Package 'SSOAP'

November 30, 2011

Version 0.8-1

Date 2011/10/29

Title Client-side SOAP access for S

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

Depends R (>= 1.5.0), methods

Imports XML, RCurl, XMLSchema

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

Description A package that provides a client
interface to SOAP (Simple Object Access Protocol) servers from within R.

LazyLoad yes

License GPL2

Collate ClassDefns.R fault.S http.S parseSOAP.S print.S RPC.R
SOAP.S SOAPServer.S zSOAP.S duration.S nativeWrite.R
functionAccessors.R writeGeneratedCode.R wsdlProcess.R utils.R toSOAP.R httpGet.R

Note There is an opportunity to do a lot of optimization in
the machine generated code to reduce the processing
time. This is a simple approach that ''works'' with little regard for optimization.

URL http://www.omegahat.org/SSOAP, http://www.omegahat.org
http://www.omegahat.org/bugs

R topics documented:

`.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, .soapHeader = NULL, .elementFormQualified = FALSE,
      .returnNodeName = NA)
```

Arguments

`server`	a `SOAPServer` object
`method`	the name of the SOAP method to invoke
`...`	name=value arguments to pass to the
`.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.
`action`	the SOAPAction string to put in the HTTP header. This is required. If it is an object of class `AsIs`, it is left exactly as it is. This allows one to call this function as .SOAP(...., action = I("einfo")) without having to provide a handler to bypass the default action mechanism.

`nameSpaces` 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 `nameSpaces` is a single string, we use it to index the element in the `.SOAPDefaultNameSpaces` list.

`xmlns` 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.

`handlers` 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 `action` for creating the final form of the SOAPAction string, `converter` for processing the XML returned by the SOAP server in the case of a successful invocation, and so on.

`.types` [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.

`.convert` a function, a logical value or a `SOAPType`. 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 SOAPType object. This should return an R object representing the SOAP content. If, alternativley, this is supplied as a logical value, this controls whether the default converters are used (`TRUE`) or not (`FALSE`). These converters are taken from the `handlers` argument. And finally, if `.convert` is a `SOAPType` object, we call `convertFromSOAP` with the

`.opts` a named list of elements that are passed to the curlPerform 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. `.opts = list(verbose = TRUE)`

`curlHandle` this is passed to `curlPerform` as the `curlHandle` 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.

`.header` a named character vector of elements which are used in the HTTP header for the SOAP request. These are calculated by default within the .SOAP call, but the parameter allows them to be pre-computed and supplied in the call.

`.literal` a logical value indicating whether to use the literal encoding for serializing the data being sent to and from the server.

`.soapHeader` this allows the caller to specify the SOAP content for the Header part of the SOAP request. This is sometimes used to supply information such as login and password or other forms of authentication and authorization. The value for this parameter can be the text of the XML header node, an XML node itself, or alternatively a function that returns such a node (or text with the XML content). The function is called with the SOAP document being created and the name of the SOAP method.

`.elementFormQualified`
a logical value. If this is `FALSE`, only the XML element identifying the method call in the Body of the SOAP request uses the target namespace. The XML nodes representing the arguments in the method call do not use this namespace

but are global. Alternatively, if this is TRUE, the target namespace of the schema is defined as the default name space on the XML element for the method call and so is inherited by the elements for the parameters.

`.returnNodeName`
: the name of the node in the SOAP response that is the container for the content of the response. This is often "return" but can be any legal XML node name and is often given to us in a WSDL.

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)
```

.SOAPHttpGet *Functions for making SOAP calls via HTTP GET or POST requests*

Description

These functions provide the basic functions for implementing a SOAP request via HTTP GET or POST bindings rather than the regular SOAP Envelope and Body bindings. These are often easier to use manually as they allow the caller to think of the request as a form or simple URL with named parameters.

The functions can be used in generated code derived from processing a WSDL.

Usage

```
.SOAPHttpGet(.url, ..., .params = list(...), .convert = TRUE, .opts = list())
.SOAPHttpPost(.url, ..., .params = list(...), .convert = TRUE, .opts = list(), .style = 'POST')
```

Arguments

.url	the full URL (as a string) identifying the SOAP method
...	a named list of parameters that are passed directly to the HTTP request.
.params	this is a list of the parameters. This can be used by a caller who already has the inputs to the SOAP request in a list.
.convert	a value that controls how the result is returned. If this is FALSE, the text of the HTTP request is returned directly. If this is TRUE, we use fromXML to convert the XML content to an R object. If .convert is an object derived from the GenericSchemaType class that describes a data structure, then we use that data desciption to convert the XML to the corresponding R data type.
.opts	a list of named options that are passed to getForm and postForm to control the RCurl request. See listCurlOptions.
.style	this is the style parameter of postForm and should be either "POST" or "HTTP-POST".

Value

Either the text of the SOAP response or an R object converted from the XML content.

Author(s)

Duncan Temple Lang

See Also

.SOAP

Examples

```
 .SOAPHttpGet("http://www.chemspider.com/MassSpecAPI.asmx/SearchByMass2", mass = 89.0476, range = 0.01)

## Not run:
    # need token
  .SOAPHttpGet("http://www.chemspider.com/MassSpecAPI.asmx/GetExtendedCompoundInfoArray", CSIDs = c("23500

## End(Not run)
```

`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, nodeName = "return", ...)
```

Arguments

`val`	the XML object representing the data.
`type`	the target "type" of object to which the XML should be converted.
`nodeName`	the name of the node in the SOAP response that is the container for the content of the response. This is often `"return"` but can be any legal XML node name and is often given to us in a WSDL.
`...`	additional arguments for (future) methods

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`

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(parent = globalenv()), where = globalenv(),
                       server = def@server, nameSpaces = def@nameSpaces,
                       addSoapHeader = FALSE, verbose = FALSE, force = FALSE,
                       putFunctions = FALSE,  verb = def@verb,
                       opFun = getOperationFunction(verb), opts = new("CodeGenOpts"), ...)
```

Arguments

operations	a list of the descriptions of the server's methods. Each method description provides information about the parameters and the return value.
def	the SOAPServerDescription-class object.
name	currently unused
env	an environment object. This is used ?
where	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 where argument of setClass, i.e. an integer, a package name ("package:name") or, explicitly, an environment.
server	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.
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.
addSoapHeader	controls whether a .soapHeader parameter is added to each function that is generated for the SOAP server. If this is a logical, TRUE indicates to add the .soapHeader parameter; FALSE indicates it is omitted. If this is not a logical value, it is taken as the value to be supplied as the default value for the .soapHeader parameter in each generated function.
verbose	a logical indicating whether information about the processing should be displayed on the console, as it occurs.

force	a logical value that controls how we handle the case where we would define an S4 class corresponding to a data type in the schema but for which there already exists a class of that name (within the environment in which we are defining the schema-related classes). TRUE means that we will overwrite the existing class definition; FALSE means to leave the existing class definition. This is useful when we run the interface generation code a second time and so have existing class defintions from the first run.
putFunctions	either a logical value or an environment or something that can be passed as the third argument to assign. This controls whether we assign the individual functions as variables in where or another environment.
opts	an instance of the class CodeGenOpts or a derived sub-class that is used to specify the parameters that control the sub-functions that generate the code. At present, this is limited to the makePrototype slot that controls whether we map string elements to empty "" strings rather than zero-length character vectors. Whether this is a good choice depends on the Web service/interface and if it's WSDL deals with optional values properly.
verb	a string identifying the type of binding/transport for the requests. This can be "GET", "POST" or NA which is the default and refers to the regular SOAP messaging.
opFun	the function for creating the R function for each operation. This depends on verb to generate different functions for different transportation mechanisms.
...	additional arguments passed to processSchemaTypes

Value

An object of class SOAPClientInterface containing both functions and class definitions.

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")

## Not run:
    # This KEGG.wsdl is out of date
 tmp = processWSDL(system.file("examples", "KEGG.wsdl", package = "SSOAP"))
 iface = genSOAPClientInterface(tmp@operations[[1]], def = tmp, tmp@name, verbose=FALSE)

## End(Not run)
```

```
setAs("Definition", "character",
        function(from)
            structure(from@entry_id, names = from@definition))

setAs("ArrayOfPathwayElement", "character",
         function(from) sapply(from, as, "character"))

o = iface@functions$list_organisms()

as(o, "character")

cat("See the file", system.file("examples", "KEGG.S", package = "SSOAP"), "for more examples\n")

    # Returns National Weather Service digital weather forecast data.
 w = processWSDL("http://www.weather.gov/forecasts/xml/SOAP_server/ndfdXMLserver.php?wsdl")
 f = genSOAPClientInterface(,w)

   # For the next 5 days from now.
 str = f@functions$NDFDgenByDay(38.9936, -77.0224, Sys.time() +  60*24*60, 5, "m", "12 hourly")

 library(XML)
 xmlToList(xmlParse(str, asText = TRUE))

    # Note that the result is a string containing XML. The WSDL identifies the result
    # as an xsd:string and does not tell us about the structure of the contents.


    ##
    ## WABI
if(FALSE) {
    # This site was not available May 4th.
  w = processWSDL("http://xml.nig.ac.jp/wsdl/ARSA.wsdl")
  f = genSOAPClientInterface(,w )

  f@functions$searchSimple("Homo sapiens", 1, 100)
  x = f@functions$getENTRYbySPECIMEN("ATCC 43049", 1, 100)
  x = f@functions$getENTRYbyScientificName("Escherichia coli", 1, 400)
  els = readLines(textConnection(x))
    # get how many results are available in the database.
  totalCount = as.integer(substring(els[1], nchar("hitscount       = ")+1))
  hits = els[-1]  # the 400 results in the answer


  x = f@functions$searchByXMLPath("/ENTRY/DDBJ/accessions/accn='ab0001'",
                                  "/ENTRY/DDBJ/primary-accession,/ENTRY/DDBJ/definition",
                                  1, 100)
  els = unlist(strsplit(x, "\\n"))
  totalCount = as.integer(substring(els[1], nchar("hitscount       = ")+1))

  values = strsplit(els[-1], "\\t")
  ans = structure(sapply(values, `[`, 2), names = sapply(values, `[`, 1))

   ###
  w = processWSDL("WSDLs/MassSpecAPI.asmx?WSDL", port = 1)
  f = genSOAPClientInterface(w)
    # SearchByMass2 expects an object of class SearchByMass2 which has
```

```
   # a mass and a range slot. But we can specify these separately, by
   # name or partial name, or as a list or as a SearchByMass2 object
  f@functions$SearchByMass2(89.0476, .01)
  f@functions$SearchByMass2(mass = 89.0476, range = .01)
  f@functions$SearchByMass2(range = .01, mass = 89.0476)

  f@functions$SearchByMass2(list(range = .01, mass = 89.0476))
  f@functions$SearchByMass2(new("SearchByMass2", range = .01, mass = 89.0476))

   ####
 # A HTTP GET interface
  w = processWSDL("WSDLs/MassSpecAPI.asmx?WSDL", port = 3)
  f = genSOAPClientInterface(w)

  f@functions$GetDatabases()
  f@functions$SearchByMass2(89.0476, .01)
  f@functions$SearchByFormula("H2O")

## Not run:
   # ned the token
  f@functions$GetExtendedCompoundInfoArray(c("23500", "23543"), token)
  f@functions$GetExtendedCompoundInfoArray(as.character(c(23500, 23543)), token)

## End(Not run)


  ##########
  w = processWSDL("WSDLs/MassSpecAPI.asmx?WSDL", port = "MassSpecAPIHttpPost")
  f = genSOAPClientInterface(w)
  names(f@functions)
  db = f@functions$GetDatabases()
  f@functions$SearchByMass2(89.0476, .01)
## Not run:
   # ned the token
  f@functions$GetExtendedCompoundInfoArray(c("23500", "23543"), token)
  f@functions$GetExtendedCompoundInfoArray(as.character(c(23500, 23543)), token)

## End(Not run)

}
```

`getReturnNode` *Get XML node from SOAP response*

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 form `.SOAP`, or the root node of a previously parsed XML tree.

Usage

```
getReturnNode(node, name = "return")
```

Arguments

node	either an XML node that was obtained from parsing the text of the reply or the SOAPHTTPReply object returned from the .SOAP 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.
name	the name of the node in the SOAP response that is the container for the content of the response. This is often "return" but can be any legal XML node name and is often given to us in a WSDL.

Value

An XMLNode object.

Author(s)

Duncan Temple Lang

See Also

.SOAP xmlTreeParse

---

getSOAPType *Compute the SOAP type identifier for an S object*

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 forthe 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

---

isHTTPError *Determines if an error occurred in an HTTP communication*

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 curlPerform in the .SOAP 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

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

xmlSource	an string giving the name of an file or URL containing the XML content, or a string containing the XML content itself.
header	ignored
reduce	a logical value indicating whether to return the top-most XML node of the Body or to return the entire XML tree read from xmlSource. If this is true, we find the Body node and return its first child.
...	arguments that are passed directly to xmlTreeParse without being processed by this function.
fullNamespaceInfo	a logical value that is passed on to xmlTreeParse 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

fromXML

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(),
            useInternalNodes = TRUE, verbose = FALSE, port = 1L,
            checkCircularTypes = TRUE)
```

Arguments

fileName	the name of the WSDL file or URI.
handlers	a list of handler functions that are passed to xmlTreeParse to perform transformations on the XML nodes as the are parsed and converted to R objects. The default handlers drop comments and take care of importing files that are referenced via <wsdl:import> 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 .SOAPDefaultNameSpaces list. And if we don't know the collection of namespaces, we use NA to indicate that we shall determine this later.
useInternalNodes	a logical value indicating whether to use internal/C-level nodes for the XML tree or to use R objects representing the nodes.
verbose	a logical value indicating whether to emit messages to the console signalling progress being made and what elements are being processed. This is passed to processSchemaTypes.
port	a number or string that is used to identify which of the port elements to use in the WSDL. This allows the caller to specify whether they want to use, e.g., SOAP 1.1 or SOAP 1.2 or SOAP PUT if the WSDL provides more than one of these
checkCircularTypes	a logical value that controls whether we check for circular references in definitions of data types in the XML schema.

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), orgs = c("ecs","ypk")

## 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 = "SSOAP"))

## Not run:
    cs = processWSDL("http://www.chemspider.com/MassSpecAPI.asmx?WSDL", port= 3)
    cs = processWSDL("http://www.chemspider.com/MassSpecAPI.asmx?WSDL", port = "MassSpecAPIHttpGet")

## End(Not run)
```

---

server *Information about a programmatically generated function*

Description

These functions provide information about a function that was programmatically created by processing a WSDL file. It allows the R user to find out characteristics of these functions without having to understand the particular structure of the functions, i.e. the environments and the metadata being stored in that environment and the default server object in a parent environment shared by all the generated functions.

The help function merely makes the regular help function in base generic.

When these functions are serialized and written to a file, these methods no longer work at present. This can be remedied.

Usage

```
server(fun, ...)
returnType(fun, ...)
returnConverter(fun, ...)
```

Arguments

fun	the function object whose characteristics are being queried. This is a programatically generated function as part of the WSDL processing. This should be of class WSDLGeneratedSOAPFunction.
...	additional arguments so that these generic functions can be useful to other packages.

Value

These merely provide information about the contents of the environment of the generated function.

Note

When they are serialized to a file, this information is no longer available at present

Author(s)

Duncan Temple Lang

See Also

processWSDL

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

SOAPFault *Create a SOAP Fault object*

Description

This creates an object representing a SOAP fault object returned from a 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

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

...	name=function pairs giving values to be returned in the list of functions. These override corresponding elements in .SOAPDefaultNameSpaces.
include	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 version.
exclude	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

action	convert the user-specified SOAPAction to the target one. By default, this appends #methodName 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 xmlns argument for .SOAP.

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")
```

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

`...`	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 `version` argument in the `.SOAPDefaultNameSpaces` list.
`include`	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 `version`.
`exclude`	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.
`version`	a name that identifies an element in the `.SOAPDefaultNameSpaces` 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 `...` 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")
```

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

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

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 fromXML isHTTPError

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

SOAPServer *Create a SOAP server object*

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")
```

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

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, verb = NA, obj = new("SOAPServerDescription"))
```

Arguments

`name`	a string (character vector) giving the name of the SOAP server. This typically comes from
`server`	an object of class SOAPServer that describes the location of the Web service, giving the URL, port, path to the services
`operations`	a list describing the operations or methods provided by the Web service
`types`	a list describing the data types used by the Web service
`obj`	an instance of the desired result whose slots are filled in during the call. This should be "compatible" with (i.e. extend) SOAPServerDescription.

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.
verb	a string identifying the type of the binding/transport to use, i.e. "POST", "GET" or NA for regular SOAP.

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

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.

`verb`: a character string identifying the type of the binding, i.e. GET, POST or `NA`.

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"))
```

SOAPType-class *Classes for representing types for SOAP values*

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`

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
```

Source

SSOAP package

References

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

---

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, elementFormQualified = FALSE, ...)
```

Arguments

obj	the S object to be described in SOAP format
con	the connection on which to write the SOAP representation, usually a connection to a SOAP server.
type	an object that describes the target type, if available. This is typically an object which is derived from SOAPType-class that describes the details of the particular type.
literal	a logical value indicating whether to use the literal format of the encoding for the seralization of objects.
elementFormQualified	a logical value that indicates whether the nodes are to use no namespace or the target namespace of the schema in which they were defined.
...	additional parameters for the methods of this generic

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

writeInterface *Serialize generated interface to a file*

Description

This function allows the user to take a programmatically generated interface such as with genSOAPClientInterface and write the code to a file for inclusion in an R package or to be source'd into a different R session.

Note that this is not essential. One can save the interface generated by genSOAPClientInterface in RDA format and then use load that into a different R session, potentially on a different machine, and the interface will work as is.

Usage

```
writeInterface(iface, file = stdout(), where = globalenv())
```

Arguments

iface	the interface object created via a call to genSOAPClientInterface
file	the name of the file to which to write the code.
where	the position or package name used to find the class definitions associated with this interface. This is passed to getClass.

Details

Currently, this has to handle deparseing S4 objects directly. Also, it undoes the use of environments within the functions to store the "cached" information about the operation and the SOAP server location. Instead, it adds these as explicit parameters and to the body of the code.

Value

This function is used for its side effect of writing content in the specified file

Author(s)

Duncan Temple Lang

See Also

genSOAPClientInterface processWSDL

Examples

```
w = processWSDL(system.file("examples", "KEGG.wsdl", package = "SSOAP"))
iface = genSOAPClientInterface(w)
f = tempfile()
writeInterface(iface, f)
source(f)
## Not run:
db = list_databases()
class(db)
class(db[[1]])
```

```
## End(Not run)

u = "http://gmd.mpimp-golm.mpg.de/webservices/wsGoBioSpace.asmx?WSDL"
z = processWSDL(u, port = 1)
iface = genSOAPClientInterface(z)
f = tempfile()
writeInterface(iface, f)
source(f)

a = GetAdducts()
d = GetDepositors()

session = CreateSession(DepositorIds = c(3, 6, 8), AdductIds = c(2L, 3L))
sm = SearchMass12C(SessionID = session, mass = 579.1705, tolerance = 0.001)
```

`writeSOAPBody` *Write SOAP message elements directly to connection*

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,
              .header = NULL, .elementFormQualified = FALSE)
writeSOAPEnvelope(con, nameSpaces = SOAPNameSpaces())

writeSOAPMessage(con, nameSpaces, method, ..., .types = NULL,
                    xmlns = NULL, .soapArgs = list(), .literal = FALSE,
                     .soapHeader = NULL, .elementFormQualified = 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 parallelling 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 seralization of objects.

.header — a character string (or NULL that is written as part of the SOAP header (not the HTTP header), before any other output, i.e. before the <SOAP-ENV:Body> is emitted. This is passed as the first argument to writeSOAPHeader.

.soapHeader — a string, an XML node or a function that can be optionally specified to add content to the SOAP message as the header of the envelope. This is used in some Web services to provide transaction information such as a authentication and security details. See the eBaySvc.wsdl for an example. If this is a function, it is called with the value of con as the only argument. One might use a closure to include the "private" and auxiliary information.

.elementFormQualified — a logical value. If this is FALSE, only the XML element identifying the method call in the Body of the SOAP request uses the target namespace. The XML nodes representing the arguments in the method call do not use this namespace but are global. Alternatively, if this is TRUE, the target namespace of the schema is defined as the default name space on the XML element for the method call and so is inherited by the elements for the parameters.

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

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 getSOAPType 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

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 definte 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.

`header`: a list. This provides information about additional HTTP header information should be passed in the request for this operation. For example, this might be tickets/cookies that authorize the requestor to make the request.

returnNodeName the name of the node in the SOAP response that is the container for the content of the response. This is often `"return"` but can be any legal XML node name and is often given to us in a WSDL.

Methods

No methods defined with class "WSDLMethod" 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`

WSDLParseHandlers *Creates functions for XML parser for processing import/include XML nodes*

Description

This function is used, by default, when using regular R-level nodes (as opposed to internal/C-level nodes) when parsing an XML schema document. The function returns two functions which are used by the XML parser `xmlTreeParse` to convert import and include nodes within the XML schema by resolving the name of the referenced file and reading that document as another XML schema and inserting the results into the document.

It is often more convenient to use internal/C-level nodes and so use `xmlParse` rather than `xmlTreeParse`. The former does not use these handler functions.

Usage

```
WSDLParseHandlers(baseURI, verbose = FALSE, keepSchema = FALSE)
```

Arguments

baseURI	the name of the parent/containing document being read, relative to which the href attribute of an import or include node will be resolved.
verbose	a logical indicating whether to emit messages to the console when processing an import or include node.
keepSchema	a logical value indicating whether to assign the resulting schema from processing an include or import node to the list of schema, indexed by the namespace prefix.

Value

A named list of functions.

Author(s)

Duncan Temple Lang

References

The XML schema specification at http://www.w3.org/XML/Schema.

See Also

readSchema in the XMLSchema package and processWSDL.

Index