1.0)

Imports methods

Suggests covr,
curl,
httr,
knitr,
magrittr,
mockery,
rmarkdown,
testthat (>= 2.1.0)

VignetteBuilder knitr

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.0

SystemRequirements libxml2: libxml2-dev (deb), libxml2-devel (rpm)

Collate 'S4.R'
'as_list.R'
'xml_parse.R'
'as_xml_document.R'
'classes.R'
'init.R'
'paths.R'
'utils.R'
'xml_attr.R'
'xml_children.R'
'xml_find.R'
'xml_modify.R'

'xml_name.R'
'xml_namespaces.R'
'xml_path.R'
'xml_schema.R'
'xml_serialize.R'
'xml_structure.R'
'xml_text.R'
'xml_type.R'
'xml_url.R'
'xml_write.R'
'zzz.R'

R topics documented:

`as_list` *Coerce xml nodes to a list.*

Description

This turns an XML document (or node or nodeset) into the equivalent R list. Note that this is `as_list()`, not `as.list()`: `lapply()` automatically calls `as.list()` on its inputs, so we can't override the default.

Usage

```
as_list(x, ns = character(), ...)
```

Arguments

`x`	A document, node, or node set.
`ns`	Optionally, a named vector giving prefix-url pairs, as produced by `xml_ns()`. If provided, all names will be explicitly qualified with the ns prefix, i.e. if the element `bar` is defined in namespace `foo`, it will be called `foo:bar`. (And similarly for attributes). Default namespaces must be given an explicit name. The ns is ignored when using `xml_name<-()` and `xml_set_name()`.
`...`	Needed for compatibility with generic. Unused.

Details

`as_list` currently only handles the four most common types of children that an element might have:

- Other elements, converted to lists.
- Attributes, stored as R attributes. Attributes that have special meanings in R (`class()`, `comment()`, `dim()`, `dimnames()`, `names()`, `row.names()` and `tsp()`) are escaped with '.'
- Text, stored as a character vector.

Examples

```
as_list(read_xml("<foo> a <b /><c><![CDATA[<d></d>]]></c></foo>"))
as_list(read_xml("<foo> <bar><baz /></bar> </foo>"))
as_list(read_xml("<foo id = 'a'></foo>"))
as_list(read_xml("<foo><bar id='a'/><bar id='b'/></foo>"))
```

as_xml_document *Coerce a R list to xml nodes.*

Description

This turns an R list into the equivalent XML document. Not all R lists will produce valid XML, in particular there can only be one root node and all child nodes need to be named (or empty) lists. R attributes become XML attributes and R names become XML node names.

Usage

```
as_xml_document(x, ...)
```

Arguments

x	A document, node, or node set.
...	Needed for compatibility with generic. Unused.

Examples

```
as_xml_document(list(x = list()))

# Nesting multiple nodes
as_xml_document(list(foo = list(bar = list(baz = list()))))

# attributes are stored as R attributes
as_xml_document(list(foo = structure(list(), id = "a")))
as_xml_document(list(foo = list(
     bar = structure(list(), id = "a"),
     bar = structure(list(), id = "b"))))
```

download_xml *Download a HTML or XML file*

Description

Libcurl implementation of `C_download` (the "internal" download method) with added support for https, ftps, gzip, etc. Default behavior is identical to `download.file()`, but request can be fully configured by passing a custom `curl::handle()`.

Usage

```
download_xml(
  url,
  file = basename(url),
  quiet = TRUE,
  mode = "wb",
  handle = curl::new_handle()
)
```

```
download_html(
  url,
  file = basename(url),
  quiet = TRUE,
  mode = "wb",
  handle = curl::new_handle()
)
```

Arguments

url	A character string naming the URL of a resource to be downloaded.
file	A character string with the name where the downloaded file is saved.
quiet	If TRUE, suppress status messages (if any), and the progress bar.
mode	A character string specifying the mode with which to write the file. Useful values are "w", "wb" (binary), "a" (append) and "ab".
handle	a curl handle object

Details

The main difference between curl_download and curl_fetch_disk is that curl_download checks the http status code before starting the download, and raises an error when status is non-successful. The behavior of curl_fetch_disk on the other hand is to proceed as normal and write the error page to disk in case of a non success response.

Value

Path of downloaded file (invisibly).

See Also

curl_download

Examples

```
## Not run:
download_html("http://tidyverse.org/index.html")

## End(Not run)
```

---

read_xml *Read HTML or XML.*

Description

Read HTML or XML.

Usage

```
read_xml(x, encoding = "", ..., as_html = FALSE, options = "NOBLANKS")

read_html(x, encoding = "", ..., options = c("RECOVER", "NOERROR", "NOBLANKS"))

## S3 method for class 'character'
read_xml(x, encoding = "", ..., as_html = FALSE, options = "NOBLANKS")

## S3 method for class 'raw'
read_xml(
  x,
  encoding = "",
  base_url = "",
  ...,
  as_html = FALSE,
  options = "NOBLANKS"
)

## S3 method for class 'connection'
read_xml(
  x,
  encoding = "",
  n = 64 * 1024,
  verbose = FALSE,
  ...,
  base_url = "",
  as_html = FALSE,
  options = "NOBLANKS"
)
```

Arguments

x	A string, a connection, or a raw vector. A string can be either a path, a url or literal xml. Urls will be converted into connections either using `base::url` or, if installed, `curl::curl`. Local paths ending in `.gz`, `.bz2`, `.xz`, `.zip` will be automatically uncompressed. If a connection, the complete connection is read into a raw vector before being parsed.
encoding	Specify a default encoding for the document. Unless otherwise specified XML documents are assumed to be in UTF-8 or UTF-16. If the document is not UTF-8/16, and lacks an explicit encoding directive, this allows you to supply a default.
...	Additional arguments passed on to methods.
as_html	Optionally parse an xml file as if it's html.
options	Set parsing options for the libxml2 parser. Zero or more of **RECOVER** recover on errors **NOENT** substitute entities **DTDLOAD** load the external subset **DTDATTR** default DTD attributes **DTDVALID** validate with the DTD

NOERROR suppress error reports

NOWARNING suppress warning reports

PEDANTIC pedantic error reporting

NOBLANKS remove blank nodes

SAX1 use the SAX1 interface internally

XINCLUDE Implement XInclude substitition

NONET Forbid network access

NODICT Do not reuse the context dictionary

NSCLEAN remove redundant namespaces declarations

NOCDATA merge CDATA as text nodes

NOXINCNODE do not generate XINCLUDE START/END nodes

COMPACT compact small text nodes; no modification of the tree allowed afterwards (will possibly crash if you try to modify the tree)

OLD10 parse using XML-1.0 before update 5

NOBASEFIX do not fixup XINCLUDE xml:base uris

HUGE relax any hardcoded limit from the parser

OLDSAX parse using SAX2 interface before 2.7.0

IGNORE_ENC ignore internal document encoding hint

BIG_LINES Store big lines numbers in text PSVI field

`base_url` When loading from a connection, raw vector or literal html/xml, this allows you to specify a base url for the document. Base urls are used to turn relative urls into absolute urls.

`n` If `file` is a connection, the number of bytes to read per iteration. Defaults to 64kb.

`verbose` When reading from a slow connection, this prints some output on every iteration so you know its working.

Value

An XML document. HTML is normalised to valid XML - this may not be exactly the same transformation performed by the browser, but it's a reasonable approximation.

Setting the "user agent" header

When performing web scraping tasks it is both good practice — and often required — to set the user agent request header to a specific value. Sometimes this value is assigned to emulate a browser in order to have content render in a certain way (e.g. Mozilla/5.0 (Windows NT 5.1; rv:52.0) Gecko/20100101 Firefox/52.0 to emulate more recent Windows browsers). Most often, this value should be set to provide the web resource owner information on who you are and the intent of your actions like this Google scraping bot user agent identifier: Googlebot/2.1 (+http://www.google.com/bot.html).

You can set the HTTP user agent for URL-based requests using `httr::set_config()` and `httr::user_agent()`:

```
httr::set_config(httr::user_agent("me@example.com; +https://example.com/info.html"))
```

`httr::set_config()` changes the configuration globally, `httr::with_config()` can be used to change configuration temporarily.

Examples

```
# Literal xml/html is useful for small examples
read_xml("<foo><bar /></foo>")
read_html("<html><title>Hi<title></html>")
read_html("<html><title>Hi")

# From a local path
read_html(system.file("extdata", "r-project.html", package = "xml2"))

## Not run:
# From a url
cd <- read_xml(xml2_example("cd_catalog.xml"))
me <- read_html("http://had.co.nz")

## End(Not run)
```

`url_absolute` *Convert between relative and absolute urls.*

Description

Convert between relative and absolute urls.

Usage

```
url_absolute(x, base)

url_relative(x, base)
```

Arguments

`x`	A character vector of urls relative to that base
`base`	A string giving a base url.

Value

A character vector of urls

See Also

`xml_url` to retrieve the URL associated with a document

Examples

```
url_absolute(c(".", "..", "/", "/x"), "http://hadley.nz/a/b/c/d")

url_relative("http://hadley.nz/a/c", "http://hadley.nz")
url_relative("http://hadley.nz/a/c", "http://hadley.nz/")
url_relative("http://hadley.nz/a/c", "http://hadley.nz/a/b")
url_relative("http://hadley.nz/a/c", "http://hadley.nz/a/b/")
```

url_escape *Escape and unescape urls.*

Description

Escape and unescape urls.

Usage

```
url_escape(x, reserved = "")

url_unescape(x)
```

Arguments

x	A character vector of urls.
reserved	A string containing additional characters to avoid escaping.

Examples

```
url_escape("a b c")
url_escape("a b c", "")

url_unescape("a%20b%2fc")
url_unescape("%C2%B5")
```

url_parse *Parse a url into its component pieces.*

Description

Parse a url into its component pieces.

Usage

```
url_parse(x)
```

Arguments

x	A character vector of urls.

Value

A dataframe with one row for each element of x and columns: scheme, server, port, user, path, query, fragment.

Examples

```
url_parse("http://had.co.nz/")
url_parse("http://had.co.nz:1234/")
url_parse("http://had.co.nz:1234/?a=1&b=2")
url_parse("http://had.co.nz:1234/?a=1&b=2#def")
```

write_xml *Write XML or HTML to disk.*

Description

This writes out both XML and normalised HTML. The default behavior will output the same format which was read. If you want to force output pass `option = "as_xml"` or `option = "as_html"` respectively.

Usage

```
write_xml(x, file, ...)

## S3 method for class 'xml_document'
write_xml(x, file, ..., options = "format", encoding = "UTF-8")

write_html(x, file, ...)

## S3 method for class 'xml_document'
write_html(x, file, ..., options = "format", encoding = "UTF-8")
```

Arguments

x	A document or node to write to disk. It's not possible to save nodesets containing more than one node.
file	Path to file or connection to write to.
...	additional arguments passed to methods.
options	default: 'format'. Zero or more of **format** Format output **no_declaration** Drop the XML declaration **no_empty_tags** Remove empty tags **no_xhtml** Disable XHTML1 rules **require_xhtml** Force XHTML rules **as_xml** Force XML output **as_html** Force HTML output **format_whitespace** Format with non-significant whitespace
encoding	The character encoding to use in the document. The default encoding is 'UTF-8'. Available encodings are specified at http://xmlsoft.org/html/libxml-encoding.html#xmlCharEncoding.

Examples

```
h <- read_html("<p>Hi!</p>")

tmp <- tempfile(fileext = ".xml")
write_xml(h, tmp, options = "format")
readLines(tmp)

# write formatted HTML output
write_html(h, tmp, options = "format")
readLines(tmp)
```

xml2_example *Get path to a xml2 example*

Description

xml2 comes bundled with a number of sample files in its 'inst/extdata' directory. This function makes them easy to access.

Usage

```
xml2_example(path = NULL)
```

Arguments

path	Name of file. If NULL, the example files will be listed.

xml_attr *Retrieve an attribute.*

Description

xml_attrs() retrieves all attributes values as a named character vector, xml_attrs() <- or xml_set_attrs() sets all attribute values. xml_attr() retrieves the value of single attribute and xml_attr() <- or xml_set_attr() modifies its value. If the attribute doesn't exist, it will return default, which defaults to NA. xml_has_attr() tests if an attribute is present.

Usage

```
xml_attr(x, attr, ns = character(), default = NA_character_)

xml_has_attr(x, attr, ns = character())

xml_attrs(x, ns = character())

xml_attr(x, attr, ns = character()) <- value

xml_set_attr(x, attr, value, ns = character())

xml_attrs(x, ns = character()) <- value

xml_set_attrs(x, value, ns = character())
```

Arguments

x	A document, node, or node set.
attr	Name of attribute to extract.

ns	Optionally, a named vector giving prefix-url pairs, as produced by `xml_ns()`. If provided, all names will be explicitly qualified with the ns prefix, i.e. if the element `bar` is defined in namespace `foo`, it will be called `foo:bar`. (And similarly for attributes). Default namespaces must be given an explicit name. The ns is ignored when using `xml_name<-()` and `xml_set_name()`.
default	Default value to use when attribute is not present.
value	character vector of new value.

Value

`xml_attr()` returns a character vector. `NA` is used to represent of attributes that aren't defined.

`xml_has_attr()` returns a logical vector.

`xml_attrs()` returns a named character vector if `x` x is single node, or a list of character vectors if given a nodeset

Examples

```
x <- read_xml("<root id='1'><child id ='a' /><child id='b' d='b'/></root>")
xml_attr(x, "id")
xml_attr(x, "apple")
xml_attrs(x)

kids <- xml_children(x)
kids
xml_attr(kids, "id")
xml_has_attr(kids, "id")
xml_attrs(kids)

# Missing attributes give missing values
xml_attr(xml_children(x), "d")
xml_has_attr(xml_children(x), "d")

# If the document has a namespace, use the ns argument and
# qualified attribute names
x <- read_xml('
 <root xmlns:b="http://bar.com" xmlns:f="http://foo.com">
   <doc b:id="b" f:id="f" id="" />
 </root>
')
doc <- xml_children(x)[[1]]
ns <- xml_ns(x)

xml_attrs(doc)
xml_attrs(doc, ns)

# If you don't supply a ns spec, you get the first matching attribute
xml_attr(doc, "id")
xml_attr(doc, "b:id", ns)
xml_attr(doc, "id", ns)

# Can set a single attribute with `xml_attr() <-` or `xml_set_attr()`
xml_attr(doc, "id") <- "one"
xml_set_attr(doc, "id", "two")

# Or set multiple attributes with `xml_attrs()` or `xml_set_attrs()`
```

```
xml_attrs(doc) <- c("b:id" = "one", "f:id" = "two", "id" = "three")
xml_set_attrs(doc, c("b:id" = "one", "f:id" = "two", "id" = "three"))
```

`xml_cdata` *Construct a cdata node*

Description

Construct a cdata node

Usage

```
xml_cdata(content)
```

Arguments

`content`	The CDATA content, does not include <![CDATA[

Examples

```
x <- xml_new_root("root")
xml_add_child(x, xml_cdata("<d/>"))
as.character(x)
```

`xml_children` *Navigate around the family tree.*

Description

`xml_children` returns only elements, `xml_contents` returns all nodes. `xml_length` returns the number of children. `xml_parent` returns the parent node, `xml_parents` returns all parents up to the root. `xml_siblings` returns all nodes at the same level. `xml_child` makes it easy to specify a specific child to return.

Usage

```
xml_children(x)

xml_child(x, search = 1, ns = xml_ns(x))

xml_contents(x)

xml_parents(x)

xml_siblings(x)

xml_parent(x)

xml_length(x, only_elements = TRUE)

xml_root(x)
```

Arguments

x	A document, node, or node set.
search	For xml_child, either the child number to return (by position), or the name of the child node to return. If there are multiple child nodes with the same name, the first will be returned
ns	Optionally, a named vector giving prefix-url pairs, as produced by xml_ns(). If provided, all names will be explicitly qualified with the ns prefix, i.e. if the element bar is defined in namespace foo, it will be called foo:bar. (And similarly for attributes). Default namespaces must be given an explicit name. The ns is ignored when using xml_name<-() and xml_set_name().
only_elements	For xml_length, should it count all children, or just children that are elements (the default)?

Value

A node or nodeset (possibly empty). Results are always de-duplicated.

Examples

```
x <- read_xml("<foo> <bar><boo /></bar> <baz/> </foo>")
xml_children(x)
xml_children(xml_children(x))
xml_siblings(xml_children(x)[[1]])

# Note the each unique node only appears once in the output
xml_parent(xml_children(x))

# Mixed content
x <- read_xml("<foo> a <b/> c <d>e</d> f</foo>")
# Childen gets the elements, contents gets all node types
xml_children(x)
xml_contents(x)

xml_length(x)
xml_length(x, only_elements = FALSE)

# xml_child makes it easier to select specific children
xml_child(x)
xml_child(x, 2)
xml_child(x, "baz")
```

---

xml_comment *Construct a comment node*

Description

Construct a comment node

Usage

```
xml_comment(content)
```

Arguments

content	The comment content

Examples

```
x <- xml_new_document()
r <- xml_add_child(x, "root")
xml_add_child(r, xml_comment("Hello!"))
as.character(x)
```

---

xml_document-class *Register S4 classes*

Description

Classes are exported so they can be re-used within S4 classes, see `methods::setOldClass()`.

`xml_document`: a complete document.

`xml_missing`: a missing object, e.g. for an empty result set.

`xml_node`: a single node in a document.

`xml_nodeset`: a *set* of nodes within a document.

---

xml_dtd *Construct a document type definition*

Description

This is used to create simple document type definitions. If you need to create a more complicated definition with internal subsets it is recommended to parse a string directly with `read_xml()`.

Usage

```
xml_dtd(name = "", external_id = "", system_id = "")
```

Arguments

name	The name of the declaration
external_id	The external ID of the declaration
system_id	The system ID of the declaration

Examples

```
r <- xml_new_root(
  xml_dtd("html",
    "-//W3C//DTD XHTML 1.0 Transitional//EN",
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"))

# Use read_xml directly for more complicated DTD
d <- read_xml(
'<!DOCTYPE doc [
<!ELEMENT doc (#PCDATA)>
<!ENTITY foo " test ">
]>
<doc>This is a valid document &foo; !</doc>')
```

---

xml_find_all *Find nodes that match an xpath expression.*

Description

Xpath is like regular expressions for trees - it's worth learning if you're trying to extract nodes from arbitrary locations in a document. Use `xml_find_all` to find all matches - if there's no match you'll get an empty result. Use `xml_find_first` to find a specific match - if there's no match you'll get an `xml_missing` node.

Usage

```
xml_find_all(x, xpath, ns = xml_ns(x))

xml_find_first(x, xpath, ns = xml_ns(x))

xml_find_num(x, xpath, ns = xml_ns(x))

xml_find_chr(x, xpath, ns = xml_ns(x))

xml_find_lgl(x, xpath, ns = xml_ns(x))
```

Arguments

x	A document, node, or node set.
xpath	A string containing a xpath (1.0) expression.
ns	Optionally, a named vector giving prefix-url pairs, as produced by `xml_ns()`. If provided, all names will be explicitly qualified with the ns prefix, i.e. if the element `bar` is defined in namespace `foo`, it will be called `foo:bar`. (And similarly for attributes). Default namespaces must be given an explicit name. The ns is ignored when using `xml_name<-()` and `xml_set_name()`.

Value

`xml_find_all` always returns a nodeset: if there are no matches the nodeset will be empty. The result will always be unique; repeated nodes are automatically de-duplicated.

`xml_find_first` returns a node if applied to a node, and a nodeset if applied to a nodeset. The output is *always* the same size as the input. If there are no matches, `xml_find_first` will return a missing node; if there are multiple matches, it will return the first only.

`xml_find_num`, `xml_find_chr`, `xml_find_lgl` return numeric, character and logical results respectively.

Deprecated functions

`xml_find_one()` has been deprecated. Instead use `xml_find_first()`.

See Also

`xml_ns_strip()` to remove the default namespaces

Examples

```
x <- read_xml("<foo><bar><baz/></bar><baz/></foo>")
xml_find_all(x, ".//baz")
xml_path(xml_find_all(x, ".//baz"))

# Note the difference between .// and //
# //  finds anywhere in the document (ignoring the current node)
# .// finds anywhere beneath the current node
(bar <- xml_find_all(x, ".//bar"))
xml_find_all(bar, ".//baz")
xml_find_all(bar, "//baz")

# Find all vs find one ----------------------------------------------------
x <- read_xml("<body>
  <p>Some <b>text</b>.</p>
  <p>Some <b>other</b> <b>text</b>.</p>
  <p>No bold here!</p>
</body>")
para <- xml_find_all(x, ".//p")

# If you apply xml_find_all to a nodeset, it finds all matches,
# de-duplicates them, and returns as a single list. This means you
# never know how many results you'll get
xml_find_all(para, ".//b")

# xml_find_first only returns the first match per input node. If there are 0
# matches it will return a missing node
xml_find_first(para, ".//b")
xml_text(xml_find_first(para, ".//b"))

# Namespaces --------------------------------------------------------------
# If the document uses namespaces, you'll need use xml_ns to form
# a unique mapping between full namespace url and a short prefix
x <- read_xml('
 <root xmlns:f = "http://foo.com" xmlns:g = "http://bar.com">
   <f:doc><g:baz /></f:doc>
   <f:doc><g:baz /></f:doc>
 </root>
')
xml_find_all(x, ".//f:doc")
xml_find_all(x, ".//f:doc", xml_ns(x))
```

xml_name *The (tag) name of an xml element.*

Description

The (tag) name of an xml element.

Modify the (tag) name of an element

Usage

```
xml_name(x, ns = character())

xml_name(x, ns = character()) <- value

xml_set_name(x, value, ns = character())
```

Arguments

x	A document, node, or node set.
ns	Optionally, a named vector giving prefix-url pairs, as produced by xml_ns(). If provided, all names will be explicitly qualified with the ns prefix, i.e. if the element bar is defined in namespace foo, it will be called foo:bar. (And similarly for attributes). Default namespaces must be given an explicit name. The ns is ignored when using xml_name<-() and xml_set_name().
value	a character vector with replacement name.

Value

A character vector.

Examples

```
x <- read_xml("<bar>123</bar>")
xml_name(x)

y <- read_xml("<bar><baz>1</baz>abc<foo /></bar>")
z <- xml_children(y)
xml_name(xml_children(y))
```

xml_new_document *Create a new document, possibly with a root node*

Description

xml_new_document creates only a new document without a root node. In most cases you should instead use xml_new_root, which creates a new document and assigns the root node in one step.

Usage

```
xml_new_document(version = "1.0", encoding = "UTF-8")

xml_new_root(
  .value,
  ...,
  .copy = inherits(.value, "xml_node"),
  .version = "1.0",
  .encoding = "UTF-8"
)
```

Arguments

version	The version number of the document.
encoding	The character encoding to use in the document. The default encoding is 'UTF-8'. Available encodings are specified at http://xmlsoft.org/html/libxml-encoding.html#xmlCharEncoding.
.value	node to insert.
...	If named attributes or namespaces to set on the node, if unnamed text to assign to the node.
.copy	whether to copy the .value before replacing. If this is FALSE then the node will be moved from it's current location.
.version	The version number of the document, passed to xml_new_document(version).
.encoding	The encoding of the document, passed to xml_new_document(encoding).

Value

A xml_document object.

---

xml_ns *XML namespaces.*

Description

xml_ns extracts all namespaces from a document, matching each unique namespace url with the prefix it was first associated with. Default namespaces are named d1, d2 etc. Use xml_ns_rename to change the prefixes. Once you have a namespace object, you can pass it to other functions to work with fully qualified names instead of local names.

Usage

```
xml_ns(x)

xml_ns_rename(old, ...)
```

Arguments

x	A document, node, or node set.
old, ...	An existing xml_namespace object followed by name-value (old prefix-new prefix) pairs to replace.

Value

A character vector with class `xml_namespace` so the default display is a little nicer.

Examples

```
x <- read_xml('
 <root>
   <doc1 xmlns = "http://foo.com"><baz /></doc1>
   <doc2 xmlns = "http://bar.com"><baz /></doc2>
 </root>
')
xml_ns(x)

# When there are default namespaces, it's a good idea to rename
# them to give informative names:
ns <- xml_ns_rename(xml_ns(x), d1 = "foo", d2 = "bar")
ns

# Now we can pass ns to other xml function to use fully qualified names
baz <- xml_children(xml_children(x))
xml_name(baz)
xml_name(baz, ns)

xml_find_all(x, "//baz")
xml_find_all(x, "//foo:baz", ns)

str(as_list(x))
str(as_list(x, ns))
```

---

xml_ns_strip *Strip the default namespaces from a document*

Description

Strip the default namespaces from a document

Usage

```
xml_ns_strip(x)
```

Arguments

x	A document, node, or node set.

Examples

```
x <- read_xml(
 "<foo xmlns = 'http://foo.com'>
   <baz/>
   <bar xmlns = 'http://bar.com'>
     <baz/>
   </bar>
  </foo>")
# Need to specify the default namespaces to find the baz nodes
```

```
xml_find_all(x, "//d1:baz")
xml_find_all(x, "//d2:baz")

# After stripping the default namespaces you can find both baz nodes directly
xml_ns_strip(x)
xml_find_all(x, "//baz")
```

`xml_path` *Retrieve the xpath to a node*

Description

This is useful when you want to figure out where nodes matching an xpath expression live in a document.

Usage

```
xml_path(x)
```

Arguments

`x`	A document, node, or node set.

Value

A character vector.

Examples

```
x <- read_xml("<foo><bar><baz /></bar><baz /></foo>")
xml_path(xml_find_all(x, ".//baz"))
```

`xml_replace` *Modify a tree by inserting, replacing or removing nodes*

Description

`xml_add_sibling()` and `xml_add_child()` are used to insert a node as a sibling or a child. `xml_add_parent()` adds a new parent in between the input node and the current parent. `xml_replace()` replaces an existing node with a new node. `xml_remove()` removes a node from the tree.

Usage

```
xml_replace(.x, .value, ..., .copy = TRUE)

xml_add_sibling(.x, .value, ..., .where = c("after", "before"), .copy = TRUE)

xml_add_child(.x, .value, ..., .where = length(xml_children(.x)), .copy = TRUE)

xml_add_parent(.x, .value, ...)

xml_remove(.x, free = FALSE)
```

Arguments

.x	a document, node or nodeset.
.value	node to insert.
...	If named attributes or namespaces to set on the node, if unnamed text to assign to the node.
.copy	whether to copy the .value before replacing. If this is FALSE then the node will be moved from it's current location.
.where	to add the new node, for xml_add_child the position after which to add, use 0 for the first child. For xml_add_sibling either '"before"' or '"after"' indicating if the new node should be before or after .x.
free	When removing the node also free the memory used for that node. Note if you use this option you cannot use any existing objects pointing to the node or its children, it is likely to crash R or return garbage.

Details

Care needs to be taken when using xml_remove(),

---

xml_serialize *Serializing XML objects to connections.*

Description

Serializing XML objects to connections.

Usage

```
xml_serialize(object, connection, ...)

xml_unserialize(connection, ...)
```

Arguments

object	R object to serialize.
connection	an open connection or (for serialize) NULL or (for unserialize) a raw vector (see 'Details').
...	Additional arguments passed to read_xml().

Value

For serialize, NULL unless connection = NULL, when the result is returned in a raw vector.

For unserialize an R object.

Examples

```
library(xml2)
x <- read_xml("<a>
  <b><c>123</c></b>
  <b><c>456</c></b>
</a>")

b <- xml_find_all(x, "//b")
out <- xml_serialize(b, NULL)
xml_unserialize(out)
```

`xml_set_namespace` *Set the node's namespace*

Description

The namespace to be set must be already defined in one of the node's ancestors.

Usage

```
xml_set_namespace(.x, prefix = "", uri = "")
```

Arguments

`.x`	a node
`prefix`	The namespace prefix to use
`uri`	The namespace URI to use

Value

the node (invisibly)

`xml_structure` *Show the structure of an html/xml document.*

Description

Show the structure of an html/xml document without displaying any of the values. This is useful if you want to get a high level view of the way a document is organised. Compared to `xml_structure`, `html_structure` prints the id and class attributes.

Usage

```
xml_structure(x, indent = 2, file = "")

html_structure(x, indent = 2, file = "")
```

Arguments

x	HTML/XML document (or part there of)
indent	Number of spaces to ident
file	A connection, or a character string naming the file to print to. If "" (the default), cat prints to the standard output connection, the console unless redirected by sink. If it is "|cmd", the output is piped to the command given by 'cmd', by opening a pipe connection.

Examples

```
xml_structure(read_xml("<a><b><c/><c/></b><d/></a>"))

rproj <- read_html(system.file("extdata","r-project.html", package = "xml2"))
xml_structure(rproj)
xml_structure(xml_find_all(rproj, ".//p"))

h <- read_html("<body><p id = 'a'></p><p class = 'c d'></p></body>")
html_structure(h)
```

---

xml_text *Extract or modify the text*

Description

xml_text returns a character vector, xml_double returns a numeric vector, xml_integer returns an integer vector.

Usage

```
xml_text(x, trim = FALSE)

xml_text(x) <- value

xml_set_text(x, value)

xml_double(x)

xml_integer(x)
```

Arguments

x	A document, node, or node set.
trim	If TRUE will trim leading and trailing spaces.
value	character vector with replacement text.

Value

A character vector, the same length as x.

Examples

```
x <- read_xml("<p>This is some text. This is <b>bold!</b></p>")
xml_text(x)
xml_text(xml_children(x))

x <- read_xml("<x>This is some text. <x>This is some nested text.</x></x>")
xml_text(x)
xml_text(xml_find_all(x, "//x"))

x <- read_xml("<p>   Some text    </p>")
xml_text(x, trim = TRUE)

# xml_double() and xml_integer() are useful for extracting numeric
attributes
x <- read_xml("<plot><point x='1' y='2' /><point x='2' y='1' /></plot>")
xml_integer(xml_find_all(x, "//@x"))
```

---

xml_type *Determine the type of a node.*

Description

Determine the type of a node.

Usage

```
xml_type(x)
```

Arguments

x	A document, node, or node set.

Examples

```
x <- read_xml("<foo> a <b /> <![CDATA[ blah]]></foo>")
xml_type(x)
xml_type(xml_contents(x))
```

---

xml_url *The URL of an XML document*

Description

This is useful for interpreting relative urls with `url_relative()`.

Usage

```
xml_url(x)
```

Arguments

x	A node or document.

Value

A character vector of length 1. Returns NA if the name is not set.

Examples

```
catalog <- read_xml(xml2_example("cd_catalog.xml"))
xml_url(catalog)

x <- read_xml("<foo/>")
xml_url(x)
```

---

xml_validate *Validate XML schema*

Description

Validate an XML document against an XML 1.0 schema.

Usage

```
xml_validate(x, schema)
```

Arguments

x	A document, node, or node set.
schema	an XML document containing the schema

Value

TRUE or FALSE

Examples

```
# Example from https://msdn.microsoft.com/en-us/library/ms256129(v=vs.110).aspx
doc <- read_xml(system.file("extdata/order-doc.xml", package = "xml2"))
schema <- read_xml(system.file("extdata/order-schema.xml", package = "xml2"))
xml_validate(doc, schema)
```

Index