

# Package ‘XiMpLe’

October 12, 2022

**Type** Package

**Title** A Simple XML Tree Parser and Generator

**Description** Provides a simple XML tree parser/generator. It includes functions to read XML files into R objects, get information out of and into nodes, and write R objects back to XML code. It's not as powerful as the 'XML' package and doesn't aim to be, but for simple XML handling it could be useful. It was originally developed for the R GUI and IDE 'RKward' <<https://rkward.kde.org>>, to make plugin development easier.

**Depends** R (>= 2.9.0)

**Imports** methods

**Suggests** testthat, knitr, rmarkdown

**VignetteBuilder** knitr

**URL** <https://reaktanz.de/?c=hacking&s=XiMpLe>

**BugReports** <https://github.com/rkward-community/XiMpLe/issues>

**License** GPL (>= 3)

**Encoding** UTF-8

**LazyLoad** yes

**Version** 0.10-3

**Date** 2020-09-19

**RoxygenNote** 7.1.1

**Collate** '00\_class\_01\_XiMpLe.node.R' '00\_class\_02\_XiMpLe.doc.R'  
'00\_class\_03\_XiMpLe.validity.R' '01\_method\_01\_pasteXML.R'  
'XiMpLe-internal.R' '01\_method\_02\_node.R' '01\_method\_03\_show.R'  
'01\_method\_04\_validXML.R' '01\_method\_05\_XMLgenerators.R'  
'XMLNode.R' 'XMLTree.R' 'XMLValidity.R' 'XiMpLe-package.R'  
'parseXMLTree.R' 'pasteXMLTag.R' 'zzz\_is\_get\_utils.R'

**NeedsCompilation** no

**Author** Meik Michalke [aut, cre]

**Maintainer** Meik Michalke <[meik.michalke@hhu.de](mailto:meik.michalke@hhu.de)>

**Repository** CRAN

**Date/Publication** 2020-09-19 17:20:02 UTC

## R topics documented:

XiMpLe-package . . . . .	2
node . . . . .	3
parseXMLTree . . . . .	4
pasteXML . . . . .	5
pasteXMLTag . . . . .	7
show, XiMpLe.XML-method . . . . .	8
validXML . . . . .	9
XiMpLe.doc,-class . . . . .	11
XiMpLe.node,-class . . . . .	12
XiMpLe.validity,-class . . . . .	13
XMLgenerators . . . . .	14
XMLName . . . . .	16
XMLNode . . . . .	20
XMLTree . . . . .	21
XMLValidity . . . . .	22

**Index** 24

---

XiMpLe-package	<i>A Simple XML Tree Parser and Generator</i>
----------------	---

---

## Description

Provides a simple XML tree parser/generator. It includes functions to read XML files into R objects, get information out of and into nodes, and write R objects back to XML code. It's not as powerful as the 'XML' package and doesn't aim to be, but for simple XML handling it could be useful. It was originally developed for the R GUI and IDE 'RKWard' <<https://rkwart.kde.org>>, to make plugin development easier.

## Details

The DESCRIPTION file:

```

Package: XiMpLe
Type: Package
Version: 0.10-3
Date: 2020-09-19
Depends: R (>= 2.9.0)
Encoding: UTF-8
License: GPL (>= 3)
LazyLoad: yes
URL: https://reaktanx.de/?c=hacking&s=XiMpLe

```

**Author(s)**

NA

Maintainer: NA

**See Also**

Useful links:

- <https://reaktanz.de/?c=hacking&s=XiMpLe>
- Report bugs at <https://github.com/rkward-community/XiMpLe/issues>

---

`node`*Extract/manipulate a node or parts of it from an XML tree*

---

**Description**

This method can be used to get parts of a parsed XML tree object, or to fill it with new values.

`XiMpLe.XML` is a class union for objects of classes `XiMpLe.node` and `XiMpLe.doc`.

**Usage**

```
node(  
  obj,  
  node = list(),  
  what = NULL,  
  cond.attr = NULL,  
  cond.value = NULL,  
  element = NULL  
)
```

```
## S4 method for signature 'XiMpLe.XML'
```

```
node(  
  obj,  
  node = list(),  
  what = NULL,  
  cond.attr = NULL,  
  cond.value = NULL,  
  element = NULL  
)
```

```
node(  
  obj,  
  node = list(),  
  what = NULL,  
  cond.attr = NULL,  
  cond.value = NULL,
```

```

    element = NULL
  ) <- value

## S4 replacement method for signature 'XiMplE.XML'
node(
  obj,
  node = list(),
  what = NULL,
  cond.attr = NULL,
  cond.value = NULL,
  element = NULL
) <- value

```

### Arguments

obj	An object of class <code>XiMplE.doc</code> or <code>XiMplE.node</code> .
node	A list of node names (or their numeric values), where each element is the child of its previous element. duplicate matches will be returned as a list.
what	A character string, must be a valid slot name of class <code>XiMplE.node</code> , like "attributes" or "value". If not NULL, only that part of a node will be returned. There's also two special properties for this option: <code>what="@path"</code> will not return the node or it's contents, but a character string with the "path" to it in the object; <code>what="obj@path"</code> is the same but won't have <code>obj</code> substituted with the object's name.
cond.attr	A named character string, to further filter the returned results. If not NULL, only nodes with fully matching attributes will be considered.
cond.value	A character string, similar to <code>cond.attr</code> , but is matched against the value between a pair of tags.
element	A character string naming one list element of the node slot. If NULL, all elements will be returned.
value	The value to set.

### Examples

```

## Not run:
node(my.xml.tree, node=list("html","body"), what="attributes")
node(my.xml.tree, node=list("html","head","title"), what="value") <- "foobar"

## End(Not run)

```

---

parseXMLTree

*Read an XML file into an R object*

---

### Description

Read an XML file into an R object

**Usage**

```
parseXMLTree(file, drop = NULL, object = FALSE)
```

**Arguments**

file	Character string, valid path to the XML file which should be parsed.
drop	Character vector with the possible values "comments", "cdata" "declarations" and "doctype", defining element classes to be dropped from the resulting object.
object	Logical, if TRUE, file will not be treated as a path name but as a character vector to be parsed as XML directly.

**Value**

An object of class [XiMpLe.doc](#) with four slots:

file: Full path to the parsed file, or "object" if object=TRUE.

xml: XML declaration, if found.

dtd: Doctype definition, if found.

children: A list of objects of class [XiMpLe.node](#), with the elements "name" (the node name), "attributes" (list of attributes, if found), "children" (list of [XiMpLe.node](#) object, if found) and "value" (text value between a pair of start/end tags, if found).

**See Also**

[XiMpLe.node](#), [XiMpLe.doc](#)

**Examples**

```
## Not run:  
sample.XML.object <- parseXMLTree("~/data/sample_file.xml")  
  
## End(Not run)
```

**Description**

These methods can be used to paste objects of class [XiMpLe.node](#) or [XiMpLe.doc](#).

**Usage**

```

pasteXML(obj, ...)

## S4 method for signature 'XiMPLe.node'
pasteXML(
  obj,
  level = 1,
  shine = 1,
  indent.by = "\t",
  tidy = TRUE,
  tidy.omit = c("![CDATA[", "![CDATA[")
)

## S4 method for signature 'XiMPLe.doc'
pasteXML(
  obj,
  shine = 1,
  indent.by = "\t",
  tidy = TRUE,
  tidy.omit = c("![CDATA[", "![CDATA[")
)

```

**Arguments**

obj	An object of class <code>XiMPLe.node</code> or <code>XiMPLe.doc</code> .
...	Additional options for the generic method, see options for a specific method, respectively.
level	Indentation level.
shine	Integer, controlling if the output should be formatted for better readability. Possible values: <b>0</b> No formatting. <b>1</b> Nodes will be indented. <b>2</b> Nodes will be indented and each attribute gets a new line.
indent.by	A character string defining how indentation should be done. Defaults to tab.
tidy	Logical, if TRUE the special characters "<" and ">" will be replaced with the entities "&lt;" and "&gt;" in attributes and text values.
tidy.omit	A character vector with node names that should be excluded from tidy.

**Note**

The functions `pasteXMLNode()` and `pasteXMLTree()` have been replaced by the `pasteXML` methods. They should no longer be used.

**See Also**

[XiMPLe.node](#), [XiMPLe.doc](#)

---

 pasteXMLTag

 Write an XML tag
 

---

### Description

Creates a whole XML tag with attributes and, if it is a pair of start and end tags, also one object as child. This can be used recursively to create whole XML tree structures with this one function.

### Usage

```
pasteXMLTag(
  tag,
  attr = NULL,
  child = NULL,
  empty = TRUE,
  level = 1,
  allow.empty = FALSE,
  rename = NULL,
  shine = 2,
  indent.by = "\t",
  tidy = TRUE
)
```

### Arguments

tag	Character string, name of the XML tag.
attr	A list of attributes for the tag.
child	If empty=FALSE, a character string to be pasted as a child node between start and end tag.
empty	Logical, <true /> or <false></false>
level	Indentation level.
allow.empty	Logical, if FALSE, tags without attributes will not be returned.
rename	An optional named list if the attributes in XML need to be renamed from their list names in attr. This list must in turn have a list element named after tag, containing named character elements, where the names represent the element names in attr and their values the names the XML attribute should get.
shine	Integer, controlling if the output should be formatted for better readability. Possible values: <ul style="list-style-type: none"> <li><b>0</b> No formatting.</li> <li><b>1</b> Nodes will be indented.</li> <li><b>2</b> Nodes will be indented and each attribute gets a new line.</li> </ul>
indent.by	A character string defining how indentation should be done. Defaults to tab.
tidy	Logical, if TRUE the special characters "<", ">" and "&" will be replaced with the entities "&lt;", "&gt;" and "&amp;" in attribute values. For comment or CDATA tags, if the text includes newline characters they will also be indented.

**Value**

A character string.

**Note**

However, you will probably not want to use this function at all, as it is much more comfortable to create XML nodes or even nested trees with [XMLNode](#) and [XMLTree](#), and then feed the result to [pasteXML](#).

**See Also**

[XMLNode](#), [XMLTree](#), [pasteXML](#)

**Examples**

```
sample.XML.tag <- pasteXMLTag("a",
  attr=list(href="http://example.com", target="_blank"),
  child="klick here!",
  empty=FALSE)
```

---

show, XiMpLe.XML-method

*Show method for S4 objects of XiMpLe XML classes*

---

**Description**

Used to display objects of class [XiMpLe.doc](#) and [XiMpLe.node](#)

**Usage**

```
## S4 method for signature 'XiMpLe.XML'
show(object)
```

**Arguments**

object            An object of class [XiMpLe.doc](#) or [XiMpLe.node](#)

**See Also**

[XiMpLe.doc](#) [XiMpLe.node](#)



---

 validXML

 Validate S4 objects of *XiMPLe XML classes*


---

### Description

Checks whether objects of class `XiMPLe.doc` or `XiMPLe.node` have valid child nodes.

### Usage

```
validXML(
  obj,
  validity = XMLValidity(),
  parent = NULL,
  children = TRUE,
  attributes = TRUE,
  warn = FALSE,
  section = parent,
  caseSens = TRUE
)

## S4 method for signature 'XiMPLe.XML'
validXML(
  obj,
  validity = XMLValidity(),
  parent = NULL,
  children = TRUE,
  attributes = TRUE,
  warn = FALSE,
  section = parent,
  caseSens = TRUE
)
```

### Arguments

<code>obj</code>	An object of class <code>XiMPLe.doc</code> or <code>XiMPLe.node</code> . If <code>parent=NULL</code> , this object will be checked for validity, including its child nodes. If <code>parent</code> is either a character string or another <code>XiMPLe</code> node, it will be checked whether <code>obj</code> is a valid child node of <code>parent</code> .
<code>validity</code>	An object of class <code>XiMPLe.validity</code> , see <code>XMLValidity</code> .
<code>parent</code>	Either a character string (name of the parent node) or a <code>XiMPLe</code> node, whose name will be used as name of the parent node.
<code>children</code>	Logical, whether child node names should be checked for validity.
<code>attributes</code>	Logical, whether attributes should be checked for validity.
<code>warn</code>	Logical, whether invalid objects should cause a warning or stop with an error.

section	Either a character string (name of the section) or a XiMpLe node, whose name will be used as name of the XML section this check refers to. This is only relevant for warnings and error messages, in case you want to use something different than the actual parent node name.
caseSens	Logical, whether checks should be case sensitive or not.

### Details

XiMpLe can't handle DOM specifications yet, but this method can be used to construct validation schemes.

### Value

Returns TRUE if tests pass, and depending on the setting of warn either FALSE or an error if a test fails.

### Note

: If no parent is specified, obj will be checked recursively.

### See Also

[validXML](#), [XMLValidity](#), [XiMpLe.doc](#), and [XiMpLe.node](#)

### Examples

```
HTMLish <- XMLValidity(
  children=list(
    body=c("a", "p", "ol", "ul", "strong"),
    head=c("title"),
    html=c("head", "body"),
    li=c("a", "br", "strong"),
    ol=c("li"),
    p=c("a", "br", "ol", "ul", "strong"),
    ul=c("li")
  ),
  attrs=list(
    a=c("href", "name"),
    p=c("align")
  ),
  allChildren=c("!--"),
  allAttrs=c("id", "class"),
  empty=c("br")
)
# make XML object
validChildNodes <- XMLNode("html",
  XMLNode("head",
    XMLNode("!--", "comment always passes"),
    XMLNode("title", "test")
  ),
  XMLNode("body",
```

```

        XMLNode("p",
          XMLNode("a", "my link"),
          XMLNode("br"),
          "text goes on"
        )
      )
    )
  invalidChildNodes <- XMLNode("html",
    XMLNode("head",
      XMLNode("title",
        XMLNode("body", "test")
      )
    )
  )
)

# do validity checks
# the first should pass
validXML(
  validChildNodes,
  validity=HTMLish
)

# now this one should cause a warning and return FALSE
validXML(
  invalidChildNodes,
  validity=HTMLish,
  warn=TRUE
)

```

---

XiMple.doc,-class      *Class XiMple.doc*


---

## Description

This class is used for objects that are returned by [parseXMLTree](#).

## Usage

```
is.XiMple.doc(x)
```

## Arguments

x                    An arbitrary R object.

## Slots

file    Character string, Name of the file.

xml    Either a named list of character values (attributes for the XML declaration of the file), or a list of XiMple.nodes with tags whose names must start with a "?".

`dtd` A named list, attributes for the doctype definition of the file.

`children` A list of objects of class `XiMPLe.node`, representing the DOM structure of the XML document.

---

`XiMPLe.node,-class`      *Class XiMPLe.node*

---

### Description

This class is used to create DOM trees of XML documents, like objects that are returned by [parseXMLTree](#).

### Usage

```
is.XiMPLe.node(x)
```

### Arguments

`x`                      An arbitrary R object.

### Details

There are certain special values predefined for the `name` slot to easily create special XML elements:

`name=""` If the name is an empty character string, a pseudo node is created, [pasteXMLNode](#) will paste its value as plain text.

`name="!--"` Creates a comment tag, i.e., this will comment out all its children.

`name="![CDATA["` Creates a CDATA section and places all its children in it.

`name="*![CDATA["` Creates a CDATA section and places all its children in it, where the CDATA markers are commented out by `/* */`, as is used for JavaScript in XHTML.

### Slots

`name` Name of the node (i.e., the XML tag identifier). For special names see details.

`attributes` A list of named character values, representing the attributes of this node.

`children` A list of further objects of class `XiMPLe.node`, representing child nodes of this node.

`value` Plain text to be used as the enclosed value of this node. Set to `value=""` if you want a childless node to be forced into an non-empty pair of start and end tags by [pasteXMLNode](#).

---

XiMple.validity,-class

*Class XiMple.validity*


---

### Description

Used for objects that describe valid child nodes and attributes of XiMple.nodes.

### Usage

```
is.XiMple.validity(x)
```

### Arguments

x                    An arbitrary R object.

### Details

You should use [XMLValidity](#) to create objects of this class.

### Slots

children Named list of vectors or XiMple.validity objects. The element name defines the parent node name and each character string a valid child node name. If a value is in turn of class XiMple.validity, this object will be used for recursive validation of deeper nodes.

attrs Named list of character vectors. The element name defines the parent node name and each character string a valid attribute name.

allChildren Character vector, names of globally valid child nodes for all nodes, if any.

allAttrs Character vector, names of globally valid attributes for all nodes, if any.

empty Character vector, names of nodes that must be empty nodes (i.e., no closing tag), if any.

ignore Character vector, names of nodes that should be ignored, if any.

### See Also

[XMLValidity](#), [validXML](#)

### Examples

```
HTMLish <- XMLValidity(
  children=list(
    body=c("a", "p", "ol", "ul", "strong"),
    head=c("title"),
    html=c("head", "body"),
    li=c("a", "br", "strong"),
    ol=c("li"),
    p=c("a", "br", "ol", "ul", "strong"),
    ul=c("li")
```

```

),
  attrs=list(
    a=c("href", "name"),
    p=c("align")
  ),
  allChildren=c("!--"),
  allAttrs=c("id", "class"),
  empty=c("br")
)

# this example uses recursion: the <b> node can have the "foo"
# attribute only below an <a> node; the <d> node is also only valid
# in an <a> node
XMLRecursion <- XMLValidity(
  children=list(
    a=XMLValidity(
      children=list(
        b=c("c")
      ),
      attrs=list(
        b=c("foo")
      ),
      allChildren=c("d")
    )
  ),
  attrs=list(
    b=c("bar")
  )
)

```

---

XMLgenerators

*Generate XML generator functions from XiMPLe.validity object*


---

## Description

Takes an object of class `XiMPLe.validity` and turns it into a character vector of generator functions for each XML node that was defined.

## Usage

```

XMLgenerators(
  validity,
  prefix = "XML",
  checkValidity = TRUE,
  indent.by = "\t",
  roxygenDocs = FALSE,
  valParam = "validity",
  replaceChar = "_",
  dir = NULL,

```

```

    overwrite = FALSE,
    oneFile = NULL
)

## S4 method for signature 'XiMpLe.validity'
XMLgenerators(
  validity,
  prefix = "XML",
  checkValidity = TRUE,
  indent.by = "\t",
  roxygenDocs = FALSE,
  valParam = "validity",
  replaceChar = "_",
  dir = NULL,
  overwrite = FALSE,
  oneFile = NULL
)

```

### Arguments

<code>validity</code>	An object of class <code>XiMpLe.validity</code> .
<code>prefix</code>	A character string to be used as a prefix for the resulting function names.
<code>checkValidity</code>	Logical, whether all functions should include a check for valid XML.
<code>indent.by</code>	A character string defining how indentation should be done.
<code>roxygenDocs</code>	Logical, whether a skeleton for roxygen2-ish documentation should be added.
<code>valParam</code>	A character string, name of the additional parameter to use for validation if <code>checkValidity=TRUE</code> .
<code>replaceChar</code>	A (single) character to be used as a replacement for invalid characters for R parameter names.
<code>dir</code>	A character string, path to write files to. If <code>dir=NULL</code> , no files are being written, but the results returned in form of a character vector. If <code>dir</code> is set and the directory does not yet exist, it will be created.
<code>overwrite</code>	Logical, whether existing files should be replaced when <code>dir</code> is set.
<code>oneFile</code>	A character string. If set, all functions are to be documented in one single *.Rd file, named like the string.

### Details

The resulting code follows these rules:

- Each child node gets its own argument, except if there is only one valid child node. It will use the dots element instead then.
- Each attribute will also get its own argument.
- If `checkValidity=TRUE`, one extra argument named after the value of `valParam` will be added.
- All arguments are set to `NULL` by default.

- Only the main level of "allAttrs" will be taken into account, there's no recursion for this slot.

### Value

If dir=NULL a named vector of character strings. Otherwise one or more files are written to the location specified via dir.

### See Also

[XMLValidity](#) and [XiMpLe.validity](#)

### Examples

```
HTMLish <- XMLValidity(
  children=list(
    body=c("a", "p", "ol", "ul", "strong"),
    head=c("title"),
    html=c("head", "body"),
    li=c("a", "br", "strong"),
    ol=c("li"),
    p=c("a", "br", "ol", "ul", "strong"),
    ul=c("li")
  ),
  attrs=list(
    a=c("href", "name"),
    p=c("align")
  ),
  allChildren=c("!--"),
  allAttrs=c("id", "class"),
  empty=c("br")
)
XMLgenerators(HTMLish)
```

---

XMLName

*Getter/setter methods for S4 objects of XiMpLe XML classes*

---

### Description

Used to get/set certain slots from objects of class [XiMpLe.doc](#) and [XiMpLe.node](#).

### Usage

```
XMLName(obj)
```

```
## S4 method for signature 'XiMpLe.node'
XMLName(obj)
```

```
XMLName(obj) <- value
```



```
## S4 replacement method for signature 'XiMpLe.node'  
XMLName(obj) <- value  
  
XMLAttrs(obj)  
  
## S4 method for signature 'XiMpLe.node'  
XMLAttrs(obj)  
  
XMLAttrs(obj) <- value  
  
## S4 replacement method for signature 'XiMpLe.node'  
XMLAttrs(obj) <- value  
  
XMLChildren(obj)  
  
## S4 method for signature 'XiMpLe.node'  
XMLChildren(obj)  
  
## S4 method for signature 'XiMpLe.doc'  
XMLChildren(obj)  
  
XMLChildren(obj) <- value  
  
## S4 replacement method for signature 'XiMpLe.node'  
XMLChildren(obj) <- value  
  
## S4 replacement method for signature 'XiMpLe.doc'  
XMLChildren(obj) <- value  
  
XMLValue(obj)  
  
## S4 method for signature 'XiMpLe.node'  
XMLValue(obj)  
  
XMLValue(obj) <- value  
  
## S4 replacement method for signature 'XiMpLe.node'  
XMLValue(obj) <- value  
  
XMLFile(obj)  
  
## S4 method for signature 'XiMpLe.doc'  
XMLFile(obj)  
  
XMLFile(obj) <- value  
  
## S4 replacement method for signature 'XiMpLe.doc'
```

```
XMLFile(obj) <- value

XMLDecl(obj)

## S4 method for signature 'XiMpLe.doc'
XMLDecl(obj)

XMLDecl(obj) <- value

## S4 replacement method for signature 'XiMpLe.doc'
XMLDecl(obj) <- value

XMLDTD(obj)

## S4 method for signature 'XiMpLe.doc'
XMLDTD(obj)

XMLDTD(obj) <- value

## S4 replacement method for signature 'XiMpLe.doc'
XMLDTD(obj) <- value

XMLScan(obj, name, as.list = FALSE)

## S4 method for signature 'XiMpLe.node'
XMLScan(obj, name, as.list = FALSE)

## S4 method for signature 'XiMpLe.doc'
XMLScan(obj, name, as.list = FALSE)

XMLScan(obj, name) <- value

## S4 replacement method for signature 'XiMpLe.node'
XMLScan(obj, name) <- value

## S4 replacement method for signature 'XiMpLe.doc'
XMLScan(obj, name) <- value

XMLScanDeep(obj, find = NULL, search = "attributes")

## S4 method for signature 'XiMpLe.node'
XMLScanDeep(obj, find = NULL, search = "attributes")

## S4 method for signature 'XiMpLe.doc'
XMLScanDeep(obj, find = NULL, search = "attributes")
```

### Arguments

obj                    An object of class XiMpLe.node or XiMpLe.doc

value	The new value to set.
name	Character, name of nodes to scan for.
as.list	Logical, if TRUE always returns a list (or NULL), otherwise if exactly one result is found, it will be returned as as single <code>XiMpLe.node</code> .
find	Character, name of element to scan for.
search	Character, name of the slot to scan, one of "attributes", "name", or "value" for nodes.

## Details

These are convenience methods to get or set slots from XML objects without using the @ operator.

- `XMLName()`: get/set the XML node name (slot name of class `XiMpLe.node`)
- `XMLAttrs()`: get/set the XML node attributes (slot attrs of class `XiMpLe.node`)
- `XMLValue()`: get/set the XML node value (slot value of class `XiMpLe.node`)
- `XMLChildren()`: get/set the XML child nodes (slot children of both classes `XiMpLe.node` and `XiMpLe.doc`)
- `XMLFile()`: get/set the XML document file name (slot file of class `XiMpLe.doc`)
- `XMLDecl()`: get/set the XML document declaration (slot xml of class `XiMpLe.doc`)
- `XMLDTD()`: get/set the XML document doctype definition (slot dtd of class `XiMpLe.doc`)

Another special method can scan a node/document tree object for appearances of nodes with a particular name:

- `XMLScan(obj, name, as.list=FALSE)`: get/set the XML nodes by name (recursively searches slot name of both classes `XiMpLe.node` and `XiMpLe.doc`). If `as.list=TRUE` always returns a list (or NULL), otherwise if exactly one result is found, it will be returned as as single `XiMpLe.node`.

Finally, there is a method to scan for certain values in `XiMpLe` objects and just list them. For instance, it can be used to list all instances of a certain attribute type in a document tree:

- `XMLScanDeep(obj, find, search="attributes")`: returns all found instances of `find` in all slots defined by `search`.

## See Also

[node](#), [XiMpLe.doc](#), [XiMpLe.node](#)

## Examples

```
xmlTestNode <- XMLNode("foo", XMLNode("testchild"))
XMLName(xmlTestNode) # returns "foo"
XMLName(xmlTestNode) <- "bar"
XMLName(xmlTestNode) # now returns "bar"

# search for a child node
XMLScan(xmlTestNode, "testchild")
# remove nodes of that name
XMLScan(xmlTestNode, "testchild") <- NULL
```

XMLNode

*Constructor function for XiMpLe.node objects***Description**

Can be used to create XML nodes.

**Usage**

```
XMLNode(
  name,
  ...,
  attrs = NULL,
  namespace = "",
  namespaceDefinitions = NULL,
  .children = list(...)
)
```

**Arguments**

name	Character string, the tag name.
...	Optional children for the tag. Must be either objects of class <code>XiMpLe.node</code> or character strings, which are treated as simple text values. If this is empty, the tag will be treated as an empty tag. To force a closing tag, supply an empty string, i.e. <code>""</code> .
attrs	An optional named list of attributes.
namespace	Currently ignored.
namespaceDefinitions	Currently ignored.
.children	Alternative way of specifying children, if you have them already as a list.

**Details**

To generate a CDATA node, set `name="![CDATA["`, to create a comment, set `name="!--"`.

**Value**

An object of class `XiMpLe.node`.

**See Also**

[XMLTree](#), [pasteXML](#)

**Examples**

```
sample.XML.node <- XMLNode("a",
  attrs=list(href="http://example.com", target="_blank"),
  .children="klick here!")
```

XMLTree

*Constructor function for XiMPLe.doc objects***Description**

Can be used to create full XML trees.

**Usage**

```
XMLTree(..., xml = NULL, dtd = NULL, .children = list(...))
```

**Arguments**

...	Optional children for the XML tree. Must be either objects of class <a href="#">XiMPLe.node</a> or character strings, which are treated as simple text values.
xml	A named list, XML declaration of the XML tree. Currently just pasted, no checking is done.
dtd	A named list, doctype definition of the XML tree. Valid elements are doctype (root element), decl ("PUBLIC" or "SYSTEM"), id (the identifier) and refer (URI to .dtd). Currently just pasted, no checking is done.
.children	Alternative way of specifying children, if you have them already as a list.

**Value**

An object of class [XiMPLe.doc](#)

**See Also**

[XMLNode](#), [pasteXML](#)

**Examples**

```
sample.XML.a <- XMLNode("a",
  attrs=list(href="http://example.com", target="_blank"),
  .children="klick here!")
sample.XML.body <- XMLNode("body", .children=list(sample.XML.a))
sample.XML.html <- XMLNode("html", .children=list(XMLNode("head", ""),
  sample.XML.body))
sample.XML.tree <- XMLTree(sample.XML.html,
  xml=list(version="1.0", encoding="UTF-8"),
  dtd=list(doctype="html", decl="PUBLIC",
  id="//W3C//DTD XHTML 1.0 Transitional//EN",
  refer="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"))
```

XMLValidity

*Constructor function for XiMpLe.validity objects***Description**

Create validity definitions for XiMpLe nodes, to be used by [validXML](#).

**Usage**

```
XMLValidity(
  children = NULL,
  attrs = NULL,
  allChildren = NULL,
  allAttrs = NULL,
  empty = NULL,
  ignore = NULL
)
```

**Arguments**

children	Named list of vectors or XiMpLe.validity objects. The element name defines the parent node name and each character string a valid child node name. If a value is in turn of class XiMpLe.validity, this object will be used for recursive validation of deeper nodes.
attrs	Named list of character vectors. The element name defines the parent node name and each character string a valid attribute name.
allChildren	Character vector, names of globally valid child nodes for all nodes, if any.
allAttrs	Character vector, names of globally valid attributes for all nodes, if any.
empty	Character vector, names of nodes that must be empty nodes (i.e., no closing tag), if any.
ignore	Character vector, names of nodes that should be ignored, if any.

**Value**

An object of class [XiMpLe.validity](#)

**See Also**

[validXML](#)

**Examples**

```
HTMLish <- XMLValidity(
  children=list(
    body=c("a", "p", "ol", "ul", "strong"),
    head=c("title"),
```

```
    html=c("head", "body"),
    li=c("a", "br", "strong"),
    ol=c("li"),
    p=c("a", "br", "ol", "ul", "strong"),
    ul=c("li")
  ),
  attrs=list(
    a=c("href", "name"),
    p=c("align")
  ),
  allChildren=c("!--"),
  allAttrs=c("id", "class"),
  empty=c("br")
)
```

# Index

## \* classes

XiMple.doc, -class, [11](#)  
XiMple.node, -class, [12](#)  
XiMple.validity, -class, [13](#)

## \* methods

pasteXML, [5](#)  
show, XiMple.XML-method, [8](#)  
validXML, [9](#)  
XMLgenerators, [14](#)  
XMLName, [16](#)

is.XiMple.doc (XiMple.doc, -class), [11](#)  
is.XiMple.node (XiMple.node, -class), [12](#)  
is.XiMple.validity  
(XiMple.validity, -class), [13](#)

node, [3](#), [19](#)

node, -methods (node), [3](#)  
node, XiMple.doc-method (node), [3](#)  
node, XiMple.node-method (node), [3](#)  
node, XiMple.XML-method (node), [3](#)  
node<- (node), [3](#)  
node<- , -methods (node), [3](#)  
node<- , XiMple.doc-method (node), [3](#)  
node<- , XiMple.node-method (node), [3](#)  
node<- , XiMple.XML-method (node), [3](#)

parseXMLTree, [4](#), [11](#), [12](#)

pasteXML, [5](#), [8](#), [20](#), [21](#)  
pasteXML, -methods (pasteXML), [5](#)  
pasteXML, XiMple.doc-method (pasteXML), [5](#)  
pasteXML, XiMple.node-method (pasteXML),  
[5](#)  
pasteXMLNode, [12](#)  
pasteXMLNode (pasteXML), [5](#)  
pasteXMLTag, [7](#)  
pasteXMLTree (pasteXML), [5](#)

show, -methods (show, XiMple.XML-method),  
[8](#)

show, XiMple.doc-method  
(show, XiMple.XML-method), [8](#)  
show, XiMple.node-method  
(show, XiMple.XML-method), [8](#)  
show, XiMple.XML-method, [8](#)

validXML, [9](#), [10](#), [13](#), [22](#)  
validXML, -methods (validXML), [9](#)  
validXML, XiMple.doc-method (validXML), [9](#)  
validXML, XiMple.node-method (validXML),  
[9](#)  
validXML, XiMple.XML-method (validXML), [9](#)

XiMple (XiMple-package), [2](#)  
XiMple-package, [2](#)  
XiMple.doc, [4-6](#), [8-10](#), [16](#), [19](#), [21](#)  
XiMple.doc, -class, [11](#)  
XiMple.doc-class (XiMple.doc, -class), [11](#)  
XiMple.node, [4-6](#), [8-10](#), [16](#), [19-21](#)  
XiMple.node, -class, [12](#)  
XiMple.node-class (XiMple.node, -class),  
[12](#)  
XiMple.validity, [9](#), [14](#), [16](#), [22](#)  
XiMple.validity, -class, [13](#)  
XiMple.validity-class  
(XiMple.validity, -class), [13](#)

XiMple.XML-class (node), [3](#)  
XMLAttrs (XMLName), [16](#)  
XMLAttrs, -methods (XMLName), [16](#)  
XMLAttrs, XiMple.node-method (XMLName),  
[16](#)  
XMLAttrs<- (XMLName), [16](#)  
XMLAttrs<- , -methods (XMLName), [16](#)  
XMLAttrs<- , XiMple.node-method  
(XMLName), [16](#)  
XMLChildren (XMLName), [16](#)  
XMLChildren, -methods (XMLName), [16](#)  
XMLChildren, XiMple.doc-method  
(XMLName), [16](#)



- XMLChildren, `XiMplE.node-method`  
(XMLName), 16
- XMLChildren<- (XMLName), 16
- XMLChildren<-,-methods (XMLName), 16
- XMLChildren<- ,`XiMplE.doc-method`  
(XMLName), 16
- XMLChildren<- ,`XiMplE.node-method`  
(XMLName), 16
- XMLDecl (XMLName), 16
- XMLDecl,-methods (XMLName), 16
- XMLDecl,`XiMplE.doc-method` (XMLName), 16
- XMLDecl<- (XMLName), 16
- XMLDecl<-,-methods (XMLName), 16
- XMLDecl<- ,`XiMplE.doc-method` (XMLName),  
16
- XMLDTD (XMLName), 16
- XMLDTD,-methods (XMLName), 16
- XMLDTD,`XiMplE.doc-method` (XMLName), 16
- XMLDTD<- (XMLName), 16
- XMLDTD<-,-methods (XMLName), 16
- XMLDTD<- ,`XiMplE.doc-method` (XMLName), 16
- XMLFile (XMLName), 16
- XMLFile,-methods (XMLName), 16
- XMLFile,`XiMplE.doc-method` (XMLName), 16
- XMLFile<- (XMLName), 16
- XMLFile<-,-methods (XMLName), 16
- XMLFile<- ,`XiMplE.doc-method` (XMLName),  
16
- XMLgenerators, 14
- XMLgenerators,-methods (XMLgenerators),  
14
- XMLgenerators,`XiMplE.validity-method`  
(XMLgenerators), 14
- XMLName, 16
- XMLName,-methods (XMLName), 16
- XMLName,`XiMplE.node-method` (XMLName), 16
- XMLName<- (XMLName), 16
- XMLName<-,-methods (XMLName), 16
- XMLName<- ,`XiMplE.node-method` (XMLName),  
16
- XMLNode, 8, 20, 21
- XMLScan (XMLName), 16
- XMLScan,-methods (XMLName), 16
- XMLScan,`XiMplE.doc-method` (XMLName), 16
- XMLScan,`XiMplE.node-method` (XMLName), 16
- XMLScan<- (XMLName), 16
- XMLScan<-,-methods (XMLName), 16
- XMLScan<- ,`XiMplE.doc-method` (XMLName),  
16
- XMLScan<- ,`XiMplE.node-method` (XMLName),  
16
- XMLScanDeep (XMLName), 16
- XMLScanDeep,-methods (XMLName), 16
- XMLScanDeep,`XiMplE.doc-method`  
(XMLName), 16
- XMLScanDeep,`XiMplE.node-method`  
(XMLName), 16
- XMLTree, 8, 20, 21
- XMLValidity, 9, 10, 13, 16, 22
- XMLValue (XMLName), 16
- XMLValue,-methods (XMLName), 16
- XMLValue,`XiMplE.node-method` (XMLName),  
16
- XMLValue<- (XMLName), 16
- XMLValue<-,-methods (XMLName), 16
- XMLValue<- ,`XiMplE.node-method`  
(XMLName), 16