Package: zuxml (via r-universe)

October 28, 2026

Title Small XML Parser and Document Tree

Version 0.1.0.9000

Description Reads, navigates, and writes XML documents using a bundled copy of the 'Expat' parser (<https://libexpat.github.io/>), so that no system XML library is required. Documents are parsed into a faithful immutable tree exposed through a small vectorized navigation interface. Parsing is strict by default: document type declarations are rejected, external entity resolution is not compiled in, and configurable limits bound nesting depth and memory use. A registered C interface allows other packages to parse XML without linking against the parser themselves, and to feed it incrementally so that a large document need not be held in memory all at once.

License MIT + file LICENSE

Copyright See inst/COPYRIGHTS.

URL https://github.com/pedrobtz/zuxml, https://pedrobtz.github.io/zuxml/

BugReports https://github.com/pedrobtz/zuxml/issues

Depends R (>= 4.1)

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

Encoding UTF-8

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.1.1

Repository https://pedrobtz.r-universe.dev

Date/Publication 2026-10-28 07:46:42 UTC

RemoteUrl https://github.com/pedrobtz/zuxml

RemoteRef HEAD

RemoteSha d395cbc9ba6b27f70b5b21f6c62b64ee86b0bf91

Contents

xml_metadata 2
xml_navigate 3
xml_parse 4
xml_properties 6
xml_serialize 7
xml_table 8
zuxml-conditions 9
zuxml_info 10

Index **11**

`xml_metadata` *Document metadata*

Description

Document metadata

Usage

```
xml_version(x)

xml_encoding(x)

xml_standalone(x)
```

Arguments

`x`	A `zuxml_document`.

Value

A length-1 vector: a character string for `xml_version()` and `xml_encoding()`, a logical for `xml_standalone()`. Each is `NA` when the XML declaration did not state that property.

Examples

```
xml_encoding(xml_parse('<?xml version="1.0" encoding="UTF-8"?><a/>'))
```

`xml_navigate` *Navigate an XML document*

Description

Every accessor is vectorized: given a nodeset of length `n` it returns a result of length `n`, and the traversal functions return a single flat nodeset.

Usage

```
xml_root(x)

xml_parent(x)

xml_children(x)

xml_elements(x, name = NULL, ns = NULL)

xml_find(x, name = NULL, ns = NULL)

xml_find_first(x, name = NULL, ns = NULL)
```

Arguments

`x`	A `zuxml_document`, node, or nodeset.
`name`	Local name to match, or `NULL` for any.
`ns`	Namespace URI, `NA` for none, `NULL` for any.

Details

Filtering is uniform. `name` matches the *local* name. `ns = NULL` matches any namespace, `ns = NA` matches only nodes in no namespace, and a string matches that namespace URI. Matching never considers the prefix: two prefixes bound to one URI are the same name.

`xml_elements()` and `xml_find()` return elements only, including when `name` is `NULL`: they search by name, and only elements have one. Use `xml_children()` to reach text, comment and processing-instruction nodes, which it returns along with elements.

`xml_find()` returns one flat nodeset, so it cannot keep results aligned with its input: if one <book> has no <title>, the titles no longer line up with the books. `xml_find_first()` returns exactly one node per input node, its first matching descendant in document order, or a **missing node** when there is none.

A missing node gives `NA` from every accessor (`xml_name()`, `xml_text()`, `xml_attr()`, `xml_type()` and the rest) and from `xml_serialize()`, and an empty named vector from `xml_attrs()`. Traversals skip it: it has no parent, children or descendants. `xml_attr()`'s `default` applies to it, as to any node lacking the attribute.

Value

`xml_root()`, `xml_parent()`, `xml_children()`, `xml_elements()`, `xml_find()` and `xml_find_first()` return a nodeset. `xml_find_first()`'s has the length of `x`.

Examples

```
doc <- xml_parse("<r><a><b>1</b></a><b>2</b></r>")
xml_name(xml_children(xml_root(doc)))
xml_text(xml_find(doc, "b"))

# One result per book, NA where a book has no title.
books <- xml_elements(xml_root(xml_parse(
  "<r><book><title>A</title></book><book/><book><title>C</title></book></r>")))
xml_text(xml_find_first(books, "title"))
```

---

xml_parse *Parse an XML document*

Description

`xml_parse()` parses XML held in memory; `xml_read()` parses a file, a URL or a connection.

Usage

```
xml_parse(
  x,
  encoding = NULL,
  comments = TRUE,
  pis = TRUE,
  allow_doctype = FALSE,
  max_depth = 256L,
  max_nodes = 1e+07,
  max_attrs = 4096L,
  max_text = 64 * 1024^2,
  max_memory = 1024 * 1024^2
)

xml_read(path, encoding = NULL, ...)
```

Arguments

`x`	A single string, or a raw vector, containing XML. A character vector of any other length is an error: to parse lines read with `readLines()`, join them first with `paste(x, collapse = "\n")`.
`encoding`	Encoding of the input. `NULL` (default) lets the parser detect it from a byte-order mark or the XML declaration. An explicit value overrides the declaration, which

is what an HTTP `charset` should do. Encodings that Expat cannot handle natively are transcoded with `iconv()`. A character `x` is always UTF-8 by the time it is parsed, so for one, `encoding` must be `NULL` or `"UTF-8"`.

`comments`, `pis` Retain comment and processing-instruction nodes.

`allow_doctype` Accept a `DOCTYPE` declaration. An internal subset is rejected even when this is `TRUE`, because it is the only place a document can declare entities.

`max_depth`, `max_nodes`, `max_attrs`, `max_text`, `max_memory`
Resource limits: the nesting depth, the number of nodes, the attributes on one element, the bytes in one text node, and the bytes the document may allocate. Each must be a single positive whole number, or `Inf` for the largest value the parser can represent. `max_nodes` is capped at `.Machine$integer.max`, because node ids are R integers. Each limit has its own error condition.

`path` Path to a file, a URL, or a connection.

`...` Passed on to `xml_parse()`.

Details

Parsing is strict and secure by default: document type declarations are rejected, general entities are not compiled in at all, and the limits below bound what a hostile document can cost. See `vignette("security")` for the threat model, or `zuxml_info()` for the compiled-in policy.

`xml_read()` streams its input: bytes are fed to the parser as they are read, so the document is never held whole in memory, only the tree is. A string is taken as a URL if it starts with `http://`, `https://`, `ftp://`, `ftps://` or `file://`, and is otherwise a path. A connection that is not open is opened in binary mode for the call and closed afterwards; one that is already open must be in binary mode (`"rb"`) and blocking, is read from its current position, and is left open. An `encoding` that Expat cannot handle natively needs the whole input before it can be transcoded, so that case is read fully first.

Value

A `zuxml_document`.

See Also

zuxml-conditions for the errors these raise.

Examples

```
doc <- xml_parse("<catalog><book id='1'><title>XML</title></book></catalog>")
xml_text(xml_find(doc, "title"))

f <- tempfile(fileext = ".xml")
xml_write(doc, f)
xml_read(f)
xml_read(gzfile(f))          # any connection, compressed or not
xml_read(paste0("file://", f))
## Not run:
xml_read("https://www.w3.org/TR/2008/REC-xml-20081126/REC-xml-20081126.xml")
```

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

`xml_properties` *Read properties of XML nodes*

Description

All of these are vectorized over a nodeset.

Usage

```
xml_name(x)

xml_local(x)

xml_ns(x)

xml_prefix(x)

xml_type(x)

xml_attrs(x)

xml_attr(x, name, ns = NULL, default = NA_character_)

xml_text(x, recursive = TRUE, trim = FALSE)
```

Arguments

`x`	A node or nodeset.
`name`	Attribute local name.
`ns`	Namespace URI, `NA` for none, `NULL` for any.
`default`	Value for nodes lacking the attribute.
`recursive`	Concatenate text of all descendants (default) or only direct text children.
`trim`	Trim leading and trailing whitespace from the result.

Value

A character vector, except `xml_attrs()` which returns a list of named character vectors.

Examples

```
doc <- xml_parse("<p>Hello <em>XML</em> world</p>")
xml_text(xml_root(doc))
xml_text(xml_root(doc), recursive = FALSE)
```

xml_serialize *Serialize XML*

Description

Writes a node and its subtree back to XML text.

Usage

```
xml_serialize(x, declaration = FALSE)

xml_write(x, path, declaration = TRUE)

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

## S3 method for class 'zuxml_document'
as.character(x, ...)
```

Arguments

x	A zuxml_document, node, or nodeset.
declaration	Prepend an XML declaration.
path	File to write to.
...	Ignored.

Details

Element order, attribute order, text order, mixed-content interleaving and namespace semantics are preserved. The original *formatting* is not: parsing does not retain quote style, whitespace between attributes, CDATA boundaries, entity spelling, or whether an empty element was written as <a/> or <a>. zuxml is not a lossless source editor.

Each element of the result is escaped for its own node. Two adjacent text nodes – which is what comments = FALSE or pis = FALSE leaves behind where a dropped node used to sit – are therefore escaped independently, so concatenating the pieces yourself can produce content a parser will reject. Serialize the common parent instead, which escapes across the whole subtree.

Value

xml_serialize() returns a character vector, one element per node. xml_write() returns path invisibly.

Examples

```
doc <- xml_parse("<p>Hello <em>XML</em> world</p>")
xml_serialize(doc)
```

xml_table — *Extract tables and lists*

Description

`xml_table()` turns `table` elements into data frames, and `xml_list()` turns `ul` and `ol` elements into R lists. Both read well-formed XML, such as XHTML or a table embedded in another XML format. They do not apply HTML parsing rules: a `table` without `tbody` has exactly the rows it contains.

Usage

```
xml_table(x, header = NA, trim = TRUE, ns = NULL, max_cells = 1e+07)

xml_list(x, trim = TRUE, ns = NULL)
```

Arguments

`x`	A node or nodeset of `table` elements for `xml_table()`, or of `ul` and `ol` elements for `xml_list()`. A document stands for its root.
`header`	Use the first row as column names. `NA`, the default, does so when every cell of the first row is a `th`.
`trim`	Trim leading and trailing whitespace from each value.
`ns`	Namespace URI of the table or list elements, `NA` for none, `NULL` for any.
`max_cells`	The most cells one table may expand to. A positive whole number, or `Inf`.

Details

Tables. The rows are the `tr` elements that are children of the `table`, or of its `thead`, `tbody` or `tfoot`, in document order. A table nested inside a cell is not descended into; extract it separately. The cells are the `td` and `th` elements of a row, and a cell's value is its text, as `xml_text()` gives it. Every column is character: convert types yourself, for example with `type.convert(df, as.is = TRUE)`.

`colspan` and `rowspan` are expanded by repeating the cell's value. A span that is not a positive whole number counts as 1, and spans are capped at HTML's own limits, 1000 columns and 65534 rows. A row span never adds rows past the end of the table. Short rows are padded with `NA`.

Expansion can make a small document produce a very large table, so the expanded size of each table is bounded by `max_cells`, and exceeding it is a `zuxml_limit_error`.

Lists. Each `li` child of a `ul` or `ol` becomes one item. An item is its text as a string or, when the `li` has `ul` or `ol` children, a list with two elements: `text`, the item's own text without the nested lists', and `items`, the items of its nested lists, extracted the same way. Only direct `ul` and `ol` children of an `li` count as nested lists. Comments and processing instructions inside an item are ignored.

Value

A list with one element per node in `x`: a data frame for `xml_table()`, a list of items for `xml_list()`.

See Also

xml_find() to locate the tables or lists in a document, and zuxml-conditions for the errors these raise.

Examples

```
doc <- xml_parse("<table>
  <tr><th>fruit</th><th>price</th></tr>
  <tr><td>apple</td><td>1.20</td></tr>
  <tr><td>pear</td><td>0.90</td></tr>
</table>")
xml_table(xml_root(doc))[[1]]

doc <- xml_parse("<ul>
  <li>fruit<ul><li>apple</li><li>pear</li></ul></li>
  <li>bread</li>
</ul>")
str(xml_list(xml_root(doc))[[1]])
```

---

zuxml-conditions *Conditions raised by zuxml*

Description

Every error zuxml raises carries a condition class, so that it can be caught by kind rather than by matching the message, which is not stable. Every class below is a subclass of `zuxml_error`.

Details

`zuxml_invalid_argument` An argument was not usable: an `x` that is not a single string or a raw vector, an `encoding` other than UTF-8 for character input, a limit that is not a positive whole number, or a flag that is not `TRUE` or `FALSE`.

`zuxml_parse_error` The input is not well-formed XML, ends too early, or references an undefined entity.

`zuxml_encoding_error` The input is not valid in its encoding, or names an encoding that neither Expat nor iconv() can read.

`zuxml_doctype_error` The document has a `DOCTYPE` declaration and `allow_doctype` is `FALSE`, or it has an internal subset, which is refused either way.

`zuxml_limit_error` A resource limit was reached. The subclasses `zuxml_depth_limit`, `zuxml_node_limit`, `zuxml_attr_limit`, `zuxml_text_limit` and `zuxml_memory_limit` name which one. xml_serialize() raises the parent class alone when a node's text is longer than an R string can hold.

`zuxml_memory_error` An allocation failed.

`zuxml_cancelled` A handler stopped the parse. It can only happen through the C API, where a downstream package supplies the handler; no exported R function raises it.

A condition raised by a parse carries `line`, `column` and `byte_offset`, the position where the parser stopped; `status`, the C status's enumerator name, such as `"ZUX_ERR_DEPTH_LIMIT"`; and

expat_code, Expat's own error code, which is NA when the failure was not Expat's. It is there for diagnostics: do not branch on it. A limit error also carries limit, the argument's name, such as "max_depth", and limit_value, the value it was given. A zuxml_invalid_argument condition carries arg, the name of the argument at fault.

Examples

```
tryCatch(
  xml_parse("<a><b></a>"),
  zuxml_parse_error = function(e) c(line = e$line, column = e$column)
)
tryCatch(
  xml_parse("<a><b/></a>", max_depth = 1),
  zuxml_limit_error = function(e) e$limit
)
```

---

zuxml_info *Report the zuxml build configuration*

Description

Reports the vendored 'Expat' version and the parser policy compiled into this build. Intended for diagnostics and for security audits: in a correctly built zuxml, dtd and general_entities are both FALSE and entropy names a real operating-system entropy source.

Usage

```
zuxml_info()
```

Value

An object of class zuxml_info: a list with elements zuxml_version, expat_version, namespaces, dtd, general_entities, context_bytes, xml_char_bytes, byteorder, entropy and parser_ok.

Examples

```
zuxml_info()
```

Index

as.character.zuxml_document (xml_serialize), 7
as.character.zuxml_nodeset (xml_serialize), 7

connection, *5*

iconv(), *5*, *9*

readLines(), *4*

xml_attr (xml_properties), 6
xml_attr(), *3*
xml_attrs (xml_properties), 6
xml_attrs(), *3*
xml_children (xml_navigate), 3
xml_elements (xml_navigate), 3
xml_encoding (xml_metadata), 2
xml_find (xml_navigate), 3
xml_find(), *9*
xml_find_first (xml_navigate), 3
xml_list (xml_table), 8
xml_local (xml_properties), 6
xml_metadata, 2
xml_name (xml_properties), 6
xml_name(), *3*
xml_navigate, 3
xml_ns (xml_properties), 6
xml_parent (xml_navigate), 3
xml_parse, 4
xml_prefix (xml_properties), 6
xml_properties, 6
xml_read (xml_parse), 4
xml_root (xml_navigate), 3
xml_serialize, 7
xml_serialize(), *3*, *9*
xml_standalone (xml_metadata), 2
xml_table, 8
xml_text (xml_properties), 6
xml_text(), *3*, *8*
xml_type (xml_properties), 6
xml_type(), *3*
xml_version (xml_metadata), 2
xml_write (xml_serialize), 7

zuxml-conditions, *5*, 9, *9*
zuxml_info, 10