Search Site Search

Install
: sudo gem install nokogiri

Contribute
: github.com/tenderlove/nokogiri (http://github.com/tenderlove/nokogiri)

An HTML, XML, SAX, & Reader parser with the ability to search documents via XPath or CSS3 selectors… and much more

Nokogiri 鋸 (/)

- Installation (/tutorials/installing_nokogiri.html)
- Tutorials (/tutorials)
- Getting Help (/tutorials/getting_help.html)

Files

Hide (#)

- CHANGELOG.ja.rdoc (/CHANGELOG_ja_rdoc.html)
- CHANGELOG.rdoc (/CHANGELOG_rdoc.html)
- Manifest.txt (/Manifest_txt.html)
- README.ja.rdoc (/README_ja_rdoc.html)
- README.rdoc (/README_rdoc.html)

Classes

Hide (#)

Filter Classes

- Nokogiri (/Nokogiri.html)
- Nokogiri::CSS (/Nokogiri/CSS.html)
- Nokogiri::CSS::Node (/Nokogiri/CSS/Node.html)
- Nokogiri::CSS::Parser (/Nokogiri/CSS/Parser.html)
- Nokogiri::CSS::SyntaxError (/Nokogiri/CSS/SyntaxError.html)
- Nokogiri::CSS::Tokenizer (/Nokogiri/CSS/Tokenizer.html)
- Nokogiri::CSS::Tokenizer::ScanError (/Nokogiri/CSS/Tokenizer/ScanError.html)
- Nokogiri::Decorators (/Nokogiri/Decorators.html)
- Nokogiri::Decorators::Slop (/Nokogiri/Decorators/Slop.html)
- Nokogiri::EncodingHandler (/Nokogiri/EncodingHandler.html)
- Nokogiri::HTML (/Nokogiri/HTML.html)

- Nokogiri::HTML::Builder (/Nokogiri/HTML/Builder.html)
- Nokogiri::HTML::Document (/Nokogiri/HTML/Document.html)
- Nokogiri::HTML::DocumentFragment (/Nokogiri/HTML/DocumentFragment.html)
- Nokogiri::HTML::ElementDescription (/Nokogiri/HTML/ElementDescription.html)
- Nokogiri::HTML::EntityDescription (/Nokogiri/HTML/EntityDescription.html)
- Nokogiri::HTML::EntityLookup (/Nokogiri/HTML/EntityLookup.html)
- Nokogiri::HTML::SAX (/Nokogiri/HTML/SAX.html)
- Nokogiri::HTML::SAX::Parser (/Nokogiri/HTML/SAX/Parser.html)
- Nokogiri::HTML::SAX::ParserContext (/Nokogiri/HTML/SAX/ParserContext.html)
- Nokogiri::SyntaxError (/Nokogiri/SyntaxError.html)
- Nokogiri::XML (/Nokogiri/XML.html)
- Nokogiri::XML::Attr (/Nokogiri/XML/Attr.html)
- Nokogiri::XML::AttributeDecl (/Nokogiri/XML/AttributeDecl.html)
- Nokogiri::XML::Builder (/Nokogiri/XML/Builder.html)
- Nokogiri::XML::CDATA (/Nokogiri/XML/CDATA.html)
- Nokogiri::XML::CharacterData (/Nokogiri/XML/CharacterData.html)
- Nokogiri::XML::Comment (/Nokogiri/XML/Comment.html)
- Nokogiri::XML::DTD (/Nokogiri/XML/DTD.html)
- Nokogiri::XML::Document (/Nokogiri/XML/Document.html)
- Nokogiri::XML::DocumentFragment (/Nokogiri/XML/DocumentFragment.html)
- Nokogiri::XML::Element (/Nokogiri/XML/Element.html)
- Nokogiri::XML::ElementContent (/Nokogiri/XML/ElementContent.html)
- Nokogiri::XML::ElementDecl (/Nokogiri/XML/ElementDecl.html)
- Nokogiri::XML::EntityDecl (/Nokogiri/XML/EntityDecl.html)
- Nokogiri::XML::EntityReference (/Nokogiri/XML/EntityReference.html)
- Nokogiri::XML::Namespace (/Nokogiri/XML/Namespace.html)
- Nokogiri::XML::Node (/Nokogiri/XML/Node.html)
- Nokogiri::XML::Node::SaveOptions (/Nokogiri/XML/Node/SaveOptions.html)
- Nokogiri::XML::NodeSet (/Nokogiri/XML/NodeSet.html)
- Nokogiri::XML::Notation (/Nokogiri/XML/Notation.html)
- Nokogiri::XML::PP (/Nokogiri/XML/PP.html)
- Nokogiri::XML::PP::CharacterData (/Nokogiri/XML/PP/CharacterData.html)
- Nokogiri::XML::PP::Node (/Nokogiri/XML/PP/Node.html)
- Nokogiri::XML::ParseOptions (/Nokogiri/XML/ParseOptions.html)
- Nokogiri::XML::ProcessingInstruction (/Nokogiri/XML/ProcessingInstruction.html)
- Nokogiri::XML::Reader (/Nokogiri/XML/Reader.html)
- Nokogiri::XML::RelaxNG (/Nokogiri/XML/RelaxNG.html)
- Nokogiri::XML::SAX (/Nokogiri/XML/SAX.html)
- Nokogiri::XML::SAX::Document (/Nokogiri/XML/SAX/Document.html)
- Nokogiri::XML::SAX::Parser (/Nokogiri/XML/SAX/Parser.html)
- Nokogiri::XML::SAX::Parser::Attribute (/Nokogiri/XML/SAX/Parser/Attribute.html)
- Nokogiri::XML::SAX::ParserContext (/Nokogiri/XML/SAX/ParserContext.html)
- Nokogiri::XML::SAX::PushParser (/Nokogiri/XML/SAX/PushParser.html)
- Nokogiri::XML::Schema (/Nokogiri/XML/Schema.html)

- Nokogiri::XML::SyntaxError (/Nokogiri/XML/SyntaxError.html)
- Nokogiri::XML::Text (/Nokogiri/XML/Text.html)
- Nokogiri::XML::XPath (/Nokogiri/XML/XPath.html)
- Nokogiri::XML::XPath::SyntaxError (/Nokogiri/XML/XPath/SyntaxError.html)
- Nokogiri::XML::XPathContext (/Nokogiri/XML/XPathContext.html)
- Nokogiri::XSLT (/Nokogiri/XSLT.html)
- Nokogiri::XSLT::Stylesheet (/Nokogiri/XSLT/Stylesheet.html)
- Object (/Object.html)
- XSD (/XSD.html)
- XSD::XMLParser (/XSD/XMLParser.html)
- XSD::XMLParser::Nokogiri (/XSD/XMLParser/Nokogiri.html)

Methods

Hide (#)

Filter Methods

- % (#method-i-25)
- / (#method-i-2F)
- << (#method-i-3C-3C)
- <=> (#method-i-3C-3D-3E)
- == (#method-i-3D-3D)
- > (#method-i-3E)
- [] (#method-i-5B-5D)
- []= (#method-i-5B-5D-3D)
- accept (#method-i-accept)
- add_child (#method-i-add_child)
- add_namespace (#method-i-add_namespace)
- add_namespace_definition (#method-i-add_namespace_definition)
- add_next_sibling (#method-i-add_next_sibling)
- add_previous_sibling (#method-i-add_previous_sibling)
- after (#method-i-after)
- ancestors (#method-i-ancestors)
- at (#method-i-at)
- at_css (#method-i-at_css)
- at_xpath (#method-i-at_xpath)
- attr (#method-i-attr)
- attribute (#method-i-attribute)
- attribute_nodes (#method-i-attribute_nodes)
- attribute_with_ns (#method-i-attribute_with_ns)
- attributes (#method-i-attributes)
- before (#method-i-before)
- blank? (#method-i-blank-3F)
- cdata? (#method-i-cdata-3F)

- child (#method-i-child)
- children (#method-i-children)
- children= (#method-i-children-3D)
- clone (#method-i-clone)
- comment? (#method-i-comment-3F)
- content (#method-i-content)
- content= (#method-i-content-3D)
- create_external_subset (#method-i-create_external_subset)
- create_internal_subset (#method-i-create_internal_subset)
- css (#method-i-css)
- css_path (#method-i-css_path)
- decorate! (#method-i-decorate-21)
- default_namespace= (#method-i-default_namespace-3D)
- delete (#method-i-delete)
- description (#method-i-description)
- do_xinclude (#method-i-do_xinclude)
- document (#method-i-document)
- dup (#method-i-dup)
- each (#method-i-each)
- elem? (#method-i-elem-3F)
- element? (#method-i-element-3F)
- element_children (#method-i-element_children)
- elements (#method-i-elements)
- encode_special_chars (#method-i-encode_special_chars)
- external_subset (#method-i-external_subset)
- first_element_child (#method-i-first_element_child)
- fragment (#method-i-fragment)
- fragment? (#method-i-fragment-3F)
- get_attribute (#method-i-get_attribute)
- has_attribute? (#method-i-has_attribute-3F)
- html? (#method-i-html-3F)
- inner_html (#method-i-inner_html)
- inner_html= (#method-i-inner_html-3D)
- inner_text (#method-i-inner_text)
- internal_subset (#method-i-internal_subset)
- key? (#method-i-key-3F)
- keys (#method-i-keys)
- last_element_child (#method-i-last_element_child)
- line (#method-i-line)
- matches? (#method-i-matches-3F)
- name (#method-i-name)
- name= (#method-i-name-3D)
- namespace (#method-i-namespace)
- namespace= (#method-i-namespace-3D)

- namespace_definitions (#method-i-namespace_definitions)
- namespace_scopes (#method-i-namespace_scopes)
- namespaced_key? (#method-i-namespaced_key-3F)
- namespaces (#method-i-namespaces)
- new (#method-c-new)
- next (#method-i-next)
- next_element (#method-i-next_element)
- next_sibling (#method-i-next_sibling)
- node_name (#method-i-node_name)
- node_name= (#method-i-node_name-3D)
- node_type (#method-i-node_type)
- parent (#method-i-parent)
- parent= (#method-i-parent-3D)
- parse (#method-i-parse)
- path (#method-i-path)
- pointer_id (#method-i-pointer_id)
- previous (#method-i-previous)
- previous= (#method-i-previous-3D)
- previous_element (#method-i-previous_element)
- previous_sibling (#method-i-previous_sibling)
- read_only? (#method-i-read_only-3F)
- remove (#method-i-remove)
- remove_attribute (#method-i-remove_attribute)
- replace (#method-i-replace)
- search (#method-i-search)
- serialize (#method-i-serialize)
- set_attribute (#method-i-set_attribute)
- swap (#method-i-swap)
- text (#method-i-text)
- text? (#method-i-text-3F)
- to_html (#method-i-to_html)
- to_s (#method-i-to_s)
- to_str (#method-i-to_str)
- to_xhtml (#method-i-to_xhtml)
- to_xml (#method-i-to_xml)
- traverse (#method-i-traverse)
- type (#method-i-type)
- unlink (#method-i-unlink)
- values (#method-i-values)
- write_html_to (#method-i-write_html_to)
- write_to (#method-i-write_to)
- write_xhtml_to (#method-i-write_xhtml_to)
- write_xml_to (#method-i-write_xml_to)
- xml? (#method-i-xml-3F)

- xpath (#method-i-xpath)

Class Nokogiri::XML::Node (/Nokogiri/XML/Node.html) inherits from Object (/Object.html)

Nokogiri::XML::Node (Node.html) is your window to the fun filled world of dealing with XML (../XML.html) and HTML (../HTML.html) tags. A Nokogiri::XML::Node (Node.html) may be treated similarly to a hash with regard to attributes. For example (from irb):

view source (#viewSource)print (#printSource)? (#about)

```
irb(main):004:0> node
=> <a href="#foo" id="link">link</a>
irb(main):005:0> node['href']
=> "#foo"
irb(main):006:0> node.keys
=> ["href", "id"]
irb(main):007:0> node.values
=> ["#foo", "link"]
irb(main):008:0> node['class'] = 'green'
=> "green"
irb(main):009:0> node
=> <a href="#foo" id="link" class="green">link</a>
irb(main):010:0>
```

See Nokogiri::XML::Node#[] and Nokogiri::XML#[]= for more information.

Nokogiri::XML::Node (Node.html) also has methods that let you move around your tree. For navigating your tree, see:

- Nokogiri::XML::Node#parent (Node.html#method-i-parent)
- Nokogiri::XML::Node#children (Node.html#method-i-children)
- Nokogiri::XML::Node#next (Node.html#method-i-next)
- Nokogiri::XML::Node#previous (Node.html#method-i-previous)

You may search this node's subtree using Node#xpath (Node.html#method-i-xpath) and Node#css (Node.html#method-i-css)

Constants

ATTRIBUTE_DECL

Attribute declaration type

ATTRIBUTE_NODE

Attribute node type

CDATA_SECTION_NODE

CDATA (CDATA.html) node type, see Nokogiri::XML::Node#cdata? (Node.html#method-i-cdata-3F)

COMMENT_NODE

Comment (Comment.html) node type, see Nokogiri::XML::Node#comment? (Node.html#method-i-comment-3F)

DOCB_DOCUMENT_NODE

DOCB document node type

DOCUMENT_FRAG_NODE

Document (Document.html) fragment node type

DOCUMENT_NODE

Document (Document.html) node type, see Nokogiri::XML::Node#xml? (Node.html#method-i-xml-3F)

DOCUMENT_TYPE_NODE

Document (Document.html) type node type

DTD_NODE

DTD (DTD.html) node type

ELEMENT_DECL

Element (Element.html) declaration type

ELEMENT_NODE

Element (Element.html) node type, see Nokogiri::XML::Node#element? (Node.html#method-i-element-3F)

ENTITY_DECL

Entity declaration type

ENTITY_NODE

Entity node type

ENTITY_REF_NODE

Entity reference node type

HTML_DOCUMENT_NODE

HTML (../HTML.html) document node type, see **Nokogiri::XML::Node#html? (Node.html#method-i-html-3F)**

NAMESPACE_DECL

Namespace (Namespace.html) declaration type

NOTATION_NODE

Notation (Notation.html) node type

PI_NODE

PI node type

TEXT_NODE

Text (Text.html) node type, see **Nokogiri::XML::Node#text? (Node.html#method-i-text-3F)**

XINCLUDE_END

XInclude end type

XINCLUDE_START

XInclude start type

Public Class Methods

new(p1, p2, *args) **Show Source (#)**

Create a new node with `name` sharing GC lifecycle with `document`

Public Instance Methods

%(path, ns = document.root ? document.root.namespaces : {})

/(*paths)

<<(node_or_tags) **Show Source (#)**

Add `node_or_tags` as a child of this **Node (Node.html)**. `node_or_tags` can be a **Nokogiri::XML::Node (Node.html)**, a ::DocumentFragment, a ::NodeSet, or a string containing markup.

Returns self, to support chaining of calls (e.g., root << child1 << child2)

Also see related method **`add_child` (Node.html#method-i-add_child)**.

<=>(other) **Show Source (#)**

Compare two **Node (Node.html)** objects with respect to their **Document (Document.html)**. Nodes from different documents cannot be compared.

==(other) **Show Source (#)**

Test to see if this **Node (Node.html)** is equal to `other`

>(selector) **Show Source (#)**

Search this node's immediate children using **CSS (../CSS.html)** selector `selector`

[](name) **Show Source (#)**

Get the attribute value for the attribute `name`

[]=(name, value) **Show Source (#)**

Set the attribute value for the attribute `name` to `value`

accept(visitor) **Show Source (#)**

Accept a visitor. This method calls "visit" on `visitor` with self.

add_child(node_or_tags) **Show Source (#)**

Add `node_or_tags` as a child of this **Node (Node.html)**. `node_or_tags` can be a **Nokogiri::XML::Node (Node.html)**, a ::DocumentFragment, a ::NodeSet, or a string containing markup.

Returns the reparented node (if `node_or_tags` is a **Node (Node.html)**), or **NodeSet (NodeSet.html)** (if `node_or_tags` is a **DocumentFragment (DocumentFragment.html)**, **NodeSet (NodeSet.html)**, or string).

Also see related method +<<+.

add_namespace(p1, p2)

add_namespace_definition(p1, p2) **Show Source (#)**

Adds a namespace definition with `prefix` using `href` value. The result is as if parsed **XML (../XML.html)** for this node had included an attribute 'xmlns:prefix=value'. A default namespace for this node ("xmlns=") can be added by passing 'nil' for prefix. Namespaces added this way will not show up in **attributes (Node.html#method-i-attributes)**, but they will be included as an xmlns attribute when the node is serialized to **XML (../XML.html)**.

add_next_sibling(node_or_tags) **Show Source (#)**

Insert `node_or_tags` after this **Node (Node.html)** (as a sibling). `node_or_tags` can be a **Nokogiri::XML::Node (Node.html)**, a ::DocumentFragment, a ::NodeSet, or a string containing markup.

Returns the reparented node (if `node_or_tags` is a **Node (Node.html)**), or **NodeSet (NodeSet.html)** (if `node_or_tags` is a **DocumentFragment (DocumentFragment.html)**, **NodeSet (NodeSet.html)**, or string).

Also see related method `after`.

add_previous_sibling(node_or_tags) **Show Source (#)**

Insert `node_or_tags` before this **Node (Node.html)** (as a sibling). `node_or_tags` can be a **Nokogiri::XML::Node (Node.html)**, a ::DocumentFragment, a ::NodeSet, or a string containing markup.

Returns the reparented node (if `node_or_tags` is a **Node (Node.html)**), or **NodeSet (NodeSet.html)** (if `node_or_tags` is a **DocumentFragment (DocumentFragment.html)**, **NodeSet (NodeSet.html)**, or string).

Also see related method `before`.

after(node_or_tags) **Show Source (#)**

Insert `node_or_tags` after this node (as a sibling). `node_or_tags` can be a **Nokogiri::XML::Node (Node.html)**, a **Nokogiri::XML::DocumentFragment (DocumentFragment.html)**, or a string containing markup.

Returns self, to support chaining of calls.

Also see related method **`add_next_sibling` (Node.html#method-i-add_next_sibling)**.

ancestors(selector = nil) **Show Source (#)**

Get a list of ancestor **Node (Node.html)** for this **Node (Node.html)**. If `selector` is given, the ancestors must match `selector`

at(path, ns = document.root ? document.root.namespaces : {}) **Show Source (#)**

Search for the first occurrence of `path`.

Returns nil if nothing is found, otherwise a **Node (Node.html)**.

at_css(*rules) **Show Source (#)**

Search this node for the first occurrence of **CSS (../CSS.html)** `rules`. Equivalent to `css(rules).first` See **Node#css (Node.html#method-i-css)** for more information.

at_xpath(*paths) **Show Source (#)**

Search this node for the first occurrence of **XPath (XPath.html)** `paths`. Equivalent to `xpath(paths).first` See **Node#xpath (Node.html#method-i-xpath)** for more information.

attr(name)

attribute(p1) **Show Source (#)**

Get the attribute node with `name`

attribute_nodes() **Show Source (#)**

returns a list containing the **Node (Node.html)** attributes.

attribute_with_ns(p1, p2) **Show Source (#)**

Get the attribute node with `name` and `namespace`

attributes() **Show Source (#)**

Returns a hash containing the node's attributes. The key is the attribute name without any namespace, the value is a **Nokogiri::XML::Attr (Attr.html)** representing the attribute. If you need to distinguish attributes with the same name, with different namespaces use **attribute_nodes (Node.html#method-i-attribute_nodes)** instead.

before(node_or_tags) **Show Source (#)**

Insert `node_or_tags` before this node (as a sibling). `node_or_tags` can be a **Nokogiri::XML::Node (Node.html)**, a ::DocumentFragment, a ::NodeSet, or a string containing markup.

Returns self, to support chaining of calls.

Also see related method **`add_previous_sibling` (Node.html#method-i-add_previous_sibling)**.

blank?() **Show Source (#)**

Is this node blank?

cdata?() **Show Source (#)**

Returns true if this is a **CDATA (CDATA.html)**

child() **Show Source (#)**

Returns the child node

children() **Show Source (#)**

Get the list of children for this node as a **NodeSet (NodeSet.html)**

children=(node_or_tags) **Show Source (#)**

Set the inner html for this **Node (Node.html)** `node_or_tags` `node_or_tags` can be a **Nokogiri::XML::Node (Node.html)**, a **Nokogiri::XML::DocumentFragment (DocumentFragment.html)**, or a string containing markup.

Returns the reparented node (if `node_or_tags` is a **Node (Node.html)**), or **NodeSet (NodeSet.html)** (if `node_or_tags` is a **DocumentFragment (DocumentFragment.html)**, **NodeSet (NodeSet.html)**, or string).

Also see related method **`inner_html` (Node.html#method-i-inner_html)**=

clone(p1 = v1)

comment?() **Show Source (#)**

Returns true if this is a **Comment (Comment.html)**

content() **Show Source (#)**

Returns the content for this **Node (Node.html)**

content=(string) **Show Source (#)**

Set the Node's content to a **Text (Text.html)** node containing `string`. The string gets **XML (../XML.html)** escaped, not interpreted as markup.

create_external_subset(p1, p2, p3) **Show Source (#)**

Create an external subset

create_internal_subset(p1, p2, p3) **Show Source (#)**

Create the internal subset of a document.

view source (#viewSource)print (#printSource)? (#about)

```
doc.create_internal_subset("chapter", "-//OASIS//DTD DocBook XML//EN", "chapter.dtd")
# => <!DOCTYPE chapter PUBLIC "-//OASIS//DTD DocBook XML//EN" "chapter.dtd">

doc.create_internal_subset("chapter", nil, "chapter.dtd")
# => <!DOCTYPE chapter SYSTEM "chapter.dtd">
```

css(*rules) **Show Source (#)**

Search this node for **CSS (../CSS.html)** `rules`. `rules` must be one or more **CSS (../CSS.html)** selectors. For example:

view source (#viewSource)print (#printSource)? (#about)

```
node.css('title')
node.css('body h1.bold')
node.css('div + p.green', 'div#one')
```

A hash of namespace bindings may be appended. For example:

view source (#viewSource)print (#printSource)? (#about)

```
node.css('bike|tire', {'bike' => 'http://schwinn.com/ (http://schwinn.com/)'})
```

Custom **CSS (../CSS.html)** pseudo classes may also be defined. To define custom pseudo classes, create a class and implement the custom pseudo class you want defined. The first argument to the method will be the current matching **NodeSet (NodeSet.html)**. Any other arguments are ones that you pass in. For example:

view source (#viewSource)print (#printSource)? (#about)

```
node.css('title:regex("w+")', Class.new {
    def regex node_set, regex
        node_set.find_all { |node| node['some_attribute'] =~ /#{regex}/ }
    end
}.new)
```

Note that the **CSS (../CSS.html)** query string is case-sensitive with regards to your document type. That is, if you're looking for "H1" in an **HTML (../HTML.html)** document, you'll never find anything, since **HTML (../HTML.html)** tags will match only lowercase **CSS (../CSS.html)** queries. However, "H1" might be found in an **XML (../XML.html)** document, where tags names are case-sensitive (e.g., "H1" is distinct from "h1").

css_path() **Show Source (#)**

Get the path to this node as a **CSS (../CSS.html)** expression

decorate!() **Show Source (#)**

Decorate this node with the decorators set up in this node's **Document (Document.html)**

default_namespace=(url) **Show Source (#)**

Adds a default namespace supplied as a string `url` href, to self. The consequence is as an xmlns attribute with supplied argument were present in parsed **XML (../XML.html)**. A default namespace set with this method will now show up in **attributes (Node.html#method-i-attributes)**, but when this node is serialized to **XML (../XML.html)** an "xmlns" attribute will appear. See also **namespace (Node.html#method-i-namespace)** and **namespace= (Node.html#method-i-namespace-3D)**

delete(name)

description() **Show Source (#)**

Fetch the **Nokogiri::HTML::ElementDescription (../HTML/ElementDescription.html)** for this node. Returns nil on **XML (../XML.html)** documents and on unknown tags.

do_xinclude(options = XML::ParseOptions::DEFAULT_XML, &block) **Show Source (#)**

Do xinclude substitution on the subtree below node. If given a block, a **Nokogiri::XML::ParseOptions (ParseOptions.html)** object initialized from `options`, will be passed to it, allowing more convenient modification of the parser options.

document() **Show Source (#)**

Get the document for this Node (Node.html)

dup(p1 = v1) Show Source (#)

Copy this node. An optional depth may be passed in, but it defaults to a deep copy. 0 is a shallow copy, 1 is a deep copy.

each() Show Source (#)

Iterate over each attribute name and value pair for this Node (Node.html).

elem?()

element?() Show Source (#)

Returns true if this is an Element (Element.html) node

element_children() Show Source (#)

Get the list of children for this node as a NodeSet (NodeSet.html). All nodes will be element nodes.

Example:

view source (#viewSource)print (#printSource)? (#about)

```
1.@doc.root.element_children.all? { |x| x.element? } # => true
```

elements()

encode_special_chars(p1) Show Source (#)

Encode any special characters in `string`

external_subset() Show Source (#)

Get the external subset

first_element_child() Show Source (#)

Returns the first child node of this node that is an element.

Example:

view source (#viewSource)print (#printSource)? (#about)

```
1.@doc.root.first_element_child.element? # => true
```

fragment(tags) Show Source (#)

Create a DocumentFragment (DocumentFragment.html) containing `tags` that is relative to *this* context node.

fragment?() Show Source (#)

Returns true if this is a DocumentFragment (DocumentFragment.html)

get_attribute(name)

has_attribute?(p1)

html?() Show Source (#)

Returns true if this is an HTML::Document (../HTML/Document.html) node

inner_html(*args) Show Source (#)

Get the inner_html (Node.html#method-i-inner_html) for this node's Node#children (Node.html#method-i-children)

inner_html=(node_or_tags) Show Source (#)

Set the inner html for this Node (Node.html) to `node_or_tags` `node_or_tags` can be a Nokogiri::XML::Node (Node.html), a Nokogiri::XML::DocumentFragment (DocumentFragment.html), or a string containing markup.

Returns self.

Also see related method `children=`

inner_text()

internal_subset() Show Source (#)

Get the internal subset

key?(p1) Show Source (#)

Returns true if `attribute` is set

keys() Show Source (#)

Get the attribute names for this Node (Node.html).

last_element_child() Show Source (#)

Returns the last child node of this node that is an element.

Example:

view source (#viewSource)print (#printSource)? (#about)

```
1.@doc.root.last_element_child.element? # => true
```

line() Show Source (#)

Returns the line for this Node (Node.html) matches?(selector) **Show Source (#)**

Returns true if this **Node (Node.html)** matches `selector`

name()

name=(p1)

namespace() **Show Source (#)**

returns the default namespace set on this node (as with an “xmlns=” attribute), as a **Namespace (Namespace.html)** object.

namespace=(ns) **Show Source (#)**

Set the default namespace on this node (as would be defined with an “xmlns=” attribute in **XML (../XML.html)** source), as a **Namespace (Namespace.html)** object `ns`. Note that a **Namespace (Namespace.html)** added this way will NOT be serialized as an xmlns attribute for this node. You probably want **default_namespace= (Node.html#method-i-default_namespace-3D)** instead, or perhaps **add_namespace_definition (Node.html#method-i-add_namespace_definition)** with a nil prefix argument.

namespace_definitions() **Show Source (#)**

returns namespaces defined on self element directly, as an array of **Namespace (Namespace.html)** objects. Includes both a default namespace (as in“xmlns=”), and prefixed namespaces (as in “xmlns:prefix=”).

namespace_scopes() **Show Source (#)**

returns namespaces in scope for self – those defined on self element directly or any ancestor node – as an array of **Namespace (Namespace.html)** objects. Default namespaces (“xmlns=” style) for self are included in this array; Default namespaces for ancestors, however, are not. See also **namespaces (Node.html#method-i-namespaces)**

namespaced_key?(p1, p2) **Show Source (#)**

Returns true if `attribute` is set with `namespace`

namespaces() **Show Source (#)**

Returns a Hash of {prefix => value} for all namespaces on this node and its ancestors.

This method returns the same namespaces as **namespace_scopes (Node.html#method-i-namespace_scopes)**.

Returns namespaces in scope for self – those defined on self element directly or any ancestor node – as a Hash of attribute-name/value pairs. Note that the keys in this hash **XML (../XML.html)** attributes that would be used to define this namespace, such as “xmlns:prefix”, not just the prefix.

Default namespace set on self will be included with key “xmlns”. However, default namespaces set on ancestor will NOT be, even if self has no explicit default namespace.

next()

next_element() **Show Source (#)**

Returns the next **Nokogiri::XML::Element (Element.html)** type sibling node.

next_sibling() **Show Source (#)**

Returns the next sibling node

node_name() **Show Source (#)**

Returns the name for this **Node (Node.html)**

node_name=(p1) **Show Source (#)**

Set the name for this **Node (Node.html)**

node_type() **Show Source (#)**

Get the type for this **Node (Node.html)**

parent() **Show Source (#)**

Get the parent **Node (Node.html)** for this **Node (Node.html)**

parent=(parent_node) **Show Source (#)**

Set the parent **Node (Node.html)** for this **Node (Node.html)**

parse(string_or_io, options = nil) **Show Source (#)**

Parse `string_or_io` as a document fragment within the context of **this** node. Returns a **XML::NodeSet (NodeSet.html)** containing the nodes parsed from `string_or_io`.

path() **Show Source (#)**

Returns the path associated with this **Node (Node.html)**

pointer_id() **Show Source (#)**

Get the internal pointer number

previous()

previous=(node_or_tags)

previous_element() **Show Source (#)**

Returns the previous **Nokogiri::XML::Element (Element.html)** type sibling node.

previous_sibling() **Show Source (#)**

Returns the previous sibling node

read_only?() **Show Source (#)**

Is this a read only node?

remove()

remove_attribute(name) **Show Source (#)**

Remove the attribute named `name`

replace(node_or_tags) **Show Source (#)**

Replace this **Node (Node.html)** with `node_or_tags`. `node_or_tags` can be a **Nokogiri::XML::Node (Node.html)**, a ::DocumentFragment, a ::NodeSet, or a string containing markup.

Returns the reparented node (if `node_or_tags` is a **Node (Node.html)**), or **NodeSet (NodeSet.html)** (if `node_or_tags` is a **DocumentFragment (DocumentFragment.html)**, **NodeSet (NodeSet.html)**, or string).

Also see related method `swap`.

search(*paths) **Show Source (#)**

Search this node for `paths`. `paths` can be **XPath (XPath.html)** or **CSS (../CSS.html)**, and an optional hash of namespaces may be appended. See **Node#xpath (Node.html#method-i-xpath)** and **Node#css (Node.html#method-i-css)**.

serialize(*args, &block) **Show Source (#)**

Serialize **Node (Node.html)** using `options`. Save options can also be set using a block. See **SaveOptions (Node/SaveOptions.html)**.

These two statements are equivalent:

view source (#viewSource)print (#printSource)? (#about)

```
node.serialize(:encoding => 'UTF-8', :save_with => FORMAT | AS_XML)
```

or

view source (#viewSource)print (#printSource)? (#about)

```
node.serialize(:encoding => 'UTF-8') do |config|
    config.format.as_xml
end
```

set_attribute(name, value)

swap(node_or_tags) **Show Source (#)**

Swap this **Node (Node.html)** for `node_or_tags` `node_or_tags` can be a **Nokogiri::XML::Node (Node.html)**, a ::DocumentFragment, a ::NodeSet, or a string containing markup.

Returns self, to support chaining of calls.

Also see related method `replace`.

text()

text?() **Show Source (#)**

Returns true if this is a **Text (Text.html)** node

to_html(options = {}) **Show Source (#)**

Serialize this **Node (Node.html)** to **HTML (../HTML.html)**

view source (#viewSource)print (#printSource)? (#about)

```
1.doc.to_html
```

See **Node#write_to (Node.html#method-i-write_to)** for a list of `options`. For formatted output, use **Node#to_xhtml (Node.html#method-i-to_xhtml)** instead.

to_s() **Show Source (#)**

Turn this node in to a string. If the document is **HTML (../HTML.html)**, this method returns html. If the document is **XML (../XML.html)**, this method returns **XML (../XML.html)**.

to_str()

to_xhtml(options = {}) **Show Source (#)**

Serialize this **Node (Node.html)** to XHTML using `options`

view source (#viewSource)print (#printSource)? (#about)

```
1.doc.to_xhtml(:indent => 5, :encoding => 'UTF-8')
```

See **Node#write_to (Node.html#method-i-write_to)** for a list of `options`

to_xml(options = {}) **Show Source (#)**

Serialize this **Node (Node.html)** to **XML (../XML.html)** using `options`

view source (#viewSource)print (#printSource)? (#about)

```
1.doc.to_xml(:indent => 5, :encoding => 'UTF-8')
```

See **Node#write_to (Node.html#method-i-write_to)** for a list of `options` traverse(&block) **Show Source (#)**

Yields self and all children to `block` recursively.

type()

unlink() **Show Source (#)**

Unlink this node from its current context.

values() **Show Source (#)**

Get the attribute values for this **Node (Node.html)**.

write_html_to(io, options = {}) **Show Source (#)**

Write **Node (Node.html)** as **HTML (../HTML.html)** to `io` with `options`

See **Node#write_to (Node.html#method-i-write_to)** for a list of `options`

write_to(io, *options) **Show Source (#)**

Write **Node (Node.html)** to `io` with `options`. `options` modify the output of this method. Valid options are:

- `:encoding` for changing the encoding
- `:indent_text` the indentation text, defaults to one space
- `:indent` the number of `:indent_text` to use, defaults to 2
- `:save_with` a combination of **SaveOptions (Node/SaveOptions.html)** constants.

To save with UTF-8 indented twice:

view source (#viewSource)print (#printSource)? (#about)

```
1.node.write_to(io, :encoding => 'UTF-8', :indent => 2)
```

To save indented with two dashes:

view source (#viewSource)print (#printSource)? (#about)

```
1.node.write_to(io, :indent_text => '-', :indent => 2
```

write_xhtml_to(io, options = {}) **Show Source (#)**

Write **Node (Node.html)** as XHTML to `io` with `options`

See **Node#write_to (Node.html#method-i-write_to)** for a list of `options`

write_xml_to(io, options = {}) **Show Source (#)**

Write **Node (Node.html)** as **XML (../XML.html)** to `io` with `options`

view source (#viewSource)print (#printSource)? (#about)

```
doc.write_xml_to io, :encoding => 'UTF-8'
```

See **Node#write_to (Node.html#method-i-write_to)** for a list of options

xml?() **Show Source (#)**

Returns true if this is an **XML::Document (Document.html)** node

xpath(*paths) **Show Source (#)**

Search this node for **XPath (XPath.html)** `paths`. `paths` must be one or more **XPath (XPath.html)** queries.

view source (#viewSource)print (#printSource)? (#about)

```
node.xpath('.//title')
```

A hash of namespace bindings may be appended. For example:

view source (#viewSource)print (#printSource)? (#about)

```
node.xpath('.//foo:name', {'foo' => 'http://example.org/ (http://example.org/)'})
node.xpath('.//xmlns:name', node.root.namespaces)
```

A hash of variable bindings may also be appended to the namespace bindings. For example:

view source (#viewSource)print (#printSource)? (#about)

```
node.xpath('.//address[@domestic=$value]', nil, {:value => 'Yes'})
```

Custom **XPath (XPath.html)** functions may also be defined. To define custom functions create a class and implement the function you want to define. The first argument to the method will be the current matching **NodeSet (NodeSet.html)**. Any other arguments are ones that you pass in. Note that this class may appear anywhere in the argument list. For example:

view source (#viewSource)print (#printSource)? (#about)

```
node.xpath('.//title[regex(., "w+")]', Class.new {
    def regex node_set, regex
        node_set.find_all { |node| node['some_attribute'] =~ /#{regex}/ }
    end
}.new)
```

XML is like violence — if it doesn’t solve your problems, you are not using enough of it