

# R documentation

of all in 'pkg/SSOAP/man'

March 26, 2007

## R topics documented:

convertFromSOAP	2
duration-class	2
fromSOAP	3
genSOAPClientInterface	5
getNodeById	6
getReturnNode	7
getSOAPType	8
isHTTPError	9
parseSOAP	10
processWSDL	11
SOAPClientInterface-class	12
SOAPTypes	13
SOAPFault	14
SOAPHandlers	15
SOAP.logical	16
SOAPNameSpaces	17
.SOAP	18
SOAPResult-class	20
SOAPResult	21
SOAPServer-class	22
SOAPServerDescription-class	23
SOAPServerDescription	24
SOAPServer	25
SOAPType-class	27
toSOAP	28
writeSOAPBody	29
writeTypes	30
WSDLMethod-class	31
.XMLRPC	32
xmlRPCResult	33

<b>Index</b>	<b>35</b>
--------------	-----------

`convertFromSOAP`      *Convert SOAP result to S object*

---

### Description

This generic function and its methods provide facilities for converting data from a SOAP XML structure to a “value” object in R.

### Usage

```
convertFromSOAP(val, type)
```

### Arguments

`val`                    the XML object representing the data.  
`type`                    the target “type” of object to which the XML should be converted.

### Value

An R object of the type identified by `type`.

### References

<http://www.w3.org/TR/SOAP/> <http://www.omegahat.org/SSOAP>, <http://www.omegahat.org/bugs>.

### See Also

[.SOAP](#)

### Examples

---

`duration-class`      *Classes for built-in SOAP data types*

---

### Description

XML Schema and SOAP uses several built-in types which are extensions or restrictions of the basic R type. These classes represent these XML Schema/SOAP types in R. Precise definitions for the possible values of the data types of each class can be found in the SOAP specification at <http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/>. More informal and intuitive descriptions can be found in tutorials on the Web such as [http://www.w3schools.com/schema/schema\\_dtypes\\_string.asp](http://www.w3schools.com/schema/schema_dtypes_string.asp).

The `token` type is essentially a string but with white space cleaned up, i.e. multiple spaces reduced to a single space, new lines and control feeds eliminated, as are tabs, and not leading or trailing white space.

A duration is given as a string providing the number of years, months, days and potentially hours, minutes and seconds.

## Objects from the Class

### Slots

A token is merely a character vector with a pre-processing step that converts its contents to a token.

The duration is not fully implemented at this point. It will resemble the POSIXt class with fields in a list for the years, months, days, hours, minutes and seconds.

### Extends

token: Class "`character`", from data part. Class "`vector`", by class "character", distance 2.

duration: extends `data.frame` and has six fields/columns named years, months, days, hours, minutes, seconds.

### Methods

No methods defined with class "duration" in the signature.

### Author(s)

Duncan Temple Lang

### References

<http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/>

### See Also

`processWSDL` `SSOAP:::.SOAPDefaultTypes`

### Examples

---

fromSOAP

*Convert an XML node to an S Object*

---

### Description

Convert an XML node representing a SOAP value to an S object. The different functions handle different types of SOAP objects. This also works on an XML string or filename. The `fromSOAPArray` supports the `offset` and `position` attributes for partially transmitted and sparse arrays. Currently, there are limits on the

**Usage**

```

fromSOAP(node, root = NULL, converters = SOAPPrimitiveConverters,
         append = TRUE, type = NULL, findReturn = TRUE, multiRefs = list())
fromSOAPStruct(node, root = NULL,
              converters = SOAPPrimitiveConverters, type = NULL, multiRefs = list())
fromSOAPArray(node, type = NULL, root = NULL,
              converters = SOAPPrimitiveConverters,
              multiRefs = list(), simplify = TRUE)

```

**Arguments**

node	the XML node (and sub-nodes) giving the SOAP content. The <code>fromSOAP</code> function works recursively, operating on sub-nodes, etc. This argument is the individual node.
root	the top-most node of the XML “document” which is used for resolving <code>href</code> references within elements to other nodes. This doesn’t change across the recursive calls.
converters	a named-list of functions. The names are used to find the appropriate converter for a non-array or non-struct SOAP object. The name comes from the value of the “ <code>xsi:type</code> ” attribute or from the string “ <code>xsd:&lt;element name&gt;</code> ” where <code>element-name</code> is the name of the XML node. This will become more advanced in the future.
append	a logical value indicating whether to merge the converters with the default <code>SOAPPrimitiveConverters</code> , or use them as is. This is available for convenience so that the caller doesn’t have to perform the two-step merging herself.
type	an optional type identifier that if not specified is computed from the <code>xsi:type</code> attribute of the XML node ( <code>node</code> ) if available.
findReturn	a logical value indicating whether to recursively descend the XML node to find the <code>&lt;return&gt;</code> element and process that rather than working directly from <code>node</code> .
multiRefs	?
simplify	a logical value that, if <code>TRUE</code> , causes <code>fromSOAPArray</code> to try to convert the list of elements into a vector. It does this only if the element type of the SOAP array is “primitive”, defined currently as being in the names of the <code>SOAPPrimitiveConverters</code> list. This is similar to the simplification step <code>sapply</code> performs after calling <code>lapply</code> .

**Details****Value****Author(s)**

Duncan Temple Lang <duncan@wald.ucdavis.edu>

**References**

<http://www.w3.org/TR/SOAP/> <http://www.omegahat.org/SSOAP>, <http://www.omegahat.org/bugs>.

**See Also**

[.SOAP](#)

**Examples**


---

```
genSOAPClientInterface
```

*Create R functions to access SOAP server methods*

---

**Description**

This function creates function definitions, etc. that provide access to the methods described in the SOAP server description details.

**Usage**

```
genSOAPClientInterface(operations = def@operations[[1]], def, name = def@name,
                       env = new.env(), where = globalenv(), server = def@server,
                       nameSpaces = def@nameSpaces, verbose = FALSE)
```

**Arguments**

<code>operations</code>	a list of the descriptions of the server's methods. Each method description provides information about the parameters and the return value.
<code>def</code>	the <code>SOAPServerDescription-class</code> object.
<code>name</code>	currently unused
<code>env</code>	an environment object. This is used ?
<code>where</code>	the location (usually in the search path) where new S4 classes will be defined to represent the complex return types. This can be any value that is acceptable for the <code>where</code> argument of <code>setClass</code> , i.e. an integer, a package name ("package:name") or, explicitly, an environment.
<code>server</code>	an object which will be used as the server in the SOAP calls. This provides the user with a mechanism to provide an alternative server object such as one which contains a password or which already has a connection to the SOAP server, or controls the connection in different ways.
<code>nameSpaces</code>	a character vector that identifies the namespace-URI mappings used for calls to this server. This maps the namespace abbreviations to the actual URIs. This can be a named character vector of these mappings, or alternatively a simple character string that identifies the name of the element in the <code>.SOAPDefaultNameSpaces</code> list. And if we don't know the collection of namespaces, we use <code>NA</code> to indicate that we shall determine this later.
<code>verbose</code>	a logical indicating whether information about the processing should be displayed on the console, as it occurs.

**Details****Value****Author(s)**

Duncan Temple Lang <duncan@wald.ucdavis.edu>

**References**

<http://www.w3.org/TR/SOAP/> <http://www.omegahat.org/SSOAP>, <http://www.omegahat.org/bugs>.

**See Also**

[processWSDL](#)

**Examples**

```
kegg = processWSDL("http://soap.genome.jp/KEGG.wsdl")
  # note that we force the use of the 1.1 name spaces to get arrays
  # handled correctly on the server side.
iface = genSOAPClientInterface(def = kegg, nameSpaces = "1.1")

tmp = processWSDL(system.file("examples", "KEGG.wsdl", package = "SSOAP"))
ff = genSOAPClientInterface(tmp@operations[[1]], def = tmp, tmp@name, verbose=FALSE)

as.character.Definition = function(x) structure(x[["definition"]], names = x[["entry_id"]])
as.character.list = function(x) sapply(x, as.character)

o = ff@functions$list_organisms()

as.character.list(o)
```

---

getNodeById

*Find XML node by 'id' attribute*

---

**Description**

This function basically finds the top-level node within a SOAP XML tree that has the specified `id` attribute value. This is used to find/resolve references to other parts of the message

**Usage**

```
getNodeById(id, root)
```

**Arguments**

<code>id</code>	the name of the node, usually identified by a <code>href</code> in another node.
<code>root</code>	the top-level XML node, i.e. the root node of the XML tree

**Value**

An XML node.

**Author(s)**

Duncan Temple Lang <duncan@wald.ucdavis.edu>

**References**

<http://www.w3.org/TR/SOAP/> <http://www.omegahat.org/SSOAP>, <http://www.omegahat.org/bugs>.

**See Also**

[fromSOAP](#)

**Examples**

---

getReturnNode	<i>Get XML node from SOAP response</i>
---------------	----------------------------------------

---

**Description**

Parse the XML content from the SOAP response and traverse the tree to find the node in the Body element associated with the result of the request. It looks for a node named This attempts to be helpful by taking input in various forms, i.e. text of the body of the HTTP response, header and body in a SOAPHTTPReply object returned from [.SOAP](#), or the root node of a previously parsed XML tree.

**Usage**

```
getReturnNode (node)
```

**Arguments**

node	either an XML node that was obtained from parsing the text of the reply or the SOAPHTTPReply object returned from the <a href="#">.SOAP</a> call which contains the header and body of the HTTP request, or alternatively this can be the text content from the body of the HTTP response.
------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Value**

An XMLNode object.

**Author(s)**

Duncan Temple Lang

**See Also**

[.SOAP xmlTreeParse](#)

## Examples

---

getSOAPType	<i>Compute the SOAP type identifier for an S object</i>
-------------	---------------------------------------------------------

---

## Description

This determines the SOAP type for the given S object so that it can be used in an XML element. It returns a collection of name-value pairs (as a named character vector) which can be used as XML attributes for the element defining the value.

## Usage

```
getSOAPType(obj, value = NULL)
```

## Arguments

obj	the S object whose SOAP type information is to be determined
value	for arrays, this specifies the type of the elements of that array.

## Details

This consults SOAPTypes to find the types.

## Value

A named character vector giving the XML attribute name and value pairs.

## Author(s)

Duncan Temple Lang <duncan@wald.ucdavis.edu>

## References

<http://www.w3.org/TR/SOAP/> <http://www.omegahat.org/SSOAP>, <http://www.omegahat.org/bugs>.

## See Also

[writeSOAPMessage](#)

## Examples

---

isHTTPError	<i>Determines if an error occurred in an HTTP communication</i>
-------------	-----------------------------------------------------------------

---

### Description

This examines the HTTP header information (computed via the `curlPerform` call that implements the HTTP communication with the SOAP server) and determines if there was an error reported from the server.

### Usage

```
isHTTPError(response)
```

### Arguments

response	the header information computed by collecting the header lines from the SOAP server's HTTP response. This is currently done via the call to <code>curlPerform</code> in the <code>.SOAP</code> function.
----------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

### Details

This just looks at the `status` entry and compares it to the value 200.

The `curlPerform` is capable of performing redirections, etc. to handle manageable errors. See the options for that function.

### Value

A logical value indicating whether there was an error (TRUE) or not (FALSE).

### Author(s)

Duncan Temple Lang <duncan@wald.ucdavis.edu>

### References

<http://www.w3.org/TR/SOAP/> <http://www.omegahat.org/SSOAP>, <http://www.omegahat.org/bugs>.

### See Also

`.SOAP curlPerform`

### Examples

---

`parseSOAP`*Parse XML message*

---

**Description**

This is a convenience function to parse the SOAP message returned from a server and retrieve the payload of the message as an XML tree in S.

**Usage**

```
parseSOAP(xmlSource, header = FALSE, reduce = TRUE, ...,
          fullNamespaceInfo = TRUE)
```

**Arguments**

<code>xmlSource</code>	an string giving the name of a file or URL containing the XML content, or a string containing the XML content itself.
<code>header</code>	ignored
<code>reduce</code>	a logical value indicating whether to return the top-most XML node of the Body or to return the entire XML tree read from <code>xmlSource</code> . If this is true, we find the Body node and return its first child.
<code>...</code>	arguments that are passed directly to <a href="#">xmlTreeParse</a> without being processed by this function.
<code>fullNamespaceInfo</code>	a logical value that is passed on to <a href="#">xmlTreeParse</a> when parsing the SOAP response This controls how the namespace information on each XML node is represented, with TRUE causing the prefix and URI to be included rather than just the prefix.

**Value**

An XMLNode object.

**Author(s)**

Duncan Temple Lang <duncan@wald.ucdavis.edu>

**References**

<http://www.w3.org/TR/SOAP/> <http://www.omegahat.org/SSOAP>, <http://www.omegahat.org/bugs>.

**See Also**

[fromSOAP](#)

**Examples**

---

processWSDL

*Read and process a Web Service Description Language file*

---

## Description

This reads and converts a WSDL file for a server into a collection of functions and methods.

## Usage

```
processWSDL(fileName = "KEGG.wsdl", handlers =  
            WSDLParseHandlers(fileName), namespaces = character())
```

## Arguments

fileName	the name of the WSDL file or URI.
handlers	a list of handler functions that are passed to <code>xmlTreeParse</code> to perform transformations on the XML nodes as they are parsed and converted to R objects. The default handlers drop comments and take care of importing files that are referenced via <code>&lt;wsdl:import&gt;</code> nodes.
namespaces	a character vector that identifies the namespace-URI mappings used for calls to this server. This maps the namespace abbreviations to the actual URIs. This can be a named character vector of these mappings, or alternatively a simple character string that identifies the name of the element in the <code>.SOAPDefaultNameSpaces</code> list. And if we don't know the collection of namespaces, we use <code>NA</code> to indicate that we shall determine this later.

## Details

## Value

An object of class `SOAPServerDescription`.

## Author(s)

Duncan Temple Lang <duncan@wald.ucdavis.edu>

## See Also

`UseDashInSOAPNames` is an R option that can be set by the user that is understood by this package to control whether to either leave SOAP method names as-is, or if `FALSE`, to remove `_` in the names and capitalize the first character in all but the first word of the name. In other words, if `UseDashInSOAPNames` is set to `FALSE`, the name `abc_def_ghi` is mapped to `abcDefGhi`. By default, the value is unset and treated as `TRUE`, so dashes are preserved.

**Examples**

```

tmp = processWSDL(system.file("examples", "KEGG.wsdl", package = "SSOAP"))

# The first set of operations, and the method "color_pathway_by_objects"
o = tmp@operations[[1]][["color_pathway_by_objects"]]
names(o@parameters)
o@parameters[["fg_color_list"]]
o@returnValue

ff = genSOAPClientInterface(tmp@operations[[1]], def = tmp, tmp@name, verbose=FALSE)

# ff$functions$get_all_neighbors_by_gene(kid="eco:b0002", threshold= as.integer(500), org

## Not run:
x = ff@functions$get_paralogs_by_gene("eco:b0002", 1, 10)
## End(Not run)

tp = get(".operation", environment(ff@functions$get_paralogs_by_gene))@returnValue

# A different WSDL file.
tmp = processWSDL(system.file("examples", "XMethodsFilesystemService.wsdl.xml", package

```

---

SOAPClientInterface-class

*Representation of machine-generated interface to SOAP methods and classes*

---

**Description**

This is a simple class that combines a list of functions and a list of classes that are machine-generated to provide an R interface to a SOAP server. It is just a mechanism for combining the code.

**Objects from the Class**

We use the function `genSOAPClientInterface` to create instances of this class. Objects can be created by calls of the form `new("SOAPClientInterface", ...)`.

**Slots**

**functions:** Object of class "list". A list of the functions that are proxies to the SOAP methods.

**classes:** Object of class "list". A list of classes defined in support of the functions. These correspond to the new data types defined by the SOAP server.

**Methods**

No methods defined with class "SOAPClientInterface" in the signature.

**Author(s)**

Duncan Temple Lang <duncan@wald.ucdavis.edu>

**References**

<http://www.w3.org/TR/SOAP/> <http://www.omegahat.org/SSOAP>, <http://www.omegahat.org/bugs>.

**See Also**

[genSOAPClientInterface](#)

**Examples**


---

SOAPTypes

*Data objects used in SSOAP*

---

**Description**

These are S objects that store default data for the SOAP functions.

SOAPTypes is a list mapping the type of a primitive S object (typically computed via `typeof`) to a SOAP type, typically given as a named character vector. The name-value pair gives an XML attribute name and value which identifies the SOAP type (e.g. `xsi:type = xsd:string`).

SOAPPrimitiveConverters is a list of functions that handle mapping SOAP values to S primitive values. These are indexed by the different primitive SOAP type values, e.g. `xsd:string`, `xsd:boolean`, etc.

`.SOAPDefaultNameSpaces` is a named list in which each element is a named-character vector giving the namespace identifier and URI (e.g. `xsi="http://www.w3.org/2001/XMLSchema-instance"`) that are added to the top-most node of the XML message, i.e. in the `Envelope` node. The names of the list are used to index the different collections of namespaces, making `.SOAPDefaultNameSpaces` act like a catalog of SOAP specifications.

`.SOAPDefaultHandlers` is a collection of named functions that are used to parameterize the way the `.SOAP` functions works. These functions are callbacks that can modify the way the HTTP request and SOAP message are constructed, and how the response is processed.

The idea is that one can change these globally to customize the SSOAP package to local needs.

**Usage**

```
SOAPTypes
SOAPPrimitiveConverters
SOAPNameSpaces
```

**Format****Source**

SSOAP package

## References

<http://www.w3.org/TR/SOAP/> <http://www.omegahat.org/SSOAP>, <http://www.omegahat.org/bugs>.

## Examples

---

SOAPFault

*Create a SOAP Fault object*

---

## Description

This creates an object representing a SOAP fault object returned from a SOAP server. It creates an S4 object of the appropriate class, either one of the built-in SOAP fault classes or a general SOAP fault object.

## Usage

SOAPFault (node)

## Arguments

node                    the top-level XML node from the SOAP response giving the fault information

## Value

An object derived from SOAPFault. Can be one of SOAPVersionMismatchFault, SOAPMustUnderstandFault, SOAPClientFault or SOAPGeneralFault.

## Author(s)

Duncan Temple Lang <duncan@wald.ucdavis.edu>

## References

<http://www.w3.org/TR/SOAP/> <http://www.omegahat.org/SSOAP>, <http://www.omegahat.org/bugs>.

## See Also

[.SOAP](#)

## Examples

---

`SOAPHandlers`*Get SOAP function handlers*

---

### Description

This returns a collection of functions that are used by the `.SOAP` function to control exactly how the HTTP request and SOAP message is created and how the result is processed. Values are merged with the values from `.SOAPDefaultNameSpaces`.

This is a convenient mechanism for specifying the collection of functions to use to parameterize the different aspects of the SOAP mechanism in S.

### Usage

```
SOAPHandlers(..., include = character(0), exclude = character(0))
```

### Arguments

<code>...</code>	name=function pairs giving values to be returned in the list of functions. These override corresponding elements in <code>.SOAPDefaultNameSpaces</code> .
<code>include</code>	a character vector giving the names of the elements to include. This is used to identify (a few) elements that are to be kept from the defaults identified by <code>version</code> .
<code>exclude</code>	a character vector giving the names of the elements to discard. This is usually deployed when we want to keep a large number of elements and it is more convenient to explicitly exclude some.

### Value

A named list of functions. The names correspond to the different elements that are accessed by the `.SOAP` function. Currently, these are

<code>action</code>	convert the user-specified <code>SOAPAction</code> to the target one. By default, this appends <code>#methodName</code> to the user's value. This takes four arguments: the user's action, the name of the method, the SOAP server object and the vector of request-specific namespaces (i.e. the <code>xmlns</code> argument for <code>.SOAP</code> ).
---------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

### Author(s)

Duncan Temple Lang <duncan@wald.ucdavis.edu>

### References

<http://www.w3.org/TR/SOAP/> <http://www.omegahat.org/SSOAP>,

### See Also

[SOAPNameSpaces](#) `.merge`

## Examples

```
SOAPHandlers()  
SOAPHandlers(action = function(action, method, server, xmlns) action)  
  
SOAPHandlers(exclude="action")
```

---

SOAP.logical

*Convert a SOAP logical value to S*

---

## Description

This is a simple function that handles converting a boolean value given in SOAP to an S logical value. This operates on the value within the SOAP node (not the node itself). This is different from [as.logical](#) only in that it handles "0" (i.e. a string).

## Usage

```
SOAP.logical(x)
```

## Arguments

`x` a string, or any S value that is to coerced to a logical value.

## Details

## Value

A logical value.

## Author(s)

Duncan Temple Lang <duncan@wald.ucdavis.edu>

## References

<http://www.w3.org/TR/SOAP/> <http://www.omegahat.org/SSOAP>, <http://www.omegahat.org/bugs>.

## See Also

[fromSOAP](#) [SOAPPrimitiveConverters](#)

## Examples

```
## Not run:  
# Since not exported  
SOAP.logical("0")  
SOAP.logical("TRUE")  
SOAP.logical("FALSE")  
## End(Not run)
```

---

SOAPNameSpaces      *Get SOAP namespace definitions*

---

### Description

This is a convenience function that makes it easy to in-line the specification of the top-level or global SOAP namespaces within a `.SOAP` call. It provides a way to cumulate namespace identifiers and URIs into a named vector by specifying the relevant collection within the “catalog” of SOAP-namespace collections and to augment that collection, override elements and/or include and exclude certain elements by name.

### Usage

```
SOAPNameSpaces(..., include = character(0), exclude = character(0),  
                version = getOption("SSOAP:DefaultNamespace"))
```

### Arguments

<code>...</code>	an arbitrary number of id-URI pairs that define a namespace. These are included in the collection returned from this function along with any values identified via the <code>version</code> argument in the <code>.SOAPDefaultNameSpaces</code> list.
<code>include</code>	a character vector giving the names of the elements to include. This is used to identify (a few) elements that are to be kept from the defaults identified by <code>version</code> .
<code>exclude</code>	a character vector giving the names of the elements to discard. This is usually deployed when we want to keep a large number of elements and it is more convenient to explicitly exclude some.
<code>version</code>	a name that identifies an element in the <code>.SOAPDefaultNameSpaces</code> list that is used to get the default values. If this does not match a name in that list, no defaults are used and only the values in <code>...</code> are used.

### Value

A named vector giving the id-URI pairs of namespaces.

### Author(s)

Duncan Temple Lang <duncan@wald.ucdavis.edu>

### References

<http://www.w3.org/TR/SOAP/> <http://www.omegahat.org/SSOAP>, <http://www.omegahat.org/bugs>.

### See Also

`.merge`

## Examples

```
SOAPNameSpaces ()
SOAPNameSpaces (omegahat="http://www.omegahat.org",
                 r = "http://www.r-project.org")

SOAPNameSpaces (omegahat="http://www.omegahat.org",
                 r = "http://www.r-project.org", include="SOAP-ENV")

SOAPNameSpaces (omegahat="http://www.omegahat.org",
                 r = "http://www.r-project.org", exclude="xsd")

SOAPNameSpaces (omegahat="http://www.omegahat.org",
                 r = "http://www.r-project.org",
                 xsd = "my own XSD URI")
```

---

.SOAP

*Invoke a SOAP method*

---

## Description

This is used to call a SOAP method in a SOAP server, passing the relevant arguments from S and converting the response into an S object. The communication between S and the SOAP server is handled via connections.

## Usage

```
.SOAP(server, method, ..., .soapArgs = list(), action, nameSpaces = SOAPNameSpaces(),
       xmlns = NULL, handlers = SOAPHandlers(), .types = NULL,
       .convert = TRUE, .opts = list(), curlHandle = getCurlHandle(),
       .header = getSOAPRequestHeader(action, .server = server),
       .literal = FALSE)
```

## Arguments

<code>server</code>	a <code>SOAPServer</code> object
<code>method</code>	the name of the SOAP method to invoke
<code>...</code>	<code>name=value</code> arguments to pass to the
<code>.soapArgs</code>	an alternative mechanism for passing arguments to the <code>.SOAP</code> call. This is a list of named or unnamed values which is used as the arguments for the SOAP method invocation.
<code>action</code>	the <code>SOAPAction</code> string to put in the HTTP header. This is required. If it is an object of class <code>AsIs</code> , it is left exactly as it is. This allows one to call this function as <code>.SOAP(..., action = I("einfo"))</code> without having to provide a handler to bypass the default action mechanism.
<code>nameSpaces</code>	a named character vector giving the XML namespaces to add to the Body. These are given as a named character vector with the names giving the local namespace identifier and the value being the URI corresponding to that namespace identifier. For ease of use, one can identify the collections corresponding to the 1999 or 2001 schema using the simpler strings "1.1" and "1.2" respectively. If <code>nameSpaces</code> is a single string, we use it to index the element in the <code>.SOAPDefaultNameSpaces</code> list.

<code>xmlns</code>	the name space to use for the XML nodes which specify the actual method call, i.e. within the BODY. This is either a single string, or a name-value pair given as a character vector. The name is the namespace identifier and the value is the URI.
<code>handlers</code>	a collection of functions that, if present, are called at different points in the SOAP invocation to process the input and output. These can be thought of as event callbacks and include <code>action</code> for creating the final form of the SOAPAction string, <code>converter</code> for processing the XML returned by the SOAP server in the case of a successful invocation, and so on.
<code>.types</code>	[not yet implemented] allows one to explicitly control the conversion of the arguments to the appropriate/desired SOAP type. This is useful when you know what the server is expecting.
<code>.convert</code>	a function, a logical value or a <code>SOAPType</code> . If this is a function, it should take two arguments: the content to be converted from SOAP format to R and the target type described as a <code>SOAPType</code> object. This should return an R object representing the SOAP content. If, alternatively, this is supplied as a logical value, this controls whether the default converters are used ( <code>TRUE</code> ) or not ( <code>FALSE</code> ). These converters are taken from the <code>handlers</code> argument. And finally, if <code>.convert</code> is a <code>SOAPType</code> object, we call <code>convertFromSOAP</code> with the
<code>.opts</code>	a named list of elements that are passed to the <code>curlPerform</code> function which actually invokes the SOAP method. These options control aspects of the HTTP request, including debugging information that is displayed on the console, e.g. <code>.opts = list(verbose = TRUE)</code>
<code>curlHandle</code>	this is passed to <code>curlPerform</code> as the <code>curlHandle</code> argument. By providing this as a parameter here, the user can reuse an existing curl handle with options explicitly set just once. Additionally, one can control the connection to the Web server using keep-alive connections, etc. to improve performance.
<code>.header</code>	a named character vector of elements which are used in the HTTP header for the SOAP request. These are calculated by default within the <code>.SOAP</code> call, but the parameter allows them to be pre-computed and supplied in the call.
<code>.literal</code>	a logical value indicating whether to use the literal encoding for serializing the data being sent to and from the server.

**Details**

**Value**

An S object representing the return value from the SOAP method invocation.

**Author(s)**

Duncan Temple Lang <duncan@wald.ucdavis.edu>

**References**

<http://www.w3.org/TR/SOAP/> <http://www.omegahat.org/SSOAP>, <http://www.omegahat.org/bugs>.

**See Also**

[writeSOAPMessage](#) [isHTTPError](#) [curlPerform](#) [postForm](#)

**Examples**

```
## Not run:
.SOAP(SOAPServer("services.xmethods.net", "soap"),
      "getRate", country1="England", country2 = "Japan",
      action="urn:xmethods-CurrencyExchange")

.SOAP(SOAPServer("services.xmethods.net", "soap/servlet/rpcrouter"),
      "getPrice", "0596000278",
      action="urn:xmethods-BNPriceCheck")

s <- SOAPServer("http://services.xmethods.net/soap")
.SOAP(s,
      "getQuote", "AMZN",
      action="urn:xmethods-delayed-quotes#getQuote")

.SOAP(SOAPServer("services.soaplite.com", "temper.cgi"),
      "c2f", 37.5,
      action="http://www.soaplite.com/Temperatures")

# Different action and namespace.
# Specify handlers=NULL to avoid the additional processing of the
# SOAPAction string, i.e. the appending of the method name.
# This kills off all the handlers. If we want to remove only the
# "action" element, we have to be more explicit.

s1 <- SOAPServer("services.soaplite.com", "interop.cgi")
.SOAP(s1,
      "echoString", "From R",
      action="urn:soapinterop",
      xmlns=c(namesp1="http://soapinterop.org/"),
      handlers =NULL)
## End(Not run)
```

---

SOAPResult-class     *Description of a the result of a SOAP request*

---

**Description**

This class is used to provide access to the result of the SOAP request over HTTP. It separates the result into the header and body/content returned by the HTTP server.

**Objects from the Class**

This is not typically used except within code in the SSOAP package when it is returned from a call to `.SOAP`. Objects can be created by calls of the form `new("SOAPResult", ...)`.

**Slots**

**header:** Object of class "list". A named list of values from the header of the HTTP communication returned by the SOAP server in response to a SOAP request.

**content:** Object of class "character". The body or content of the HTTP request. The header field provides information about how to interpret the content in this field, e.g. the style of encoding.

**Methods**

**convertFromSOAP** `signature(val = "SOAPResult")`: converts the value in the content slot to an R object, using the fields in the header slot to interpret the content appropriately.

**Author(s)**

Duncan Temple Lang <duncan@wald.ucdavis.edu>

**References**

<http://www.w3.org/TR/SOAP/> <http://www.omegahat.org/SSOAP>, <http://www.omegahat.org/bugs>.

**See Also**

[.SOAP](#)

**Examples**

---

SOAPResult

*Create an object to represent the raw result of a SOAP invocation*

---

**Description**

This function creates an object of class `SOAPResult` which is used to represent both the header and the body of the HTTP response from a SOAP invocation. Such an object is created in the [.SOAP](#) function and used to convert the body into an S value.

**Usage**

```
SOAPResult(content, header, obj = new("SOAPResult"))
```

**Arguments**

<code>content</code>	a character vector representing the body of the HTTP response.
<code>header</code>	a named list of name-value pairs giving the details of the header of the HTTP response.
<code>obj</code>	an instance of the class that we are creating, derived from <code>SOAPResult</code> .

**Value**

An object of class SOAPResult.

content	Description of 'comp1'
header	Description of 'comp2'
...	

**Author(s)**

Duncan Temple Lang <duncan@wald.ucdavis.edu>

**References**

<http://www.w3.org/TR/SOAP/> <http://www.omegahat.org/SSOAP>

**See Also**

[.SOAP fromSOAP isHTTPError](#)

**Examples**


---

SOAPServer-class    *Classes for SOAP Server object*

---

**Description**

These classes provide a way to describe the location of a SOAP server. This is separate from the servers actions. Rather, we use this to represent the identity of the server in terms of its Web address, i.e. the machine name or IP address, port number and URI or path within the server. The different classes represent the communication protocol, typically HTTP or HTTPS, i.e. HTTP over SSL, the secure communication protocol.

A DynamicSOAPServer might be better termed a “compiled” server. It contains information about the methods and data types accessible via the server. This information is converted into R classes and functions that can be used to invoke the server’s methods.

DynamicSOAPServer does not extend SOAPServer currently. Rather, it contains a SOAPServer. This is because we need to be able to determine the protocol and so have different types of SOAPServer objects associated with the interface methods. This class hierarchy may change in the future.

**Objects from the Class**

The function `SOAPServer` is used as the general constructor. Alternatively, one can use the `new` syntax, `new("HTTPServer", host = "www.omegahat.org", url = "theServer")`

**Slots**

**host:** Object of class "character". The machine name or IP address of the server.

**port:** Object of class "integer". The port number, if other than the default HTTP port 80.

**url:** Object of class "character". The document within the server that is the location of the server.

## Methods

**\$ signature**(x = "SOAPServer", name = "character"): returns a function that will invoke the server's method whose name is given by name. This is merely syntactic sugar to allow the expression `server$foo(1, 2, 3)` to invoke the method `foo` in the remote server.

**\$ signature**(x = "DynamicSOAPServer", name = "character"): returns a function that will invoke the server's method whose name is given by name. This is merely syntactic sugar to allow the expression `server$foo(1, 2, 3)` to invoke the method `foo` in the remote server.

**names signature**(x = "DynamicSOAPServer") returns the names of the server's available methods.

## Author(s)

Duncan Temple Lang <duncan@wald.ucdavis.edu>

## References

<http://www.w3.org/TR/SOAP/> <http://www.omegahat.org/SSOAP>, <http://www.omegahat.org/bugs>.

## See Also

[SOAPServer](#)

## Examples

---

SOAPServerDescription-class

*Description of a SOAP Server's methods and data types*

---

## Description

This represents a complete description of the methods and associated data types for inputs and outputs of a SOAP server.

## Objects from the Class

Objects can be created by calls of the form `new("SOAPServerDescription", ...)`. More typically, however, one will use `processWSDL` to create such an object.

## Slots

**name:** Object of class "character". The name of the server.

**server:** Object of class "SOAPServer". The details of how to identify or connect to the server object.

**operations:** Object of class "list". A list of the sets of operations/methods. A server may have more than one collection of methods. This list is the top-level container and each element is itself a list containing `WSDLMethod` objects.

**types:** Object of class "list". The named collection of data types defined within the WSDL for the server.

**nameSpaces:** a character vector that identifies the namespace-URI mappings used for calls to this server. This maps the namespace abbreviations to the actual URIs. This can be a named character vector of these mappings, or alternatively a simple character string that identifies the name of the element in the `.SOAPDefaultNameSpaces` list. And if we don't know the collection of namespaces, we use NA to indicate that we shall determine this later.

## Methods

No methods defined with class "SOAPServerDescription" in the signature.

## Author(s)

Duncan Temple Lang <duncan@wald.ucdavis.edu>

## References

<http://www.w3.org/TR/SOAP/> <http://www.omegahat.org/SSOAP>, <http://www.omegahat.org/bugs>.

## See Also

[processWSDL](#)

## Examples

```
serverDesc = processWSDL(system.file("examples", "KEGG.wsdl", package = "SSOAP"))
```

---

SOAPServerDescription

*Constructor for describing methods and data structures of a SOAP server*

---

## Description

This function creates an instance of the class `SOAPServerDescription` (or the value of `obj`) and populates it with the specified collections of SOAP operations and data structure types, and information about the location of the SOAP server. This description can then be used to generate code to interface to the server's methods (see [genSOAPClientInterface](#)). The information is typically generated by reading the WSDL file, e.g. via [processWSDL](#).

## Usage

```
SOAPServerDescription(name, server, operations, types,
                      nameSpaces = NA, obj = new("SOAPServerDescription"))
```

**Arguments**

name	a string (character vector) giving the name of the SOAP server. This typically comes from
server	
operations	
types	
obj	an instance of the desired result whose slots are filled in during the call. This should be “compatible” with (i.e. extend) <a href="#">SOAPServerDescription</a> .
nameSpaces	a character vector that identifies the namespace-URI mappings used for calls to this server. This maps the namespace abbreviations to the actual URIs. This can be a named character vector of these mappings, or alternatively a simple character string that identifies the name of the element in the <code>.SOAPDefaultNameSpaces</code> list. And if we don’t know the collection of namespaces, we use NA to indicate that we shall determine this later.

**Value**

The object `obj` returned with the relevant fields filled in with values

**Author(s)**

Duncan Temple Lang <duncan@wald.ucdavis.edu>

**References**

<http://www.w3.org/TR/SOAP/> <http://www.omegahat.org/SSOAP>

**See Also**

[processWSDL](#)

**Examples**


---

SOAPServer	<i>Create a SOAP server object</i>
------------	------------------------------------

---

**Description**

These are constructors for the basic `SOAPServer` class which represent the location of the web services and methods. The basic `SOAPServer-class` is used to identify the host, port and URL of a SOAP server. The dynamic SOAP server is represented by the class `DynamicSOAPServer-class` and that contains not only the location of the SOAP server, but also information about its methods and data types. This information is typically read from a Web Services Description Language (WSDL) file.

We can use the form `server$method(arg1, arg2, ...)` to invoke a method in both types of server.

**Usage**

```
SOAPServer(host, url, port = NA, s = new(className))
dynamicSOAPServer(iface, obj = new("DynamicSOAPServer"))
```

**Arguments**

host	typically, the name of the host machine, e.g. "www.omegahat.org". Alternatively, a complete URL (e.g. http://www.omegahat.org/SOAP) may be given as the value for host and the individual parts are extracted from this.
url	the file/URL within the server that contains the SOAP server. If this is omitted, we attempt to find the value from the value of host.
port	the port number on which the server is listening. This is typically 80, the standard HTTP port. However, one can specify this when creating the S server object to identify a different port. This is useful when testing a server since one can use a user-level port. It is left as NA if not specified to indicate that it was not explicitly set to 80.
s	the object being created and initialized. Having this as an argument allows the caller to specify the class of the desired object and supply a partially initialized value and still get the "standard" initialization for the server object. className is computed in the body of the function and this mechanism works via lazy evaluation.
iface	this is an object of class SOAPClientInterface-class typically returned from a call to genSOAPClientInterface. This represents the collection of methods that the SOAP server provides.
obj	the object that will be returned from the dymanicSOAPServer function. This is specified with a default value so that this constructor can be easily reused for derived classes. Typically, a user-level call to this function will not need to specify this.

**Value**

An object of class HTTPSOAPServer, FTPSOAPServer or SOAPServer. If the host is specified with an ftp: or http: prefix, an object of class FTPSOAPServer or HTTPSOAPServer respectively is returned. Otherwise, a generic SOAPServer is created.

**Note**

In the future, we will use a SOAPConnection class that builds on the server and maintains a connection to the server. The URL may get dropped from the server as we can use the same basic host and port combination with different URLs for different requests. Experience will give us a better handle on an appropriate interface.

Also, we may store a server-specific, default SOAPAction value in the server.

**Author(s)**

Duncan Temple Lang <duncan@wald.ucdavis.edu>

**References**

<http://www.w3.org/TR/SOAP/> <http://www.omegahat.org/SSOAP>, <http://www.omegahat.org/bugs>.

**See Also**

[.SOAP](#) §

**Examples**

```
server = SOAPServer("www.nanonull.com", "TimeService/TimeService.asmx")
```

---

SOAPType-class	<i>Classes for representing types for SOAP values</i>
----------------	-------------------------------------------------------

---

**Description**

These classes are for representing types from SOAP-related schema. They are used in describing SOAP methods read from a Web Service Description Language (WSDL) file.

**Objects from the Class**

Objects can be created by calls of the form `new("SOAPType", ...)`. More typically, however, these are created by reading a WSDL file using [processWSDL](#).

**Slots**

**name:** Object of class "character". The name for the type of value being described.

**ns:** Object of class "character". The shorthand for the XML namespace associated with this type.

**nsuri:** Object of class "character". The URI/identifier that identifies the XML namespace associated with this type.

**Methods**

**toSOAP** `signature(obj = "vector", con = "ANY", type = "SOAPType"): convert an R object to its SOAP representation for the specified SOAP type.`

**Author(s)**

Duncan Temple Lang <duncan@wald.ucdavis.edu>

**References**

<http://www.w3.org/TR/SOAP/> <http://www.omegahat.org/SSOAP>, <http://www.omegahat.org/bugs>.

**See Also**

[processWSDL](#)

**Examples**

---

`toSOAP`*Convert S object to SOAP format*

---

**Description**

This converts an S object to its SOAP representation, writing it out to a connection.

**Usage**

```
toSOAP(obj, con = xmlOutputBuffer(header = ""), type = NULL,  
       literal = FALSE, ...)
```

**Arguments**

<code>obj</code>	the S object to be described in SOAP format
<code>con</code>	the connection on which to write the SOAP representation, usually a connection to a SOAP server.
<code>type</code>	an object that describes the target type, if available. This is typically an object which is derived from <code>SOAPType-class</code> that describes the details of the particular type.
<code>literal</code>	a logical value indicating whether to use the literal format of the encoding for the seralization of objects.
<code>...</code>	

**Value**

The side-effect of writing the representation to the connection is the important aspect of this.

**Author(s)**

Duncan Temple Lang <duncan@wald.ucdavis.edu>

**References**

<http://www.w3.org/TR/SOAP/> <http://www.omegahat.org/SSOAP>, <http://www.omegahat.org/bugs>.

**See Also**

[.SOAP](#)

**Examples**

---

writeSOAPBody	<i>Write SOAP message elements directly to connection</i>
---------------	-----------------------------------------------------------

---

### Description

These functions write the different parts of the SOAP request directly to an S connection. This means that they generate their content for the connection in order.

### Usage

```
writeSOAPBody(method, ..., xmlns = NULL, con, .types = NULL,
              .soapArgs = list(), .literal = FALSE)
writeSOAPEnvelope(con, namespaces = .SOAPNameSpaces())
writeSOAPMessage(con, namespaces, method, ..., .types = NULL,
                 xmlns = NULL, .soapArgs = list(), .literal = FALSE)
```

### Arguments

method	the name of the SOAP method to be invoked
...	For writeSOAPBody and writeSOAPMessage, these are the name-value arguments for the SOAP method being called.
.soapArgs	an alternative mechanism for passing arguments to the .SOAP call. This is a list of named or unnamed values which is used as the arguments for the SOAP method invocation.
xmlns	the namespace given either as a simple string or as a named character vector of namespace URIs and local names. (Currently only one namespace is used). This is used for the top-level element of the node within the SOAP Body, corresponding to the actual request.
con	the connection object on which to write the HTTP and SOAP content
.types	a list paralleling the arguments to the SOAP method (i.e. ... or .soapArgs) that specify the expected/required type of the individual arguments. This information is typically constructed from the WSDL (Web Services Description Language) if that is available. Otherwise, this can be an empty list in which case no constraints are placed on the arguments and the values are used as-is.
nameSpaces	a named character vector giving the namespace identifier and URI pairs. These are added as attributes in the SOAP Body element of the generated XML.
.literal	a logical value indicating whether to use the literal format of the encoding for the serialization of objects.

### Details

### Value

For each function, the return value is irrelevant. It is the side-effect of writing to the connection that is used for.

**Note**

A different approach is to create the XML “payload” first as a string (by creating it as an XML tree and then serializing that to a buffer). This allows one to add the Content-Length to the HTTP header.

**Author(s)**

Duncan Temple Lang <duncan@wald.ucdavis.edu>

**References**

<http://www.w3.org/TR/SOAP/> <http://www.omegahat.org/SSOAP>, <http://www.omegahat.org/bugs>.

**See Also**

[.SOAP](#)

**Examples**

---

writeTypes

*Output SOAP type information for an S object.*

---

**Description**

This is used in the [toSOAP](#) methods when writing an S object to a SOAP connection. It writes the SOAP attributes representing the SOAP type for the S object being serialized to the SOAP connection.

**Usage**

```
writeTypes(x, con, types = getSOAPType(x))
```

**Arguments**

x	the S object being serialized to the SOAP connection
con	the S connection object to which to write the type information attributes
types	the type information attributes which are computed by calling <a href="#">getSOAPType</a> by default.

**Value**

The string giving the SOAP type attributes that are written to the connection.

**Author(s)**

Duncan Temple Lang <duncan@wald.ucdavis.edu>

## References

<http://www.w3.org/TR/SOAP/> <http://www.omegahat.org/SSOAP>, <http://www.omegahat.org/bugs>.

## See Also

[toSOAP](#)

## Examples

---

WSDLMethod-class     *Description of a SOAP method*

---

## Description

This class is used to describe the elements of a SOAP method as described in a Web Service Description Language (WSDL) file.

## Objects from the Class

Objects can be created by calls of the form `new ("WSDLMethod", ...)`.

## Slots

**name:** Object of class "character". The name of the method.

**parameters:** Object of class "list". An ordered list of the parameter types for this method.

**returnValue:** Object of class "SOAPType". The type of the return value.

**action:** Object of class "SOAPAction". The SOAP action value associated with this method.

**namespace:** Object of class "character". The namespace associated with this method.

**use:** a character vector with elements for input and output indicating whether the parts of the message are encoded using some encoding rules, or definite the schema. Each value is either "literal" or "encoded". See <http://www.w3.org/TR/wsdl>, section 3.5.

**documentation:** a string providing a human-readable (or more specifically arbitrary formed text) supposed to describe the method

**bindingStyle:** the format/protocol of the XML messages sent to invoke and reply to a method, e.g. document and RPC are the most common ones. But others are possible.

## Methods

No methods defined with class "WSDLMethod" in the signature.

## Author(s)

Duncan Temple Lang <[duncan@wald.ucdavis.edu](mailto:duncan@wald.ucdavis.edu)>

**References**

<http://www.w3.org/TR/SOAP/> <http://www.omegahat.org/SSOAP>, <http://www.omegahat.org/bugs>.

**See Also**

[processWSDL](#)

**Examples**


---

.XMLRPC

*Invoke a server method via XML-RPC*

---

**Description**

This function uses a different, simpler protocol for invoking a method in a remote server

**Usage**

```
.XMLRPC(server, method, ..., .opts = list(), write = basicTextGatherer(), .convert
```

**Arguments**

server	
method	Describe method here
...	Describe ... here
.opts	Describe .opts here
write	Describe write here
.convert	Describe .convert here
curl	Describe curl here

**Details**

If necessary, more details than the description above

**Value**

For typical calls where one only specifies the server, the method name and any arguments, the return value is the result converted from the XML returned by the remote server. If this is a regular value, it is converted to its R equivalent, i.e. a number, logical or string value or a list corresponding to a struct in XML-RPC. If the method invocation results in a fault, an XMLRPCError is raised by the function. This is a specialized `simpleError` object (see [signalCondition](#))

**Author(s)**

Duncan Temple Lang <[duncan@wald.ucdavis.edu](mailto:duncan@wald.ucdavis.edu)>

**References**

<http://www.omegahat.org/SSOAP> <http://www.xmlrpc.com/spec>

**See Also**

[.XMLRPC xmlTreeParse](#)

**Examples**


---

xmlRPCResult	<i>Process result from XML-RPC request</i>
--------------	--------------------------------------------

---

**Description**

This function is used to process the response from a method invocation via XML-RPC to a remote server. This converts the result contained in the XML response into either a regular R value or a `XMLRPCError` object which contains a diagnostic error message and error code.

`xmlRPCHandlers` is a collection of XML-to-R converters for processing the XML nodes in the method response into the corresponding R values. One can call this function and then replace individual entries with one's own specialized methods.

**Usage**

```
xmlRPCResult(txt, handlers = xmlRPCHandlers(call), simplify = TRUE, call = sys.c
xmlRPCHandlers(call)
```

**Arguments**

<code>txt</code>	the XML content returned from the method invocation.
<code>handlers</code>	a
<code>simplify</code>	a logical value indicating whether to simplify the result when it is a list of length 1, i.e. to return just the child element. This is convenient for interactive use, but ensuring that one gets back a predictable type for the result is more useful in programmatic use.
<code>call</code>	the R command/expression associated with the XML-RPC request so that this can be added to the <code>XMLRPCError</code> if an exception is raised by the server. This is typically not specified by the caller, but the default value works appropriately.

**Details****Value****Author(s)**

Duncan Temple Lang <duncan@wald.ucdavis.edu>

**References**

<http://www.omegahat.org/SSOAP> <http://www.xmlrpc.com/spec>

**See Also**

[.XMLRPC xmlTreeParse](#)

**Examples**

# Index

## \*Topic **IO**

- .XMLRPC, 31
- getReturnNode, 6
- SOAPServer-class, 21
- xmlRPCResult, 32

## \*Topic **classes**

- duration-class, 2
- SOAPClientInterface-class, 11
- SOAPResult-class, 19
- SOAPServer-class, 21
- SOAPServerDescription-class, 22
- SOAPType-class, 26
- WSDLMethod-class, 30

## \*Topic **connection**

- .SOAP, 17
- convertFromSOAP, 1
- fromSOAP, 3
- getNodeById, 6
- getSOAPType, 7
- isHTTPError, 8
- parseSOAP, 9
- processWSDL, 10
- SOAP.logical, 15
- SOAPFault, 13
- SOAPHandlers, 14
- SOAPNameSpaces, 16
- SOAPServer, 24
- toSOAP, 27
- writeSOAPBody, 28
- writeTypes, 29

## \*Topic **datasets**

- SOATypes, 12

## \*Topic **interface**

- .SOAP, 17
- convertFromSOAP, 1
- fromSOAP, 3
- genSOAPClientInterface, 4
- getNodeById, 6
- getSOAPType, 7
- isHTTPError, 8
- parseSOAP, 9
- processWSDL, 10

- SOAP.logical, 15
- SOAPFault, 13
- SOAPHandlers, 14
- SOAPNameSpaces, 16
- SOAPResult, 20
- SOAPServer, 24
- SOAPServerDescription, 23
- toSOAP, 27
- writeSOAPBody, 28
- writeTypes, 29

## \*Topic **programming**

- .XMLRPC, 31
- genSOAPClientInterface, 4
- getReturnNode, 6
- xmlRPCResult, 32
- .SOAP, 1, 4, 6–9, 12–14, 16, 17, 19–21, 26, 27, 29
- .SOAPDefaultHandlers (SOATypes), 12
- .SOAPDefaultNameSpaces (SOATypes), 12
- .XMLRPC, 31, 32, 33
- \$, 26
- \$, DynamicSOAPServer, character-method (SOAPServer-class), 21
- \$, SOAPServer, character-method (SOAPServer-class), 21

- as.logical, 15

- BasicSOAPType-class (SOAPType-class), 26

- character, 2
- convertFromSOAP, 1
- convertFromSOAP, SOAPResult-method (convertFromSOAP), 1
- curlPerform, 8, 9, 18, 19

- duration-class, 2
- dynamicSOAPServer (SOAPServer), 24
- DynamicSOAPServer-class, 24
- DynamicSOAPServer-class (SOAPServer-class), 21

- fromSOAP, [3](#), [6](#), [10](#), [15](#), [21](#)
- fromSOAP, ANY, ANY, ANY, ANY, ANY, ANY-method  
(fromSOAP), [3](#)
- fromSOAP, ANY, ANY, ANY, ANY, ANY, ArrayType-method  
(fromSOAP), [3](#)
- fromSOAP, ANY, ANY, ANY, ANY, ANY, character-method  
(fromSOAP), [3](#)
- fromSOAP, ANY, ANY, ANY, ANY, ANY, missing-method  
(fromSOAP), [3](#)
- fromSOAP, ANY, ANY, ANY, ANY, ANY, NULL-method  
(fromSOAP), [3](#)
- fromSOAP, ANY, ANY, ANY, ANY, ANY, SOAPVoidType-method  
(fromSOAP), [3](#)
- fromSOAP, ANY, ANY, ANY, ANY, NULL-method  
(fromSOAP), [3](#)
- fromSOAP, character, ANY, ANY, ANY, ANY-method  
(fromSOAP), [3](#)
- fromSOAP, character, ANY, ANY, ANY, ANY, ArrayType-method  
(fromSOAP), [3](#)
- fromSOAP, character, PrimitiveSOAPType, ANY, ANY, ANY-method  
(fromSOAP), [3](#)
- fromSOAP, XmlNode, ANY, ANY, ANY, ANY, ClassDefinition-method  
(fromSOAP), [3](#)
- fromSOAP, XmlNode, ANY, ANY, ANY, ANY, PrimitiveSOAPType-method  
(fromSOAP), [3](#)
- fromSOAPArray (fromSOAP), [3](#)
- fromSOAPStruct (fromSOAP), [3](#)
- genSOAPClientInterface, [4](#), [11](#), [12](#),  
[23](#), [25](#)
- getNodeById, [6](#)
- getReturnNode, [6](#)
- getSOAPType, [7](#), [29](#)
- HTTPSOAPServer-class  
(SOAPServer-class), [21](#)
- HTTPSSOAPServer-class  
(SOAPServer-class), [21](#)
- isHTTPError, [8](#), [19](#), [21](#)
- lapply, [3](#)
- names, DynamicSOAPServer-method  
(SOAPServer-class), [21](#)
- new, [21](#)
- parseSOAP, [9](#)
- postForm, [19](#)
- processWSDL, [2](#), [5](#), [10](#), [22–24](#), [26](#), [31](#)
- sapply, [3](#)
- setClass, [4](#)
- signalCondition, [31](#)
- SOAP.logical, [15](#)
- SOAPClientInterface-class, [25](#)
- SOAPClientInterface-class, [11](#)
- SOAPFault, [13](#)
- SOAPHandlers, [14](#)
- SOAPNameSpaces, [14](#), [16](#)
- SOAPPrimitiveConverters, [15](#)
- SOAPPrimitiveConverters  
(SOAPTypes), [12](#)
- SOAPResult, [20](#)
- SOAPResult-class, [19](#)
- SOAPServer, [21](#), [22](#), [24](#)
- SOAPServer-class, [24](#)
- SOAPServer-class, [21](#)
- SOAPServerDescription, [10](#), [23](#), [24](#)
- SOAPServerDescription-class, [4](#)
- SOAPServerDescription-class, [22](#)
- SOAPType-class, [27](#)
- SOAPType-class, [26](#)
- SOAPTypeReference-class  
(SOAPType-class), [26](#)
- SOAPTypes, [12](#)
- SOAPVoidType-class  
(SOAPTypes), [26](#)
- token-class (duration-class), [2](#)
- toSOAP, [27](#), [29](#), [30](#)
- toSOAP, ANY, ANY, ArrayType-method  
(toSOAP), [27](#)
- toSOAP, ANY, ANY, ClassDefinition-method  
(toSOAP), [27](#)
- toSOAP, list, ANY, NULL-method  
(toSOAP), [27](#)
- toSOAP, vector, ANY, NULL-method  
(toSOAP), [27](#)
- toSOAP, vector, ANY, PrimitiveSOAPType-method  
(toSOAP), [27](#)
- toSOAP, vector, ANY, SOAPType-method  
(toSOAP), [27](#)
- typeof, [12](#)
- vector, [2](#)
- VirtualSOAPClass-class  
(SOAPType-class), [26](#)
- writeSOAPBody, [28](#)
- writeSOAPEnvelope  
(writeSOAPBody), [28](#)
- writeSOAPMessage, [8](#), [19](#)
- writeSOAPMessage (writeSOAPBody),  
[28](#)
- writeTypes, [29](#)
- WSDLMethod-class, [30](#)

`xmlRPCHandlers (xmlRPCResult)`, [32](#)  
`xmlRPCResult`, [32](#)  
`xmlTreeParse`, [7](#), [9](#), [10](#), [32](#), [33](#)