

Package ‘selectr’

June 23, 2026

Type Package

Title Translate CSS Selectors to XPath Expressions

Version 0.6-0

License BSD_3_clause + file LICENCE

Encoding UTF-8

Depends R (>= 3.3)

Imports R6

Suggests testthat, XML, xml2

URL <https://sjp.co.nz/projects/selectr/>

BugReports <https://github.com/sjp/selectr/issues>

Description Translates a CSS selector into an equivalent XPath expression. This allows us to use CSS selectors when working with the XML package as it can only evaluate XPath expressions. Also provided are convenience functions useful for using CSS selectors on XML nodes. This package is a port of the Python package 'cssselect' (<<https://cssselect.readthedocs.io/>>).

NeedsCompilation no

Author Simon Potter [aut, trl, cre],
Simon Sapin [aut],
Ian Bicking [aut]

Maintainer Simon Potter <simon@sjp.co.nz>

Repository CRAN

Date/Publication 2026-06-23 05:10:21 UTC

Contents

css_to_xpath	2
querySelectorAll	4
Index	7

css_to_xpath

*Translate a CSS selector to an equivalent XPath expression.***Description**

This function aims to create an XPath expression equivalent to what would be matched by the given CSS selector. The reason the translation is required is because the XML and xml2 packages, being a libxml2 wrappers, can only evaluate XPath expressions.

Using this function, it is possible to search an XML tree without the prerequisite of knowing XPath.

Usage

```
css_to_xpath(selector,
             prefix = "descendant-or-self::",
             translator = "generic")
```

Arguments

selector	A character vector of CSS selectors.
prefix	The prefixes to apply to the resulting XPath expressions. The default or "" are most commonly used.
translator	The type of translator that will be used. Possible options are generic (the default), or html or xhtml.

Details

Each selector given to this function will be translated to an equivalent XPath expression. The resulting XPath expression can be given a prefix which determines the scope of the expression. The default prefix determines the scope to be the node itself and all descendants of the node. Most commonly the prefix is either the default or "", unless it is known what scope a particular XPath expression should have.

A selector starting with the :scope pseudo-class is anchored at the node the expression is evaluated from: the prefix argument is ignored and the expression begins with the XPath self axis instead. For example, ":scope > a" translates to "self::*/a", matching only the a children of the queried node, and a bare ":scope" translates to "self::*", matching the queried node itself. :scope anywhere else in a selector (after a combinator, or within a functional pseudo-class such as :is() or :has()) cannot be expressed in XPath 1.0 and is an error.

The of-type pseudo-classes (:first-of-type, :last-of-type, :only-of-type, :nth-of-type() and :nth-last-of-type()) are only supported when their compound selector names an element, as in "p:first-of-type". Applied to the universal selector, as in "*:first-of-type", they would have to compare each sibling's name against the matched element's own name, which XPath 1.0 cannot express, so the translation is an error. The Python 'cssselect' library, from which selectr is ported, has the same limitation.

The Selectors 4 column combinator ("a || b") and the column pseudo-classes :nth-col() and :nth-last-col() are also not supported: which column a cell belongs to depends on table-layout

arithmetic (colspan/rowspan carry-over) that XPath 1.0 cannot express. Both are rejected with an error.

`:empty` deliberately keeps the Selectors 3 semantics that all current browsers implement: an element containing only white space, such as `<p></p>`, does not match. (The Selectors 4 specification loosened `:empty` to also match white-space-only elements, but no browser has shipped that change.)

`:lang()` ranges are matched as RFC 4647 language ranges, so a wildcard may appear as a whole range (`:lang(*)`), as a trailing subtag (`:lang(en-*)`), or in a non-trailing position (`:lang(*-CH)`, `:lang(de-*DE)`; quoted or not), the last of these being RFC 4647 extended filtering. The `html` and `xhtml` translators approximate a non-trailing wildcard from the nearest `lang`-attributed ancestor (an element matches `:lang(*-CH)` when its language tag carries a `CH` subtag in any position). The generic translator has only XPath's `lang()` function, which cannot express extended filtering, so there a non-trailing wildcard is rejected with an error rather than silently mis-matching.

`:dir()` translates to a never-matching expression with every translator, including `html`: an element's resolved directionality also depends on `dir="auto"`, `bdi`, and form-control rules that a static document cannot answer, so unlike `:lang()` it is not approximated from ancestor attributes.

The translator used is usually unnecessary to specify as the default is sufficient for most cases. However, it is of use when creating expressions relating to (X)HTML pseudo elements and languages. In particular it qualifies the following pseudo selectors to apply only to relevant (X)HTML elements: `:checked`, `:disabled`, `:enabled`, `:link`, `:optional` and `:required`. Because `type` is an HTML enumerated attribute, the form-state pseudo-classes match its keywords ASCII case-insensitively, so `<input type="RADIO">` is `:checked` and `<input type="HIDDEN">` is excluded from `:disabled` just as their lower-case spellings are. An `<input>` with no type defaults to text and so participates in `:enabled/:disabled`. A disabled `<fieldset>` disables its descendant form controls for `:disabled` and `:enabled`, except those inside its first `<legend>` child, which stay `:enabled`; nested disabled fieldsets are handled correctly.

When the translator is set to `html`, all elements and attributes will be converted to lower case. This restriction is removed when the translator is `xhtml` (or the default generic translator).

Value

A character vector of XPath expressions.

Author(s)

Simon Potter

References

CSS Selectors Level 4 <https://www.w3.org/TR/selectors-4/>, XPath <https://www.w3.org/TR/xpath/>.

Examples

```
css_to_xpath(".testclass")
css_to_xpath("#testid", prefix = "")
css_to_xpath("#testid .testclass")
css_to_xpath(":scope > .testclass")
css_to_xpath(":checked", translator = "html")
```

querySelectorAll	<i>Find nodes that match a group of CSS selectors in an XML tree.</i>
------------------	---

Description

The purpose of these functions is to mimic the functionality of the `querySelector` and `querySelectorAll` functions present in Internet browsers. This is so we can succinctly query an XML tree for nodes matching a CSS selector.

Namespaced functions `querySelectorNS` and `querySelectorAllNS` are also provided to search relative to a given namespace.

Usage

```
querySelector(doc, selector, ns = NULL, ...)
querySelectorAll(doc, selector, ns = NULL, ...)
querySelectorNS(doc, selector, ns,
                prefix = "descendant-or-self::", ...)
querySelectorAllNS(doc, selector, ns,
                  prefix = "descendant-or-self::", ...)
```

Arguments

<code>doc</code>	The XML document or node to be evaluated against.
<code>selector</code>	A selector used to query <code>doc</code> . This must be a single character string.
<code>ns</code>	The namespace that the query will be filtered to. This is a named list or vector which has as its name a namespace, and its value is the namespace URI. This can be ignored for the un-namespaced functions.
<code>prefix</code>	The prefix to apply to the resulting XPath expression. The default or "" are most commonly used.
<code>...</code>	Parameters to be passed onto <code>css_to_xpath</code> .

Details

The `querySelectorNS` and `querySelectorAllNS` functions are convenience functions for working with namespaced documents. They filter out all content that does not belong within the given namespaces. Note that when searching for particular elements in a selector, they must have a namespace prefix, e.g. `"svg|g"`.

The namespace argument, `ns`, is simply passed on to `getNodeSet` or `xml_find_all` if it is necessary to use a namespace present within the document. This can be ignored for content lacking a namespace, which is usually the case when using `querySelector` or `querySelectorAll`.

A selector starting with the `:scope` pseudo-class is anchored at the queried node itself: `querySelectorAll(node, ":scope > a")` returns only the `a` children of `node`, where `querySelectorAll(node, "a")` would return all of its `a` descendants. `:scope` after a combinator or within a functional pseudo-class is an error (it cannot be expressed in XPath 1.0).

Value

For `querySelector`, the result is a single node that represents the first matched node from a selector. If no matching nodes are found, `NULL` is returned.

For `querySelectorAll`, the result is a list of XML nodes. This list may be empty in the case that no match is found.

The `querySelectorNS` and `querySelectorAllNS` functions return the same type of content as their un-namespaced counterparts.

Author(s)

Simon Potter

References

CSS Selectors Level 4 <https://www.w3.org/TR/selectors-4/>, XPath <https://www.w3.org/TR/xpath/>, `querySelectorAll` <https://developer.mozilla.org/en-US/docs/DOM/Document.querySelectorAll> and <https://www.w3.org/TR/selectors-api/#interface-definitions>.

Examples

```
hasXML <- require(XML)
hasxml2 <- require(xml2)

if (!hasXML && !hasxml2)
  return() # can't demo without XML or xml2 packages present

parseFn <- if (hasXML) xmlParse else read_xml
# Demo for working with the XML package (if present, otherwise xml2)
exdoc <- parseFn('<a><b class="aclass"><c id="anid"></a>')
querySelector(exdoc, "#anid") # Returns the matching node
querySelector(exdoc, ".aclass") # Returns the matching node
querySelector(exdoc, "b, c") # First match from grouped selection
querySelectorAll(exdoc, "b, c") # Grouped selection
querySelectorAll(exdoc, "b") # A list of length one
querySelector(exdoc, "d") # No match
querySelectorAll(exdoc, "d") # No match

# Read in a document where two namespaces are being set:
# SVG and MathML
svgdoc <- parseFn(system.file("demos/svg-mathml.svg",
                             package = "selectr"))
# Search for <script> elements in the SVG namespace
querySelectorNS(svgdoc, "svg|script",
               c(svg = "http://www.w3.org/2000/svg"))
querySelectorAllNS(svgdoc, "svg|script",
                  c(svg = "http://www.w3.org/2000/svg"))
# MathML content is *within* SVG content,
# search for <mtext> elements within the MathML namespace
querySelectorNS(svgdoc, "math|mtext",
               c(math = "http://www.w3.org/1998/Math/MathML"))
```

```

querySelectorAllNS(svgdoc, "math|mtext",
                  c(math = "http://www.w3.org/1998/Math/MathML"))
# Search for *both* SVG and MathML content
querySelectorAllNS(svgdoc, "svg|script, math|mo",
                  c(svg = "http://www.w3.org/2000/svg",
                    math = "http://www.w3.org/1998/Math/MathML"))

if (!hasXML)
  return() # already demo'd xml2

# Demo for working with the xml2 package
exdoc <- read_xml('<a><b class="aclass"><c id="anid"></a>')
querySelector(exdoc, "#anid") # Returns the matching node
querySelector(exdoc, ".aclass") # Returns the matching node
querySelector(exdoc, "b, c") # First match from grouped selection
querySelectorAll(exdoc, "b, c") # Grouped selection
querySelectorAll(exdoc, "b") # A list of length one
querySelector(exdoc, "d") # No match
querySelectorAll(exdoc, "d") # No match

# Read in a document where two namespaces are being set:
# SVG and MathML
svgdoc <- read_xml(system.file("demos/svg-mathml.svg",
                              package = "selectr"))
# Search for <script> elements in the SVG namespace
querySelectorNS(svgdoc, "svg|script",
               c(svg = "http://www.w3.org/2000/svg"))
querySelectorAllNS(svgdoc, "svg|script",
                  c(svg = "http://www.w3.org/2000/svg"))
# MathML content is *within* SVG content,
# search for <mtext> elements within the MathML namespace
querySelectorNS(svgdoc, "math|mtext",
               c(math = "http://www.w3.org/1998/Math/MathML"))
querySelectorAllNS(svgdoc, "math|mtext",
                  c(math = "http://www.w3.org/1998/Math/MathML"))
# Search for *both* SVG and MathML content
querySelectorAllNS(svgdoc, "svg|script, math|mo",
                  c(svg = "http://www.w3.org/2000/svg",
                    math = "http://www.w3.org/1998/Math/MathML"))

```

Index

`css_to_xpath`, [2](#)

`getNodeSet`, [4](#)

`querySelector (querySelectorAll)`, [4](#)

`querySelectorAll`, [4](#)

`querySelectorAllNS (querySelectorAll)`, [4](#)

`querySelectorNS (querySelectorAll)`, [4](#)

`xml_find_all`, [4](#)