Package 'sos4R'

August 9, 2019

Type Package

Title Client for OGC Sensor Observation Services

Version 0.3.1

Date 2019-07-09

Depends R (>= 3.4.0)

Imports httr, methods, sp, stringr, xml2 (>= 1.2.2),

Suggests readr, spacetime, gstat, maps, maptools, mapdata, cshapes, xtable, xts, testthat, rgdal, knitr, rmarkdown

Description A client for Sensor Observation Services (SOS, see <https://www.opengeospatial.org/standards/sos>) as specified by the Open Geospatial Consortium (OGC). With the package users can retrieve (meta)data from SOS instances and interactively create requests for near real-time observation data based on the available sensors, phenomena, observations etc. using thematic, temporal, and spatial filtering.

License GPL-2

URL https://github.com/52North/sos4R

Encoding UTF-8

LazyLoad TRUE

ByteCompile TRUE

BugReports https://github.com/52North/sos4R/issues

Collate Constants.R R-Helper.R Class-OWS.R Class-GML.R Class-SWE.R Class-OM.R Class-SA.R Class-SAMS.R Class-WML_200.R Class-OM_20.R Class-OGC.R Class-SOS.R Class-SOS_100.R Class-SOS_200.R Class-SOS_200_GDA.R Class-SML.R Generic-methods.R OWS-methods.R OWS-methods-parsing.R SOS-methods-parsing.R OM-methods.R OM-methods-coercion.R OM-methods-parsing.R OM_20-methods-parsing.R OM_20-methods.R SA-methods.R GML-methods.R SWE-methods.R SML-methods.R GML-methods-parsing.R SA-methods-parsing.R SWE-methods-parsing.R OGC-methods.R SOS-methods-accessor.R

PrintShowStructureSummmary-methods.R SOS-methods-util.R
SOS-methods.R SOS-methods-plotting.R SOS-methods-coercion.R
SML-methods-util.R SML-methods-coercion.R
SOS_200-methods-impl.R SOS_200-methods.R
SOS_200-methods-parsing.R SOS_200_methods-gda.R
WML_200-methods-parsing.R WML_200-methods.R
SAMS-methods-parsing.R Defaults.R Development.R

RoxygenNote 6.1.1

VignetteBuilder knitr

NeedsCompilation no

Author Daniel Nuest [cre, aut] (<https://orcid.org/0000-0002-0024-5046>),
Edzer Pebesma [ctb] (<https://orcid.org/0000-0001-8049-7069>),
Ben Graeler [ctb] (<https://orcid.org/0000-0001-5443-4304>),
Benjamin Pross [ctb],
Eike Hinderk Juerrens [ctb],
52°North Initiative for Geospatial Open Source Software GmbH [cph]

Maintainer Daniel Nuest <daniel.nuest@uni-muenster.de>

Repository CRAN

Date/Publication 2019-08-09 13:30:02 UTC

R topics documented:

`sos4R-package` *A client for the OGC Sensor Observation Service*

Description

sos4R is a client for Sensor Observation Services (SOS). It allows users to retrieve metadata from SOS web service instances as specified by the Open Geospatial Consortium (OGC) and subsequently to interactively create requests for observation data based on the available sensors, phenomena, observations, offerings etc.

Details

Package:	sos4R
Type:	Package
Version:	0.3.1
Date:	2019-07-09
License:	GPL-2
LazyLoad:	yes
ByteCompile:	yes
Imports:	xml2, httr, sp, stringr, methods

Note

The development of this software was gratefully supported by the 52North Student Innovation Prize for Geoinformatics 2010.

To stay updated on all matters around **sos4R** go to the development blog at http://www.nordholmen.net/sos4r/.

If you want to ask questions about using the software, please go to the to the issue tracker at https://github.com/52North/sos4R/issues.

The most extensive documenation is contained in the **package vignette**.

Author(s)

Daniel Nuest <daniel.nuest@uni-muenster.de>

References

Na, A., Priest, M. (Eds.), 2007. Sensor Observation Service. OpenGIS Implementation Standard, Version 1.0, OGC 06-009r6

See Also

See also the package vignette.

Examples

```
## Not run:

# Take a SOS from the example list
sos.url = SosExampleServices()[[1]]

# Open the connection
sos = SOS(url = SOS)

# List offerings, procedures and observedProperties
names(sosOfferings(sos))
sosProcedures(sos)
sosObservedProperties(sos)

# Create time period (last 30 days)
tPeriod <- sosCreateEventTimeList(
time = sosCreateTimePeriod(
sos = pegelsos,
begin = Sys.time() - (3600 * 24 * 30),
end = Sys.time()))

# Request data for all observed properties and procedures of a certain offering
observation <- getObservation(sos = sos,
observedProperty = sosObservedProperties(sos),
```

```
offering = sosOfferings(sos)[[2]],
procedure = sosProcedures(sos),
eventTime = tPeriod)

# Inspect result
sosResult(observation)
str(sosResult(observation))

# Inspect attributes of the data fields
if(is.list(sosResult(observation))) {
attributes(sosResult(observation)[,1])
}
else {
attributes(sosResult(pegelObs)[,1])
}

# Use custom converting function and connection method. This mechanism works the
# same for encoders and decoders.
myConverters <- SosDataFieldConvertingFunctions(
"myNumericUnit" = sosConvertDouble)
mySos <- SOS(sos.url, binding = "KVP", dataFieldConverters = myConverters)
sosDataFieldConverters(mySos)

# get the cheat sheet
sosCheatSheet()

# view the NEWS file
news(package = "sos4R")

## End(Not run)
```

`checkRequest-methods` *Methods for Function checkRequest*

Description

A function to check request prior to sending them to a service. This function is autmatically called during the request process and can be used to check the request for consistency with ifself as well as with available metadata, and also perform additional validity checks that might not be possible with class validation.

Methods

`signature(service = "SOS", operation = "DescribeSensor", verbose = "logical")` Checking a DescribeSensor request that is send to a SOS. This method currently checks the following elements:

- `operation@service` attribute must be SOS
- `operation@request` attribute must be DescribeSensor

- `operation@procedure` must be listed in the given service's capabilities
- `operation@outputFormat` must be supported by the operations capabilities description
- `operation@binding` must be supported by the package implementations. See SosBindings.

`signature(service = "SOS", operation = "SosGetObservationById", verbose = "logical")` Checking a GetObservationById request. **Warning: Function not implemented yet.**

`signature(service = "SOS", operation = "SosGetObservation", verbose = "logical")` Checking a GetObservation request. **Warning: Function not implemented yet.**

`signature(service = "SOS", operation = "OwsGetCapabilities_1.1.0", verbose = "logical")` Checking a GetCapabilities request. **Warning: Function not implemented yet.**

`signature(service = "SOS", operation = "OwsGetCapabilities_2.0.0", verbose = "logical")` Checking a GetCapabilities request. **Warning: Function not implemented yet.**

`Constants` *Constants in sos4R*

Description

The package **sos4R** comes with a set of constant character strings and fixed supported features, for example for names of XML elements, XML Namespace prefixes, or supported formats and models.

Details

Most of these variables should be pretty self-explanatory.

Constants for names of XML elements start with a lowercase character string of the namespace prefix (e.g. "gml"), a unique name of the element (where parts like "type" and special characters may be left out, and other descripte elements may be added for clarity), and end with "Name".

Examples: codegmlEnvelopeName, `ogcGeometryOperandLineStringName`, `ogcTempOpTMEqualsName`.

The `OwsExceptionsData()` function provides access to the fixed exception codes, meanings and respective HTTP codes and messages.

The **mime types** are used for automatic detection of the best fitting parser.

References

See also `Defaults` for default parameter settings.

Whiteside A. (Ed.), OGC Web Services Common Specification, Open Geospatial Consortium Inc., OGC 06-121r3, Version: 1.1.0 with Corrigendum 1

Examples

```
# example constants
sos100NamespacePrefix
gmlNameName
sweUomName

# Data frame holding OWS exception code information
OwsExceptionsData()

# Get all namespaces
SosAllNamespaces()
```

`Defaults` *Default Parameter Settings and Handling Functions*

Description

These values are default parameters and handling functions for connections and requests to, as well as response processing of ansers from, Sensor Observation Services. These allow to simplify a SOS connection for the most common use cases and non-expert users.

Usage

```
SosDefaultBinding()

SosParsingFunctions(..., include = character(0), exclude = character(0))
SosEncodingFunctions(..., include = character(0), exclude = character(0))
SosDataFieldConvertingFunctions(..., include = character(0), exclude = character(0))

SosDisabledParsers()

SosExampleServices()

SosDefaults()

SosResetParsingFunctions(sos)

SosDefaultDCPs()

SosDefaultParsingOptions()
```

Arguments

`...`	Named references to functions to be used for the respective element during parsing, encoding oder conversion, e.g. `"myUnit" = myUnitParser`.

include — A list of names of elements whose functions shall be included in the returned list, e.g. `include = c("GetObservation","DescribeSensor")`. This inclusion is done **after** replacing the default functions based on the `...` argument.

exclude — A list of names of elements whose functions shall be excluced in the returned list, e.g. `exclude = c("DescribeSensor")`. This exclusion is done **after** replacing the default functions based on the `...` argument.

sos — An object of class `SOS`.

Details

The default values are strongly related to what is actually implemented in the package, but also often resemble the (hopefully) most common use cases.

Some defaults are accessed directly, others should be accessed using a function. The latter is required for cases where a runtime evaluation is needed, e.g. for default values of construction functions.

A special case are the functions to access the default functions for specific purposes, which are the parsing functions, the encoding functions and the field converting functions. See the examples on how to use them.

The function `SosDisabledParsers` can be used to use no parsing at all (despite the parsing for the capabilities response, which is required for establishing a connection to a SOS. This function is helpful to inspect the unprocessed responses from a service.

The function `SosResetParsingFunctions` can be used to replace the included parsing functions of a `SOS` object with the default ones. This is even useful for development of the default parsing functions.

The default parameter values are:

sosDefaultCharacterEncoding \Sexpr[results=verbatim,stage=render]{sosDefaultCharacterEncoding}

sosDefaultDescribeSensorOutputFormat \Sexpr[results=text,stage=render]{sosDefaultDescribeSensorOutputFormat}

sosDefaultGetCapSections \Sexpr[results=text,stage=render]{sosDefaultGetCapSections}

sosDefaultGetCapAcceptFormats \Sexpr[results=text,stage=render]{sosDefaultGetCapAcceptFormats}

sosDefaultGetCapOwsVersion \Sexpr[results=text,stage=render]{sosDefaultGetCapOwsVersion}

sosDefaultGetObsResponseFormat \Sexpr[results=text,stage=render]{sosDefaultGetObsResponseFormat}

sosDefaultTimeFormat \Sexpr[results=text,stage=render]{sosDefaultTimeFormat}

sosDefaultFilenameTimeFormat \Sexpr[results=text,stage=render]{sosDefaultFilenameTimeFormat}

sosDefaultTempOpPropertyName \Sexpr[results=text,stage=render]{sosDefaultTempOpPropertyName}

sosDefaultTemporalOperator \Sexpr[results=text,stage=render]{sosDefaultTemporalOperator}

sosDefaultSpatialOpPropertyName \Sexpr[results=text,stage=render]{sosDefaultSpatialOpPropertyName}

The **default parsing functions** can be replaced for a variety of XML elements, so that you only need to replace the parts of the parsing that really must be changed. Be aware that inclusion and exclusion are performed after merging the given functions with the defaults!

Example Services: This list contains a few SOS instances that were tested (to different degress) with **sos4R**. The package authors do not maintain these services, so no guarantee can be given that these are usable.

Value

The default value of the respective setting or parameter. This can be a list, especially a named list of functions.

References

Constants

Examples

```
# simple default values
show(sosDefaultCharacterEncoding)
show(sosDefaultDescribeSensorOutputFormat)
show(sosDefaultGetCapAcceptFormats)
show(sosDefaultGetCapOwsVersion)
show(sosDefaultGetCapSections)
show(sosDefaultGetObsResponseFormat)
show(sosDefaultSpatialOpPropertyName)
show(sosDefaultTempOpPropertyName)
show(sosDefaultTemporalOperator)
show(sosDefaultTimeFormat)
SosDefaultBinding()

## Not run:
# usage of defaults in construction method for SOS class
sos <- SOS("http://mysos.com/sos", binding = SosDefaultBinding(),
timeFormat = sosDefaultTimeFormat)

## End(Not run)

# functions to disable all parsing
SosDisabledParsers()

# Replace a parsing function
myER <- function(xml) {
return("EXCEPTION!!!11")
}
SosParsingFunctions("ExceptionReport" = myER)

# use inclusion and exclusion, important: even the just added function needs to
# be included manually!
SosParsingFunctions("ExceptionReport" = myER,
include = c("GetObservation", "DescribeSensor", "ExceptionReport"))
SosParsingFunctions(exclude = c("GetObservation", "DescribeSensor"))

## Not run:
# Replace an encoding function
myEncoding <- function(object, v) {
return(str(object))
}

sos = SOS(url = "http://mysos.com/sos",
```

```
encoders = SosEncodingFunctions("POST" = myPostEncoding))

# Use custom converting function and connection method. This mechanism works the
# same for encoders and decoders.
myConverters <- SosDataFieldConvertingFunctions(
"myNumericUnit" = sosConvertDouble,
mySos <- SOS(sos.url, binding = "KVP", dataFieldConverters = myConverters)
sosDataFieldConverters(mySos)

# inspecting XML using dummy parsing function
sos = SOS(url = "http://mysos.com/sos", parsers = SosDisabledParsers)
describeSensor(sos, sosProcedures(sos)[[1]])

## End(Not run)

# a list of example services
SosExampleServices()

# a named list of all defaults
SosDefaults()

# replace the parsing functions with the default ones
## Not run:
sos <- SosResetParsingFunctions(sos)

## End(Not run)

# parsing options used by xml2
SosDefaultParsingOptions()
```

`DescribeSensor` *Class and Construction Function for "SosDescribeSensor"*

Description

The DescribeSensor Operatiosn of a Sensor Observation Service can be used to retrieve metadata of procedures that are available from a SOS. This sensor description is normally encoded in SensorML.

Please also consult the specification for details on possible contents of the request.

This functions should not be called directly, but instead using the function `describeSensor`.

Usage

```
SosDescribeSensor(service, version, procedure, outputFormat)
```

Arguments

`service`	The service attribute of the request, e.g. '`SOS`'.
`version`	The version attribute of the request, e.g. '`1.0.0`'.

procedure	The value of the procedure elements in the request, e.g. 'urn:procedure:42'.
outputFormat	The value of the output format element in the request, e.g. 'text/xml;subtype="sensorML/1.0.1"'

Value

The value of the construction function is an object of class SosDescribeSensor-class

Objects from the Class

Objects can be created by calling the construction function of the form DescribeSensor(...). The contain the procedure identifier that is to be described by a service.

Slots

procedure: Object of class "character", the identifier of the procedure.

outputFormat: Object of class "character", the requested output format.

service: Object of class "character", the service type, e.g. "SOS".

request: Object of class "character", the name of the request, "DescribeSensor".

version: Object of class "character", the service version, e.g. "1.0.0"

Extends

Class "OwsServiceOperation", directly.

Methods

checkRequest signature(service = "SOS",operation = "DescribeSensor",verbose = "logical"): Checks the contents of the request before sending it.

encodeRequestKVP signature(obj = "DescribeSensor"): Encode the information in the request as key-value-pairs for HTTP GET connections, see encodeRequestKVP-methods.

encodeRequestSOAP signature(obj = "DescribeSensor"): Encode the information in the request as XML for SOAP connections, see encodeRequestSOAP-methods.

encodeRequestXML signature(obj = "DescribeSensor"): Encode the information in the request as XML for HTTP POST connections, see encodeRequestXML-methods.

show signature(object = "DescribeSensor"): Show a human readable excerpt of the contents of the object.

Author(s)

Daniel Nuest <daniel.nuest@uni-muenster.de>

References

See OGC 06-009r6 section 8.4, or the XSD schema file at http://schemas.opengis.net/sos/1.0.0/sosDescribeSensor.xsd.

See Also

See Also SensorML and describeSensor.

Examples

```
showClass("SosDescribeSensor")

# example for construction function
describeSensorRequest <- SosDescribeSensor(service = "SOS", version = "1.0.0",
procedure = "urn:procedure:42", outputFormat = "text/xml")
print(describeSensorRequest)

# encode the request in XML
sos <- SOS_Test()
encodeRequestXML(describeSensorRequest, sos)
toString(encodeRequestXML(describeSensorRequest, sos))

# request a sensor description
mySOS <- SOS(url = "http://sensorweb.demo.52north.org/sensorwebtestbed/service/kvp",
             binding = "KVP")
mySensor <- describeSensor(sos = mySOS,
procedure = sosProcedures(mySOS)[[1]],
outputFormat = 'text/xml; subtype="sensorML/1.0.1"', # space is needed!
)
```

---

describeSensor-methods

Method for a DescribeSensor Request to a SOS

Description

This method sends a DescribeSensor request for a description of the given procedure to the given Sensor Observation Service instance.

Methods

`signature(sos = "SOS", procedure = "character")` Method requests a description of the given procedure from the given SOS.

encodeKVP-methods *Encode Classes as KVP*

Description

These methods convert a given object to a key-value-pair representation to be used in GET requests. The given instance of SOS is possibly used for encoding sub-elements or accessing metadata which is required for the encoding, like time stamp format.

Methods

`signature(obj = "OgcBinaryTemporalOp", sos = "SOS")` Convert the given object to a KVP representation.

`signature(obj = "SosEventTime", sos = "ANY")` Convert the given object to a KVP representation.

`signature(obj = "POSIXt", sos = "ANY")` Convert the given object to a string format suitable for KVP representation.

See Also

SosBindings, https://en.wikipedia.org/wiki/Key_Value_Pair

encodeRequestKVP-methods
Methods for Encoding Requests to SOS in KVP Format

Description

These methods encode objects representing requests to a Sensor Observation Service into a key-value-pair format which can be used in the GET binding, see SosBindings.

Methods

`signature(obj = "SosDescribeSensor")` Encode a DescribeSensor request.

`signature(obj = "SosGetObservation")` Encode a GetObservation request.

`signature(obj = "SosGetObservationById")` Encode GetObservationById request.

`signature(obj = "OwsGetCapabilities")` Dispatching method, checks the version attribute and forwards the encoding to the appropriate method. This method should be called rather than calling the versioned methods directly!

`signature(obj = "OwsGetCapabilities_1.1.0")` Encode GetCapabilities request with OWS version 1.1.0.

`signature(obj = "OwsGetCapabilities_2.0.0")` Encode GetCapabilities request with OWS version 2.0.0.

---

encodeRequestSOAP-methods

Methods for Encoding Requests to SOS in SOAP Format

Description

These methods encode objects representing requests to a Sensor Observation Service into a SOAP message format to be used in the SOAP binding (see SosBindings).

Methods

`signature(obj = "SosDescribeSensor")` Encode a DescribeSensor operation.

`signature(obj = "SosGetObservation")` Encode a GetObservation operation.

`signature(obj = "SosGetObservationById")` Encode a GetObservationById operation.

`signature(obj = "OwsGetCapabilities")` Encode a GetCapabilities operation.

See Also

SosBindings ,encodeXML

---

encodeRequestXML-methods

Methods for Encoding Requests to SOS in XML Format

Description

These methods encode objects representing requests to a Sensor Observation Service into a XML format wich can be used in the POST binding, see SosBindings.

Methods

`signature(obj = "SosDescribeSensor")` Encode a DescribeSensor request.

`signature(obj = "SosGetObservation")` Encode a GetObservation request.

`signature(obj = "SosGetObservationById")` Encode a GetObservationById request.

`signature(obj = "OwsGetCapabilities")` Encode a GetCapabilities request.

References

Annex A (normative) "SOS Schema" in Na, A., Priest, M. (Eds.), Sensor Observation Service, Open Geospatial Consortium Inc., OGC 06-009r6, Version: 1.0.

See Also

SosBindings, encodeXML

encodeXML-methods *Encode Classes as XML*

Description

These methods convert the given objects to XML representations for HTTP POST requests. The given instance of SOS is possibly used for encoding sub-elements or accessing metadata which is required for the encoding, like time stamp format.

Methods

signature(obj = "GmlDirectPosition", sos = "SOS") Convert the given object to an XML representation.

signature(obj = "GmlEnvelope", sos = "SOS") Convert the given object to an XML representation.

signature(obj = "GmlLineString", sos = "SOS") Convert the given object to an XML representation.

signature(obj = "GmlPointProperty", sos = "SOS") Convert the given object to an XML representation.

signature(obj = "GmlPoint", sos = "SOS") Convert the given object to an XML representation.

signature(obj = "GmlPolygon", sos = "SOS") Convert the given object to an XML representation.

signature(obj = "GmlTimeInstantProperty", sos = "SOS") Convert the given object to an XML representation.

signature(obj = "GmlTimeInstant", sos = "SOS") Convert the given object to an XML representation.

signature(obj = "GmlTimePeriod", sos = "SOS") Convert the given object to an XML representation.

signature(obj = "GmlTimePosition", sos = "SOS") Convert the given object to an XML representation.

signature(obj = "POSIXt", sos = "SOS") Converts the time object to a string that is suitable to be used as the value of XML time elements.

signature(obj = "OgcBBOX", sos = "SOS") Convert the given object to an XML representation.

signature(obj = "OgcComparisonOps", sos = "SOS") Convert the given object to an XML representation.

signature(obj = "OgcContains", sos = "SOS") Convert the given object to an XML representation.

signature(obj = "OgcIntersects", sos = "SOS") Convert the given object to an XML representation.

signature(obj = "OgcOverlaps", sos = "SOS") Convert the given object to an XML representation.

`signature(obj = "SosEventTime", sos = "SOS")` Convert the given object to an XML representation.

`signature(obj = "SosFeatureOfInterest", sos = "SOS")` Convert the given object to an XML representation.

`signature(obj = "TM_After", sos = "SOS")` Convert the given object to an XML representation.

`signature(obj = "TM_Before", sos = "SOS")` Convert the given object to an XML representation.

`signature(obj = "TM_During", sos = "SOS")` Convert the given object to an XML representation.

`signature(obj = "TM_Equals", sos = "SOS")` Convert the given object to an XML representation.

`signature(obj = "XMLNode", sos = "SOS")` Convert the given object to an XML representation.

References

XML specification: http://www.w3.org/XML/.

See Also

encodeRequestXML, encodeRequestSOAP

FoiOrNULL-class *Class* "FoiOrNULL"

Description

Classes and construction functions for features of interest from the OGC Observations and Measurements specification. A feature of interest is considered to be the real world feature that is observed.

Objects from the Class

A virtual Class: No objects may be created from it.

Methods

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

Examples

```
showClass("FoiOrNULL")
```

getCapabilities-methods *Request Capabilities from a SOS*

Description

This method request the metadata description of a given Sensor Observation Service, the Capabilities document.

Methods

`signature(sos = "SOS")` Request capabilities description from a SOS.

`signature(inspect = "logical")` Print out the sent and received documents to console.

`signature(verbose = "logical")` Extensive debugging information printed to console.

getFeatureOfInterest *Function retrieving features of interest, i.e. the representations of the real world features that are observed and for which observations are provided.*

Description

This function retrieves , i.e. the representations of the real world features that are observed and for which observations are provided, from a Sensor Observation Service.

Usage

```
getFeatureOfInterest(sos, featureOfInterest, verbose = sos@verboseOutput,
                     inspect = FALSE, saveOriginal = FALSE)
```

Arguments

`sos`	The Sensor Observation Service from which features of interest should be retrieved
`featureOfInterest`	identifier(s) of features of interest
`verbose`	A boolean value indicating whether debug information is printed out to the console during the execution
`inspect`	A boolean value to enable printing of the sent request and received response to the console
`saveOriginal`	Save the received document to the current working directory. If `TRUE` a filename is automatically generated, if a character string is given it is used as the filename

Value

An object of class SosGetFeatureOfInterest_2.0.0-class.

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.
```

`GetObservation` *GetObservation and GetObservationById Request Objects*

Description

Classes (and their construction functions) to request observations from a Sensor Observation Service.

Usage

```
SosGetObservation(service, version, offering, observedProperty, responseFormat,
                  srsName = as.character(NA), eventTime = list(),
                  procedure = as.character(NA), featureOfInterest = NULL,
                  result = NULL, resultModel = as.character(NA),
                  responseMode = as.character(NA), BBOX = as.character(NA),
                  valueReferenceTemporalFilter = as.character(NA))
SosGetObservationById(service, version, observationId, responseFormat,
                    srsName = as.character(NA), resultModel = as.character(NA),
                      responseMode = as.character(NA))
```

Arguments

`service`	The service attribute of the request, e.g. '`SOS`'.
`version`	The version attribute of the request, e.g. '`1.0.0`'.
`observationId`	The value of the ObservationId element in the request, e.g. '`o_12345`', which is to be optained. This could have been obtained by the client via a URL in a feed, alert, or some other notification.
`offering`	The offering element value in the request, e.g. '`"temperatures"`'. All other parameters are depending on the selected offering.
`observedProperty`	A list of values for observedProperty elements in the request, e.g. '`"urn:property:AirTemperature"`'. IDs of phenomena are advertised in capabilities document.
`responseFormat`	The responseFormat element value in the request, e.g. '`text/xml;subtype="om/1.0.0"`'. ID of the output format to be used for the requested data. The supported output formats are listed in the selected offering capabilities.

`srsName` The srsName attribute of the request, e.g. '`urn:ogc:def:crs:EPSG:4326`'.

`eventTime` A list of objects of class SosEventTime-class which are added as eventTime elements to the request. Allows a client to request observations from a specific instant, multiple instances or periods of time in the past, present and future. The supported range is listed in the selected offering capabilities.

`procedure` A list of procedure identifiers added to the request as procedure elements.

`featureOfInterest`
An object of class SosFeatureOfInterest added to the request as the featureOfInterest element, or `NULL`. Specifies target feature for which observations are requested.

`result` An object of class OgcComparisonOps-class added to the request as result element, or `NULL`, or any element that can be encoded using `encodeXML(...)` and then be added to an XML document with `addChildren(...)`. Filtering: Only report observations where the result matches this expression.

`resultModel` The resultModel element of the request, e.g. '`om:Measurement`', which is an identifier of the result model to be used for the requested data. The resultModel values supported by a service are listed in the contents section of the service metadata, identified as QName values.

`responseMode` The responseMode element of the request, e.g. '`inline`', which allows the client to request the form of the response.

`BBOX` A bounding box to be used only with KVP encoding in request via HTTP GET, in the format '`minlon,minlat,maxlon,maxlat,srsURI?`', with the spatial reference system being optional. This element is ignored for POST requests, use the parameter `featureOfInterest` instead, see `SosBindings`.

`valueReferenceTemporalFilter`
The property name used in a temporal filter for SOS 2.0 KVP reuests, ignore for SOS 1.0.0.

Details

Please consult the specification for details on possible contents of the request.

Value

An object of class SosGetObservation-class or SosGetObservationById-class respectively.

Objects from the Class

Objects can be created by calls to the construction functions of the form `SosGetObservationById(...)` or `SosGetObservationById(...)`.

Slots

`BBOX`**:** Object of class `"character"`, specifies a bounding box for spatial filtering to be applied in GET requests (only), see SosBindings

`eventTime`**:** Object of class `"list"`, specifies the time period(s) for which observations are requested.

`featureOfInterest`: Object of class `"SosFeatureOfInterestOrNULL"`, specifies the feature for which observations are requested. This can either be represented by a reference to a feature ID advertised in the capabilities document or can be a spatial constraint

`observationId`: Object of class `"character"`, the Id of the requested observation.

`observedProperty`: Object of class `"list"`, specifies the phenomenon or phenomena for which observations are requested.

`offering`: Object of class `"character"`, specifies the offering URI advertised in the GetCapabilities document.

`procedure`: Object of class `"list"`, procedure parameter specifies the sensor system(s) for which observations are requested.

`request`: Object of class `"character"`, the name of the request.

`responseFormat`: Object of class `"character"`, specifies the desired resultFormat MIME content type for transport of the results.

`responseMode`: Object of class `"character"`, specifies whether results are requested in-line, outof-band, as an attachment, or if this is a request for an observation template that will be used for subsequent calls to GetResult.

`resultModel`: Object of class `"character"`, specifies the QName of the root element of an O&M Observation or element in the appropriate substitution group.

`result`: Object of class `"ANY"`, provides a place to put in OGC filter expressions based on property values. This instructs the SOS to only return observations where the result matches this expression.

`service`: Object of class `"character"`, service type identifier.

`srsName`: Object of class `"character"`, defines the spatial reference system that should be used for any geometries that are returned in the response. This must be one of the advertised values in the offering specified in gml:srsName elements.

`valueReferenceTemporalFilter`: Object of class `"character"`, the property name used in a temporal filter for SOS 2.0 KVP reuests.

`version`: Object of class `"character"`, specification version for operation.

Extends

SosGetObservation: Class `"OwsServiceOperation"`, directly.

SosGetObservationById: Class `"OwsServiceOperation"`, directly.

Methods

checkRequest `signature(service = "SOS",operation = "SosGetObservationById",verbose = "logical")` or `signature(service = "SOS",operation = "GetObservation",verbose = "logical")`: Check the request for validity and for compliance with the metadata available in from the given SOS.

encodeRequestKVP `signature(obj = "SosGetObservationById")` or `signature(obj = "GetObservation")`: Transform the information to key-value pair format, see `encodeKVP`.

encodeRequestXML `signature(obj = "SosGetObservationById")` or `signature(obj = "GetObservation")`: Transform the information to XML format, see `encodeXML`.

encodeRequestSOAP `signature(obj = "SosGetObservation")`: Transform the information to XML format for SOAP requests.

Author(s)

Daniel Nuest <daniel.nuest@uni-muenster.de>

References

See OGC 06-009r6 section 8.4, or the XSD schema file at http://schemas.opengis.net/sos/1.0.0/sosGetObservation.xsd.

See OGC 06-009r6 section 10.1, or the XSD schema file at http://schemas.opengis.net/sos/1.0.0/sosGetObservationById.xsd.

See Also

SosGetObservation-class, SosGetObservationById-class

Examples

```
showClass("SosGetObservation")
showClass("SosGetObservationById")

responseFormat <- "text/xml;subtype=&quot;om/1.0.0&quot;"

obsReq <- SosGetObservation(service = "SOS", version = "1.0.0",
                            offering = "temperatures",
                            observedProperty = list("urn:property:AirTemperature"),
                            responseFormat = responseFormat)
print(obsReq)

obsByIdReq <- SosGetObservationById(service = "SOS", version = "1.0.0",
                                    observationId = "o_12345",
                                    responseFormat = responseFormat)
print(obsByIdReq)

## Not run:
sos <- SOS("http://mysos.net/sos")
encodeXML(obsByIdReq, sos = sos)

## End(Not run)
```

---

getObservation-methods

Request Observations

Description

This method sends a GetObservation request to the given SOS.

Details

It takes a variety of inputs, of which only the offering is mandatory for GetObservation request, and the observationId for GetObservationByIdRequests.

Methods

`signature(sos = "SOS", offering = "SosObservationOffering")` **or** `signature(sos = "SOS", offering = "charact`
Request observation data from the given SOS for the given offering (either character identifier or an object of class `SosObservationOffering`).

`signature(sos = "SOS", observationId = "character")` Request observation data from the given SOS for the given observation identifier.

Examples

```
## Not run:
# request observations
mySOSpox <- SOS(url = "http://sensorweb.demo.52north.org/sensorwebtestbed/service/pox",
             binding = "POX", useDCPs = FALSE)

myOffering <- sosOfferings(mySOS)[["ws2500"]]
period <- sosCreateTimePeriod(sos = mySOS,
begin = as.POSIXct("2015/11/01"),
end = as.POSIXct("2015/11/02"))
eventTime <- sosCreateEventTimeList(period)

nov2015 <- getObservation(sos = mySOSpox,
                          offering = myOffering,
                          eventTime = eventTime)

# request observation by identifier and get the data
obsId <- getObservationById(sos = mySOSpox,
                            observationId = "http://www.52north.org/test/observation/1")
sosResult(obsId, coordinates = TRUE)

## End(Not run)
```

GmlDirectPosition-class

Classes and Construction Functions from the GML Namespace

Description

Classes for `GML` elements and their respective construction functions. See the referenced specification for details.

Usage

```
GmlDirectPosition(pos, srsName = as.character(NA), srsDimension = NA_integer_,
                  axisLabels = as.character(NA), uomLabels = as.character(NA))
GmlDirectPositionLatLon(lat, lon, srsName = as.character(NA),
                        srsDimension = NA_integer_,
                        axisLabels = as.character(NA),
                        uomLabels = as.character(NA))
GmlEnvelope(lowerCorner, upperCorner, srsName = as.character(NA),
            srsDimension = NA_integer_, axisLabels = as.character(NA),
            uomLabels = as.character(NA))
GmlFeatureCollection(featureMembers, id = as.character(NA))
GmlPoint(pos, id = as.character(NA), srsName = as.character(NA),
         srsDimension = NA_integer_, axisLabels = as.character(NA),
         uomLabels = as.character(NA))
GmlPointProperty(href = as.character(NA), point = NULL)
GmlFeatureProperty(href = as.character(NA), feature = NULL)
GmlTimeInstant(timePosition, id = as.character(NA), relatedTimes = list(NA),
               frame = as.character(NA))
GmlTimeInstantProperty(href = as.character(NA), time = NULL)
GmlTimeInterval(interval, unit, radix = NA, factor = NA)
GmlTimePeriod(begin = NULL, beginPosition = NULL, end = NULL,
              endPosition = NULL, duration = as.character(NA),
              timeInterval = NULL, id = as.character(NA),
              relatedTimes = list(NA), frame = as.character(NA))
GmlTimePosition(time, frame = as.character(NA),
                calendarEraName = as.character(NA),
                indeterminatePosition = as.character(NA))
GmlMeasure(value, uom)
```

Arguments

axisLabels	See corresponding slot description.
begin	See corresponding slot description.
beginPosition	See corresponding slot description.
calendarEraName	See corresponding slot description.
duration	See corresponding slot description.
end	See corresponding slot description.
endPosition	See corresponding slot description.
factor	See corresponding slot description.
feature	See corresponding slot description.
featureMembers	See corresponding slot description.
frame	See corresponding slot description.
href	See corresponding slot description.
id	See corresponding slot description.

`indeterminatePosition`
: See corresponding slot description.

`interval`
: See corresponding slot description.

`lat`
: Latitude coordinate parameter.

`lon`
: Longitue coordinate parameter.

`lowerCorner`
: See corresponding slot description.

`point`
: See corresponding slot description.

`pos`
: See corresponding slot description.

`radix`
: See corresponding slot description.

`relatedTimes`
: See corresponding slot description.

`srsDimension`
: See corresponding slot description.

`srsName`
: See corresponding slot description.

`time`
: See corresponding slot description.

`timeInterval`
: See corresponding slot description.

`timePosition`
: See corresponding slot description.

`unit`
: See corresponding slot description.

`uomLabels`
: See corresponding slot description.

`upperCorner`
: See corresponding slot description.

`value`
: See slot description.

`uom`
: See slot description.

Details

The "...OrNULL" classes are used to model optional slots.

Value

The construction functions return an object of the respective class.

Objects from this classes

Objects can be created by calling the according construction functions. e.g. in the form `GmlPoint(...)`.

Extends

GmlFeature: Class "`GmlFeatureOrNULL`", directly.

GmlFeatureCollection: Class "`GmlFeature`", directly. Class "`GmlFeatureOrNULL`", by class "GmlFeature", distance 2.

GmlLineString: Class "`GmlGeometry`", directly.

GmlPoint: Class "`GmlGeometry`", directly.

GmlTimeGeometricPrimitive: Class "`GmlTimePrimitive`", directly. Class "`GmlTimeObject`", by class "GmlTimePrimitive", distance 2. Class "`GmlTimeObjectOrNULL`", by class "GmlTimePrimitive", distance 3.

GmlTimeInstant: Class "GmlTimeGeometricPrimitive", directly. Class "GmlTimeInstantOrNULL", directly. Class "GmlTimePrimitive", by class "GmlTimeGeometricPrimitive", distance 2. Class "GmlTimeObject", by class "GmlTimeGeometricPrimitive", distance 3. Class "GmlTimeObjectOrNULL", by class "GmlTimeGeometricPrimitive", distance 4.

GmlTimeInstantProperty: Class "GmlTimeGeometricPrimitive", directly. Class "GmlTimeInstantPropertyOrNULL", directly. Class "GmlTimePrimitive", by class "GmlTimeGeometricPrimitive", distance 2. Class "GmlTimeObject", by class "GmlTimeGeometricPrimitive", distance 3. Class "GmlTimeObjectOrNULL", by class "GmlTimeGeometricPrimitive", distance 4.

GmlTimeInterval: Class "GmlTimeIntervalOrNULL", directly.

GmlTimeObject: Class "GmlTimeObjectOrNULL", directly.

GmlTimePrimitive: Class "GmlTimeObject", directly. Class "GmlTimeObjectOrNULL", by class "GmlTimeObject", distance 2.

Virtual Classes

No objects may be created from the following virtual classes: GmlFeature, GmlTimeObject, GmlTimePrimitive, GmlTimeGeometricPrimitive, GmlGeometry, all ...OrNULL classes.

Slots

axisLabels: Object of class "character": An character vector of labels for all the axes of this CRS.

begin: Object of class "GmlTimeInstantPropertyOrNULL": An object of class GmlTimeInstantProperty-class, the start time of a GmlTimePeriod-class.

beginPosition: Object of class "GmlTimePositionOrNULL": An object of class GmlTimePosition-class, the start time of a GmlTimePeriod-class.

calendarEraName: Object of class "character": The name of the calendar era.

duration: Object of class "character": Duration of an interval using ISO 8601 syntax for temporal length.

end: Object of class "GmlTimeInstantPropertyOrNULL": An object of class GmlTimeInstantProperty-class, the end time of a GmlTimePeriod-class.

endPosition: Object of class "GmlTimePositionOrNULL": An object of class GmlTimePosition-class, the end time of a GmlTimePeriod-class.

exterior: Object of class "ANY" for future use in GmlPolygon.

factor: Object of class "integer": Factor parameter for a GmlTimeInterval-class. The resolution of the time interval is to one radix ^(-factor) of the specified time unit.

feature: Object of class "GmlFeatureOrNULL": The directly contained feature in a GmlFeature-class.

featureMembers: Object of class "list": A list of GmlFeature-class in a GmlFeatureCollection-class.

frame: Object of class "character": Provides a URI reference that identifies a description of the reference system.

href: Object of class "character": Reference to a property.

id: Object of class "character": The identifier/id attribute (gml:id).

`indeterminatePosition`: Object of class `"character"`: Inexact temporal positions may be expressed using the optional indeterminatePosition parameter. This takes one of the following values: after, before, now, unknown.

`interior`: Object of class `"ANY"` for future use in GmlPolygon.

`interval`: Object of class `"character"`: Interval parameter for a `GmlTimeInterval-class`.

`lowerCorner`: Object of class `"GmlDirectPosition"`: Object of class `GmlDirectPosition-class`, the lower (left) corner of an `GmlEnvelope-class`.

`point`: Object of class `"ANY"`: An object of class `GmlPoint` in a `GmlPointProperty-class`.

`points`: Object of class `"list"` A list of positions for future use in GmlLineString.

`posList`: Object of class `"ANY"`: A list of positions for future use in GmlLineString.

`pos`: Object of class `"character"`: Character string containing the coordinates in a `GmlDirectPosition-class`, or a `GmlDirectPosition-class` in a `GmlPoint-class`.

`poss`: Object of class `"list"`: A list of positions for future use in GmlLineString.

`radix`: Object of class `"integer"`: Radix parameter for a `GmlTimeInterval-class`. The resolution of the time interval is to one radix ^(-factor) of the specified time unit.

`relatedTimes`: Object of class `"list"`: List of related times.

`srsDimension`: Object of class `"integer"`: Dimensions of the coordinate reference system as stated in the coordinate reference system definition., e.g. '2'.

`srsName`: Object of class `"character"`: Name of the spatial reference system for bounding box, e.g. '`"urn:ogc:def:crs:EPSG:4326"`'.

`timeInterval`: Object of class `"GmlTimeIntervalOrNULL"`: Time interval element in a `GmlTimePeriod-class`, an object of class `GmlTimeInterval-class`.

`time`: Object of class `"GmlTimeInstantOrNULL"`: The actual time as an object of class `POSIXt` in a `GmlTimePosition-class`, or an object of class `GmlTimeInstant-class` in a `GmlTimeInstantProperty-class`.

`timePosition`: Object of class `"GmlTimePosition"`: An object of class `GmlTimePosition-class` in a `GmlTimeInstant-class`.

`unit`: Object of class `"character"`: Unit parameter for a `GmlTimeInterval-class`.

`uomLabels`: Object of class `"character"`: Unit of measurement labels as an ordered character vector for the axes in a bounding box, e.g. '`"deg"`'.

`upperCorner`: Object of class `"GmlDirectPosition"`: Object of class `GmlDirectPosition-class`, the upper (right) corner of an `GmlEnvelope-class`.

`uom`: Object of class `"character"`: The unit of measurement in a `OmMeasure` object.

`uom`: Object of class `"character"`: The unit of measurement in a `GmlMeasure` object.

`value`: Object of class `"numeric"`: The actual value in a `GmlMeasure` object.

Methods

encodeXML `signature(obj = "GmlDirectPosition", sos = "SOS")` or `signature(obj = "GmlEnvelope", sos = "SOS")` and more: Convert the given element to an XML respresentaion, and XML "encoding".

Author(s)

Daniel Nuest <daniel.nuest@uni-muenster.de>

References

GML specification at https://www.opengeospatial.org/standards/gml.

Examples

```
showClass("GmlDirectPosition")
showClass("GmlEnvelope")
showClass("GmlFeature")
showClass("GmlFeatureCollection")
showClass("GmlFeatureOrNULL")
showClass("GmlFeatureProperty")
showClass("GmlGeometry")
showClass("GmlLineString")
showClass("GmlPoint")
showClass("GmlPointProperty")
showClass("GmlPolygon")
showClass("GmlTimeGeometricPrimitive")
showClass("GmlTimeInstant")
showClass("GmlTimeInstantOrNULL")
showClass("GmlTimeInstantProperty")
showClass("GmlTimeInstantPropertyOrNULL")
showClass("GmlTimeInterval")
showClass("GmlTimeIntervalOrNULL")
showClass("GmlTimeObject")
showClass("GmlTimeObjectOrNULL")
showClass("GmlTimePeriod")
showClass("GmlTimePosition")
showClass("GmlTimePositionOrNULL")
showClass("GmlTimePrimitive")

# create direct position
pos1 <- GmlDirectPosition(pos = "7.0 52.0")
show(pos1)

# create envelope
env1 <- GmlEnvelope(upperCorner = pos1,
                    lowerCorner = GmlDirectPosition("6.0 51.0"))
print(env1)

# wrap elements in feature collection
GmlFeatureCollection(id = "001", featureMembers=list(pos1, env1))

# create point with ID
point1 <- GmlPoint(pos = pos1, id = "002")

# create point properties
GmlPointProperty(href = "http://link.to/point")
GmlPointProperty(point = point1)

# time interval of one day
GmlTimeInterval(interval = "1", unit = "d")
```

```
# referenced feature
GmlFeatureProperty(href = "http://link.to/feature")

# create a time position and wrap it into a time instant
timePos1 <- GmlTimePosition(time = as.POSIXct("2010-01-01"))

# create direct or referenced time instant
timeInst1 <- GmlTimeInstant(timePosition = timePos1)
timeInst1

GmlTimeInstantProperty(href = "http://link.to/timeInstant")

# create different variants of time periods
# one hour with time positions
GmlTimePeriod(beginPosition = timePos1,
              endPosition = GmlTimePosition(time = timePos1@time+3600))

# one week backwards from now
timePos <- GmlTimePosition(time = Sys.time()-(3600*24*7))
aWeekAgo <- GmlTimeInstantProperty(time = GmlTimeInstant(time = timePos))
timePos <- GmlTimePosition(time = Sys.time())
now <- GmlTimeInstantProperty(time = GmlTimeInstant(time = timePos))
GmlTimePeriod(begin = aWeekAgo, end = now)
```

KML *Methods for the Namespace kml*

Description

Methods for handling responses in Keyhole Markup Language (KML) format.

At the current stage, the parsers simply returns an object from the package **XML** with the KML document.

Author(s)

Daniel Nuest <daniel.nuest@uni-muenster.de>

References

https://www.opengeospatial.org/standards/kml https://developers.google.com/kml/documentation/

Examples

```
#
```

MonitoringPoint-class *Class* "MonitoringPoint"

Description

A monitoring point is the feature of interest defined for WaterML observations, i.e. a monitoring point represents the real world feature for which observations are taken. This may be, for example, the position of a stream flow sensor at a river.

Objects from the Class

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

Slots

sampledFeatures: Object of class "list" ~~

id: Object of class "character" ~~

identifier: Object of class "list" ~~

names: Object of class "list" ~~

shape: Object of class "SamsShape" ~~

Extends

Class "GmlFeature", directly. Class "FoiOrNULL", directly. Class "GmlFeatureOrNULL", by class "GmlFeature", distance 2. Class "GmlFeatureOrGmlFeaturePropertyOrNULL", by class "GmlFeature", distance 2.

Methods

print `signature(x = "MonitoringPoint")`: ...

show `signature(object = "MonitoringPoint")`: ...

toString `signature(x = "MonitoringPoint")`: ...

Examples

```
showClass("MonitoringPoint")
```

OGC *Classes and Construction Functions for the OGC Namespace*

Description

These classes represent elements from the OpenGIS(R) Filter Encoding Implementation Specification that are used in requests to Sensor Observation Services.

Usage

```
OgcBBOX(propertyName = sosDefaultSpatialOpPropertyName, envelope)
OgcContains(propertyName = sosDefaultSpatialOpPropertyName, geometry = NULL,
            envelope = NULL)
OgcIntersects(propertyName = sosDefaultSpatialOpPropertyName, geometry = NULL,
              envelope = NULL)
OgcOverlaps(propertyName = sosDefaultSpatialOpPropertyName, geometry = NULL,
            envelope = NULL)
```

Arguments

	Arguments for the construction functions are as follows.
	The value for the propertyName attribute.
geometryName	The geometry to be used in a spatial filter.
envelope	The geometry to be used in a sptial filter.

Details

These comprise spatial and temporal operations and operators which can be encoded in different ways.

The ...OrNULL classes are used to model optional slots.

Value

The value of the construction functions is an object of the respective class.

Objects from the Class

Objects can be created by calls to the respective construction functions of the form `OgcBBOX( ...)`, `OgcContains(...)`, or `OgcIntersects`.

The following classes are virtual, no objects may be created from them: `OgcBinaryTemporalOp`, `OgcBinaryTemporalOpOrNULL`, `OgcComparisonOps`, codeOgcComparisonOpsOrNULL, `OgcSpatialOps`, `OgcSpatialOpsOrNULL`.

Slots

`propertyName`: Object of class `"character"`, the value of the propertyName attribute.

`geometry`: Object of class `"GmlGeometry"`, a geometry contained in a spatial filter.

`envelope`: Object of class `"GmlEnvelope"`, an envelope contained in a spatial filter.

`time`: Object of class `"GmlTimeGeometricPrimitive"`, a time element contained in a temporal filter.

Extends

OgcBBOX, OgcBinarySpatialOp: Class `"OgcSpatialOps"`, directly. Class `"OgcSpatialOpsOrNULL"`, by class "OgcSpatialOps", distance 2.

OgcBinaryTemporalOp: Class `"OgcBinaryTemporalOpOrNULL"`, directly.

OgcContains, OgcIntersects, OgcOverlaps: Class `"OgcBinarySpatialOp"`, directly. Class `"OgcSpatialOps"`, by class "OgcBinarySpatialOp", distance 2. Class `"OgcSpatialOpsOrNULL"`, by class "OgcBinarySpatialOp", distance 3.

OgcSpatialOps: Class `"OgcSpatialOpsOrNULL"`, directly.

Methods

encodeKVP `signature(obj = "OgcBinaryTemporalOp", sos = "SOS")`: Encode the given operation in key-value-pair style, see `encodeKVP`

encodeXML `signature(obj = "OgcBBOX", sos = "SOS")`: Encode the given operation in XML, see `encodeXML`

encodeXML `signature(obj = "OgcComparisonOps", sos = "SOS")`: Encode the given operation in XML, see `encodeXML`

encodeXML `signature(obj = "OgcContains", sos = "SOS")`: Encode the given operation in XML, see `encodeXML`

encodeXML `signature(obj = "OgcIntersects", sos = "SOS")`: Encode the given operation in XML, see `encodeXML`

encodeXML `signature(obj = "OgcOverlaps", sos = "SOS")`: Encode the given operation in XML, see `encodeXML`

Warning

The encoding functions of these classes are not completely implemented yet.

Note

This implementation of the Filter Encoding Specification is not complete.

Author(s)

Daniel Nuest <daniel.nuest@uni-muenster.de>

References

Vretanos, Panagiotis A. (Ed.), OpenGIS(R) Filter Encoding Implementation Specification, OGC 04-095, Version: 1.1.0

Schemas: http://schemas.opengis.net/filter/1.1.0/

Examples

```
showClass("OgcBBOX")
showClass("OgcBinarySpatialOp")
showClass("OgcBinaryTemporalOp")
showClass("OgcBinaryTemporalOpOrNULL")
showClass("OgcComparisonOps")
showClass("OgcContains")
showClass("OgcOverlaps")
showClass("OgcSpatialOps")
showClass("OgcSpatialOpsOrNULL")
```

OmMeasurement *Class and Construction Function for om:Measurement Elements*

Description

Classes and construction functions for objects from the OGC Observations and Measurements specification.

Usage

```
OmMeasurement(samplingTime, procedure, observedProperty, featureOfInterest, result,
            metadata = NA, resultTime = NULL, resultQuality = NA,parameter = NA)
```

Arguments

samplingTime	See slot description.
procedure	See slot description.
observedProperty	See slot description.
featureOfInterest	See slot description.
result	See slot description.
metadata	See slot description.
resultTime	See slot description.
resultQuality	See slot description.
parameter	See slot description.

Details

A Measurement contains a GmlMeasure.

Value

The construction functions return an object of the respective class.

Objects from the Class

Objects can be created by calls to the construction function of the form OmMeasurement(...).

Slots

featureOfInterest: Object of class "GmlFeature": A feature of any type (ISO 19109, ISO 19101), which is a representation of the observation target, being the real-world object regarding which the observation is made.

metadata: Object of class "ANY": Observation metadata.

observedProperty: Object of class "SwePhenomenonProperty": Identifies or describes the phenomenon for which the observation result provides an estimate of its value. It must be a property associated with the type of the feature of interest.

parameter: Object of class "ANY": A general event-specific parameter. This will typically be used to record environmental parameters, or event-specific sampling parameters that are not tightly bound to either the feature-of-interest or the procedure.

procedure: Object of class "ANY": The description of a process used to generate the result. It must be suitable for the observed property.

result: Object of class "ANY": Contains the value generated by the procedure. The type of the observation result must be consistent with the observed property, and the scale or scope for the value must be consistent with the quantity or category type.

resultQuality: Object of class "ANY": Event specific quality of a result.

resultTime: Object of class "GmlTimeObjectOrNULL": The time when the procedure associated with the observation act was applied. For some observations these are identical, in which case the resultTime may be omitted. However, there are important cases where they differ.

samplingTime: Object of class "GmlTimeObjectOrNULL": The time that the result applies to the feature-of-interest. This is the time usually required for geospatial analysis of the result.

Extends

OmMeasurement: Class "OmObservation", directly. Class "OmObservationOrNULL", by class "OmObservation", distance 2.

Methods

sosResult signature(obj = "OmMeasurement"): Get the data in the measurement as a data.frame.

sosGetCRS signature(obj = "OmMeasurement"): Get the coordinate reference system used in the feature of interest.

Author(s)

Daniel Nuest <daniel.nuest@uni-muenster.de>

References

Cox, S. (Ed.), Observations and Measurements - Part 1 - Observation schema, Open Geospatial Consortium Inc., OGC 07-022r1, Version: 1.0

See Also

See also OmObservation-class, GmlMeasure.

Examples

```
showClass("OmMeasurement")
```

OmObservation-class *Classes for om:Observation Elements*

Description

Classes and construction functions for objects from the OGC Observations and Measurements specification.

Usage

```
OmObservation(samplingTime, procedure, observedProperty, featureOfInterest,
              result, metadata = NA, resultTime = NULL, resultQuality = NA,
              parameter = NA)
OmObservationProperty(href = as.character(NA), obs = NULL)
```

Arguments

samplingTime	See slot description.
procedure	See slot description.
observedProperty	See slot description.
featureOfInterest	See slot description.
result	See slot description.
metadata	See slot description.
resultTime	See slot description.
resultQuality	See slot description.
parameter	See slot description.
href	See slot description.
obs	See slot description.

Details

The class `OmObservationProperty` can be used to reference to an (online) observation.

The ...OrNULL classes are used to model optional slots.

Value

The construction functions return an object of the respective class.

Objects from the Class

Objects can be created by calls to the construction functions of the form `OmObservation(...)` and `OmObservationProperty(...)`.

The following classes are virtual, no objects may be created from them: `OmObservationOrNULL-class`.

Slots

`samplingTime`**:** Object of class `"GmlTimeObjectOrNULL"`: The time that the result applies to the feature-of-interest. This is the time usually required for geospatial analysis of the result.

`procedure`**:** Object of class `"ANY"`: The description of a process used to generate the result. It must be suitable for the observed property.

`observedProperty`**:** Object of class `"SwePhenomenonProperty"`: Identifies or describes the phenomenon for which the observation result provides an estimate of its value. It must be a property associated with the type of the feature of interest.

`featureOfInterest`**:** Object of class `"GmlFeature"`: A feature of any type (ISO 19109, ISO 19101), which is a representation of the observation target, being the real-world object regarding which the observation is made.

`result`**:** Object of class `"ANY"`: Contains the value generated by the procedure. The type of the observation result must be consistent with the observed property, and the scale or scope for the value must be consistent with the quantity or category type.

`metadata`**:** Object of class `"ANY"`: Observation metadata.

`resultTime`**:** Object of class `"GmlTimeObjectOrNULL"`: The time when the procedure associated with the observation act was applied. For some observations these are identical, in which case the resultTime may be omitted. However, there are important cases where they differ.

`resultQuality`**:** Object of class `"ANY"`: Event specific quality of a result.

`parameter`**:** Object of class `"ANY"`: A general event-specific parameter. This will typically be used to record environmental parameters, or event-specific sampling parameters that are not tightly bound to either the feature-of-interest or the procedure.

`href`**:** Object of class `"character"`: Referenced observation in a `OmObservationProperty`.

Extends

OmObservation Class `"OmObservationOrNULL"`, directly.

Methods

sosResult `signature(obj = "OmObservation")`: Accessor function for the result slot of an observation or measurement.

Author(s)

Daniel Nuest <daniel.nuest@uni-muenster.de>

References

Cox, S. (Ed.), Observations and Measurements - Part 1 - Observation schema, Open Geospatial Consortium Inc., OGC 07-022r1, Version: 1.0

See Also

See Also as OmMeasurement-class.

Examples

```
showClass("OmObservation")
showClass("OmObservationProperty")
showClass("OmObservationOrNULL")

OmObservationProperty(href = "http://link.to/myObservation")

# get result from an observation
## Not run:
result <- observation@result

# the accessor method also works with lists of observations
result <- sosResult(observation)
resultList <- sosResult(observationList)

## End(Not run)
```

OmObservationCollection

Class "OmObservationCollection"

Description

Collection of arbitrary observations.

Objects from the Class

Objects can be created by calls to the construction function of the form `OmObservationCollection(...)`.

Slots

`boundedBy`**:** Object of class `"list"` containing a representation of the bounding box of the contained observations.

`members`**:** Object of class `"list"` containing objects of class `OmObservation` or `OmMeasurement`.

Methods

length `signature(x = "OmObservationCollection")`: Returns the number of observations in the slot `members`.

sosResult `signature(obj = "OmObservationCollection")`: Extract the result slots from the containted observations or measurements.

Author(s)

Daniel Nuest <daniel.nuest@uni-muenster.de>

References

Cox, S. (Ed.), Observations and Measurements - Part 1 - Observation schema, Open Geospatial Consortium Inc., OGC 07-022r1, Version: 1.0

See Also

`OmObservation` or `OmMeasurement`.

Examples

```
showClass("OmObservationCollection")
```

OmOM_Observation-class

Class "OmOM_Observation"

Description

Classes and construction functions for objects from the OGC Observations and Measurements specification, version 2.0.

Objects from the Class

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

Slots

`phenomenonTime`**:** Object of class `"GmlTimeObjectOrNULL"`: The time that the result applies to the feature-of-interest. This is the time usually required for geospatial analysis of the result. Note: The phenomenonTime has been renamed to samplingTime in the O&M specification version 2.0. However, in order to be consistent with version 1.0 in SOS4R, we decided to leave the slot as is and just changed the parsing/encoding.

`procedure`**:** Object of class `"ANY"`: The description of a process used to generate the result. It must be suitable for the observed property.

`observedProperty`: Object of class `"SwePhenomenonProperty"`: Identifies or describes the phenomenon for which the observation result provides an estimate of its value. It must be a property associated with the type of the feature of interest.

`featureOfInterest`: Object of class `"GmlFeature"`: A feature of any type (ISO 19109, ISO 19101), which is a representation of the observation target, being the real-world object regarding which the observation is made.

`result`: Object of class `"ANY"`: Contains the value generated by the procedure. The type of the observation result must be consistent with the observed property, and the scale or scope for the value must be consistent with the quantity or category type.

`metadata`: Object of class `"ANY"`: Observation metadata.

`resultTime`: Object of class `"GmlTimeObjectOrNULL"`: The time when the procedure associated with the observation act was applied. For some observations these are identical, in which case the resultTime may be omitted. However, there are important cases where they differ.

`resultQuality`: Object of class `"ANY"`: Event specific quality of a result.

`parameter`: Object of class `"ANY"`: A general event-specific parameter. This will typically be used to record environmental parameters, or event-specific sampling parameters that are not tightly bound to either the feature-of-interest or the procedure.

Extends

Class `"OmOM_ObservationOrNULL"`, directly.

Methods

sosCoordinates `signature(obj = "OmOM_Observation")`: ...

sosFeatureIds `signature(obj = "OmOM_Observation")`: ...

sosFeaturesOfInterest `signature(obj = "OmOM_Observation")`: ...

Examples

```
showClass("OmOM_Observation")
```

`OWS` *Classes and Construction Functions for Elements of the OWS Namespace*

Description

These classes represent elements from the OGC Web Services Common Specification and the OGC Web Services Common Standard.

Usage

```
OwsCapabilities(version, updateSequence = NA, owsVersion = sosDefaultGetCapOwsVersion,
                identification = NULL, provider = NULL, operations = NULL,
                contents = NULL, languages = NULL)
OwsException(exceptionCode, exceptionText = c(), locator = as.character(NA))
OwsExceptionReport(version, lang = as.character(NA), exceptions = list(NA))
OwsGetCapabilities(service, acceptVersions, sections = sosDefaultGetCapSections,
                   acceptFormats = sosDefaultGetCapAcceptFormats,
                   updateSequence = c(as.character(NA)),
                owsVersion = sosDefaultGetCapOwsVersion, acceptLanguages = c(NA))
OwsOperation(name, DCPs, parameters = list(NA), constraints = list(NA),
             metadata = list(NA))
OwsOperationsMetadata(operations, parameters = list(NA), constraints = list(NA),
                      extendedCapabilities = xml2::xml_missing())
OwsRange(minimumValue = as.character(NA), maximumValue = as.character(NA),
         rangeClosure = as.character(NA), spacing = as.character(NA))
OwsServiceProvider(providerName, providerSite = as.character(NA),
                   serviceContact = xml2::xml_missing())
OwsServiceIdentification(serviceType, serviceTypeVersion, profile = c(NA), title,
                    abstract = c(NA), keywords = c(NA), fees = as.character(NA),
                         accessConstraints = c(NA))
```

Arguments

abstract	Brief narrative description of this server, normally available for display to a human.
acceptFormats	Unordered character vector of zero or more response formats desired by client, with preferred formats listed first.
acceptLanguages	Unordered character vector of zero or more languages desired by client, with preferred formats listed first. Only OWS 2.0.0!
acceptVersions	Comma-separated prioritized sequence of one or more specification versions accepted by client, with preferred versions listed first.
accessConstraints	Access constraints that should be observed to assure the protection of privacy or intellectual property, and any other restrictions on retrieving or using data from or otherwise using a server.
constraints	Constraint on valid domain of a non-parameter quantity that applies to an operation which a server implements.
contents	The provider section of a capabilities document, object of class `OwsContentsOrNULL`.
DCPs	Information for a Distributed Computing Platform (DCP) supported for an operation.
exceptionCode	The code attribute of an OWS Exception, see `OwsExceptionsData`.
exceptions	The list of `OwsException` in a `OwsExceptionReport`.
exceptionText	The text element of an OWS Exception, see `OwsExceptionsData`.

`extendedCapabilities`
: The possible contents of the ExtendedCapabilities subsection are not specified by the SOS standard.

`fees`
: Fees and terms for using a server, including the monetary units as specified in ISO 4217.

`identification`
: The identification section of a capabilities document, object of class `OwsServiceIdentificationOrNULL`

`keywords`
: Unordered list of one or more commonly used or formalised word(s) or phrase(s) used to describe a server.

`lang`
: The code attribute of an OWS Exception.

`languages`
: The languages section of a capabilities document, currently an object of class `XMLAbstractNode`.

`locator`
: The locator attribute of an OWS Exception, see `OwsExceptionsData`.

`maximumValue`
: Maximum value of a range (numeric parameter).

`metadata`
: Metadata about an operation and its implementation.

`minimumValue`
: Minimum value of a range (numeric parameter).

`name`
: Name of an operation (request) (for example, GetCapabilities).

`operations`
: A list of objects of class `OwsOperation` in a `OperationsMetadata` object. The provider section of a capabilities document.

`owsVersion`
: The used OWS specification version.

`parameters`
: Parameter valid domain that applies to an operation which a server implements.

`profile`
: Identifier of OGC Web Service (OWS) Application Profile.

`providerName`
: Unique identifier for service provider organization.

`providerSite`
: Reference to the most relevant web site of a service provider.

`provider`
: The provider section of a capabilities document, object of class `OwsServiceProviderOrNULL`.

`rangeClosure`
: Specifies which of minimum and maximum values are included in this range; include when not default of "closed" range. Possible values are closed, open, open-closed, or closed-open.

`sections`
: Unordered character vector of zero or more names of sections of service metadata document to be returned in service metadata document.

`serviceContact`
: Information for contacting service provider.

`service`
: Service type identifier text.

`serviceType`
: A service type name from registry of services.

`serviceTypeVersion`
: Version of a service type implemented by a server.

`spacing`
: Regular distance or spacing between allowed values in this range; include when range is not continuous.

`title`
: Title of a server, normally used for display to a human.

`updateSequence`
: Service metadata document version, value is "increased" whenever any change is made in complete service metadata document. This can be used to request a certain version of a metadata document. Parameter is found in both request and reponse, but may not be supported by a service.

`version`
: The version of the document.

Details

OwsServiceOperation is the top class which is eventually put into the request method, sosRequest(...).

Classes ending in ...OrNULL are used to model optional slots.

Objects from the Class

Objects can be created by calling the construction functions, e.g. in the form OwsCapabilities_1.1.0(...), OwsContents(...) or OwsException(...).

The following classes are virtual and no objects may be created from it: OwsContentsOrNULL, OwsServiceIdentificationOrNULL, OwsServiceProviderOrNULL, OwsOperationsMetadataOrNULL.

Slots

Capabilities:

contents: Object of class "OwsContentsOrNULL", the contents section of a capabilities document.

identification: Object of class "OwsServiceIdentificationOrNULL", the identification section of a capabilities document.

languages: Object of class "XMLAbstractNode", the languages section of a capabilities document, only OWS 2.0.0!

operations: Object of class "OwsOperationsMetadataOrNULL", the operations section of capabilities document.

owsVersion: Object of class "character", the used version of OWS.

provider: Object of class "OwsServiceProviderOrNULL", the provider section of a capabilities document.

updateSequence: Object of class "character", the updateSequence attribute of a capabilities document.

version: Object of class "character", the version of the described service in a capabilities document.

exceptionCode: Object of class "character", the code attribute of an OWS Exception, see OwsExceptionsData.

exceptionText: Object of class "vector", the text of an OWS Exception, see OwsExceptionsData.

locator: Object of class "character", the locator attribute of an OWS Exception, see OwsExceptionsData.

version: Object of class "character", the version of an OWS Exception, see OwsExceptionsData.

lang: Object of class "character", the code attribute of an OWS Exception, see OwsExceptionsData.

exceptions: Object of class "list", the list of OwsException in a OwsExceptionReport.

sections: Object of class "vector", unordered character vector of zero or more names of sections of service metadata document to be returned in service metadata document.

acceptFormats: Object of class "vector", unordered character vector of zero or more response formats desired by client, with preferred formats listed first.

updateSequence: Object of class "vector", service metadata document version.

acceptVersions: Object of class "character", comma-separated prioritized sequence of one or more specification versions accepted by client, with preferred versions listed first.

service: Object of class "character", the name of the service.

request: Object of class "character", the name of the operation/request.

acceptLanguages: Object of class "vector", an unordered character vector of zero or more languages desired by client, with preferred formats listed first. Only OWS 2.0.0!

name: Object of class "character", name of an operation (request) (for example, GetCapabilities).

DCPs: Object of class "list", information for a Distributed Computing Platform (DCP) supported for an operation.

parameters: Object of class "list", parameter valid domain that applies to an operation which a server implements.

constraints: Object of class "list", Constraint on valid domain of a non-parameter quantity that applies to an operation which a server implements.

metadata: Object of class "list", metadata about an operation and its implementation.

operations: Object of class "list", a list of objects of class OwsOperation in a OperationsMetadata object.

extendedCapabilities: Object of class "XMLAbstractNode", the possible contents of the ExtendedCapabilities subsection.

minimumValue: Object of class "character", minimum value of a range (numeric parameter).

maximumValue: Object of class "character", maximum value of a range (numeric parameter).

rangeClosure: Object of class "character", specifies which of minimum and maximum values are included in this range.

spacing: Object of class "character", regular distance or spacing between allowed values in this range; included when range is not continuous.

serviceType: Object of class "character", the service type name from registry of services.

serviceTypeVersion: Object of class "vector", version of a service type implemented by the server.

profile: Object of class "vector", identifier of OGC Web Service (OWS) Application Profile.

title: Object of class "vector", title of the server, normally used for display to a human.

abstract: Object of class "vector", brief narrative description of this server, normally available for display to a human.

keywords: Object of class "vector", unordered list of one or more commonly used or formalised word(s) or phrase(s) used to describe a server.

fees: Object of class "character", fees and terms for using a server, including the monetary units as specified in ISO 4217.

accessConstraints: Object of class "vector", access constraints that should be observed to assure the protection of privacy or intellectual property, and any other restrictions on retrieving or using data from or otherwise using a server.

providerName: Object of class "character", unique identifier for service provider organization.

providerSite: Object of class "character", reference to the most relevant web site of a service provider.

serviceContact: Object of class "XMLAbstractNode", information for contacting service provider.

Extends

OwsCapabilities_1.1.0: Class "OwsCapabilities", directly.

OwsCapabilities_2.0.0: Class "OwsCapabilities_1.1.0", directly. Class "OwsCapabilities", by class "OwsCapabilities_1.1.0", distance 2.

OwsGetCapabilities_1.1.0 Class "OwsGetCapabilities", directly. Class "OwsServiceOperation", by class "OwsGetCapabilities", distance 2.

OwsGetCapabilities_2.0.0: Class "OwsGetCapabilities_1.1.0", directly. Class "OwsGetCapabilities", by class "OwsGetCapabilities_1.1.0", distance 2. Class "OwsServiceOperation", by class "OwsGetCapabilities_1.1.0", distance 3.

OwsServiceIdentification: Class "OwsServiceIdentificationOrNULL", directly.

OwsServiceProvider: Class "OwsServiceProviderOrNULL", directly.

Methods

show `signature(object = "<NAME OF CLASS>")`: Shows a human readable version of the object.

checkRequest `signature(service = "SOS",operation = "OwsGetCapabilities_1.1.0",verbose = "logical")` or `signature(service = "SOS",operation = "OwsGetCapabilities_2.0.0",verbose = "logical")`: Check the given operation for validity and for compliance with the metadata of the given SOS.

encodeRequestKVP `signature(obj = "OwsGetCapabilities")`: See `link{encodeRequestKVP}`.

encodeRequestSOAP `signature(obj = "OwsGetCapabilities")`: See `link{encodeRequestSOAP}`.

encodeRequestXML `signature(obj = "OwsGetCapabilities")`: See `link{encodeRequestXML}`.

encodeRequestKVP `signature(obj = "OwsGetCapabilities_2.0.0")`: See `link{encodeRequestKVP}`.

encodeRequestKVP `signature(obj = "OwsGetCapabilities_1.1.0")`: See `link{encodeRequestKVP}`.

sosRequest `signature(sos = "SOS",request = "OwsServiceOperation",verbose = "logical",inspect = "logical")`: Send the given operation as a request to the given SOS.

Author(s)

Daniel Nuest <daniel.nuest@uni-muenster.de>

References

Whiteside A. (Ed.), OGC Web Services Common Specification, Open Geospatial Consortium Inc., OGC 06-121r3, Version: 1.1.0 with Corrigendum 1

Whiteside A., Greenwood, J. (Eds.), OGC Web Services Common Standard, Open Geospatial Consortium Inc., OGC 06-121r9, Version: 2.0.0

Examples

```
showClass("OwsCapabilities_1.1.0")
showClass("OwsCapabilities_2.0.0")
showClass("OwsCapabilities")
showClass("OwsContents")
showClass("OwsContentsOrNULL")
```

```
showClass("OwsException")
showClass("OwsExceptionReport")
showClass("OwsGetCapabilities_1.1.0")
showClass("OwsGetCapabilities_2.0.0")
showClass("OwsGetCapabilities")
showClass("OwsOperation")
showClass("OwsOperationsMetadata")
showClass("OwsRange")
showClass("OwsServiceIdentification")
showClass("OwsServiceIdentificationOrNULL")
showClass("OwsServiceOperation")
showClass("OwsServiceProvider")
showClass("OwsServiceProviderOrNULL")
```

---

parse *Parsing Functions for XML Documents and Elements*

Description

The functions decode a given XML object to an R representation, which can be an object of a specific class, a list, a named character vector, ...

Usage

```
parseFile(sos, file, verbose = FALSE, ...)
parseCSV(obj, verbose = FALSE)
parseNoParsing(obj)
parseCategoryObservation(obj, sos, verbose = FALSE)
parseComplexObservation(obj, sos, verbose = FALSE)
parseComponent(obj, verbose = FALSE)
parseCompositePhenomenon(obj, verbose = FALSE)
parseCountObservation(obj, sos, verbose = FALSE)
parseDataArray(obj, sos, verbose = FALSE)
parseElementType(obj, sos, verbose = FALSE)
parseEncoding(obj, sos, verbose = FALSE)
parseFeatureCollection(obj, sos)
parseField(obj, sos, verbose = FALSE)
parseFOI(obj, sos, verbose = FALSE)
parseGeometryObservation(obj, sos, verbose = FALSE)
parseMeasure(obj)
parseMeasurement(obj, sos, verbose = FALSE)
parseObservation(obj, sos, verbose = FALSE)
parseObservationCollection(obj, sos, verbose)
parseOM(obj, sos, verbose = FALSE)
parseOwsException(obj)
parseOwsExceptionReport(obj, verbose = FALSE)
parseOwsOperation(obj, namespaces = SosAllNamespaces())
parseOwsRange(obj)
```

```
parseOwsServiceIdentification(obj, namespaces = SosAllNamespaces())
parseOwsServiceProvider(obj)
parsePhenomenonProperty(obj, verbose = FALSE)
parsePoint(obj, sos)
parsePosition(obj, sos)
parseResult(obj, sos, verbose = FALSE)
parseSamplingPoint(obj, sos)
parseSensorML(obj, sos, verbose = FALSE)
parseSosCapabilities(obj, sos)
parseSosFilter_Capabilities(obj, sos)
parseSosObservationOffering(obj, sos)
parseTemporalObservation(obj, sos, verbose = FALSE)
parseTextBlock(obj)
parseTimeGeometricPrimitiveFromParent(obj, format)
parseTimeInstant(obj, format)
parseTimeInstantProperty(obj, format)
parseTimePeriod(obj, format)
parseTimePosition(obj, format)
parseTimeObject(obj, format, timeObjectMap = list(), verbose = FALSE)
parseTruthObservation(obj, sos, verbose = FALSE)
parseValues(values, fields, encoding, sos, verbose = FALSE)
```

Arguments

obj	The object to decode, normally objects of either xml_document.
sos	An object of class SOS-class, which may be utilized/required by some parsing functions to access other parsing functions or encoding information.
verbose	A boolean value indication whether information is printed out to the console during the process - potentially a lot of output!
namespaces	A vector of namespace prefixes and definitions to use.
format	A character string defining the time format to be used in `strptime`.
values	The values to be parsed in `parseValues(...)`.
fields	Field information in `parseValues(...)`, a named list.
file	Name of the file to be parsed in `sosParse(...)`.
encoding	Encoding information in `parseValues(...)`, an object of class `SweTextBlock`.
timeObjectMap	A named list of time objects passed on during parsing (name is the id) to resolve in-document references.
...	Additional arguments that are parsed to `xmlParse(...)` in `sosParse(...)`.

Details

The naming of the functions follow the following rule: parse[optional: namespace prefix][name of the XML element to be parsed]

Not all parsing function that have a `SOS` object or `verbose` in their signature, but few actually use it at this points of development. Some of the parsing functions are **exchangeable** when creating a new SOS connection. Please see the examples!

parseOM is a special function in the respect that it matches sub parsing function depending on an objects xmlName from the list of the given SOS's parsing functions.

parseNoParsing is a convenience function that directly returns the object without any changes.

sosParse allows parsing of files for all elements that have a parsers registered with the given SOS.

Value

An objects of a specific class depending on the parsing method and the passed object, possibly even lists or named character vectors.

Warning

Functions might result in error if parsed an object of the wrong type, because that is normally not checked.

Some of the functions are placeholders for future implementations!

Author(s)

Daniel Nuest <daniel.nuest@uni-muenster.de>

See Also

SosParsingFunctions, sosParsers-methods

Examples

```
# parsing a XML string to an exception report object
er.doc <- xml2::read_xml(paste0("<ows:ExceptionReport xmlns:ows=\"http://www.opengis.net/ows/1.1\"",
  " xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\" version=\"1.0.0\"",
 " xsi:schemaLocation=\"http://schemas.opengis.net/ows/1.1.0/owsExceptionReport.xsd\">",
 "<ows:Exception exceptionCode=\"VersionNegotiationFailed\" locator=\"AcceptVersions\">",
  "<ows:ExceptionText>The parameter 'AcceptVersions' does not contain the version",
  " of this SOS: '1.0.0'</ows:ExceptionText></ows:Exception></ows:ExceptionReport>"))
er.parsed <- parseOwsExceptionReport(er.doc)
print(er.parsed)
str(er.parsed)

## Not run:
# save and re-parse an observation from file
obsId <- getObservationById(sos = mySOS, observationId = "o_3508493",
saveOriginal = TRUE)
.files <- list.files(getwd())
.startWithO_ <- .files %in% grep("o_", .files, value=TRUE)
.observationFiles <- subset(.files, .startWithO_)

obsId <- parseFile(sos = mySOS, file = .observationFiles[[1]])

## End(Not run)
```

SA *Classes of the Namespace sa*

Description

Classes and construction functions for elements from the OGC specification "Observations and Measurements - Part 2 - Sampling Features".

Usage

```
SaSamplingPoint(sampledFeatures, position, relatedObservation = list(NA),
          relatedSamplingFeature = list(NA), surveyDetails = NA, id = NA_character_)
```

Arguments

```
sampledFeatures
                ~~
position        ~~
relatedObservation
                ~~
relatedSamplingFeature
                ~~
surveyDetails   ~~
id              ~~
```

Value

Construction functions: An object of the respective class.

Objects from the Class

Objects can be created by calls to the construction functions of the form `SaSamplingPoint(...)`.

Slots

`sampledFeatures`**:** Object of class `"list"` which contains the sampled features.

`position`**:** Object of class `"GmlPointProperty"` which contains the position of a feature.

`relatedObservation`**:** Object of class `"list"` which contains identifiers of related observations.

`relatedSamplingFeature`**:** Object of class `"list"` which contains identifiers of related sampling features.

`surveyDetails`**:** Object of class `"ANY"` which can contain information about survey details (which are currently no modeled in an R class).

`id`**:** The identifier of a sampling elements (object of class `"character"`).

`shape`**:** Object of class `"ANY"` which could be used to model the shape of a sampling surface.

`area`**:** Object of class `"ANY"` which could be used to model the area of a sampling surface.

Extends

SaSamplingPoint and SaSamplingSurface: Class "GmlFeature", directly. Class "GmlFeatureOrNULL", by class "GmlFeature", distance 2.

Methods

show `signature(object = "SaSamplingPoint")`: ...

show `signature(object = "SaSamplingSurface")`: ...

Author(s)

Daniel Nuest <daniel.nuest@uni-muenster.de>

References

Simon Cox (Ed.), Observations and Measurements - Part 2 - Sampling Features, OGC 07-002r3

Examples

```
showClass("SaSamplingPoint")

# create sampling point
SaSamplingPoint(sampledFeatures = list("feature1", "feature2"),
                position = GmlPointProperty(href = "http://link.to/point"))
```

SamsSamplingFeature-class

Classes and creation functions for Sampling Features

Description

Sampling Feature classes.

SamsShape represents the geometry of a spatial sampling feature, that can be used as a feature of interest. Currently, only points are supported.

Objects from the Class

Objects can be created by calls to the creation functions:

`SamsSamplingFeature(...)`

`SamsShape(...)`

Slots

`identifier`: Object of class `"character"` ~~

`name`: Object of class `"character"` ~~

`type`: Object of class `"character"` ~~

`sampledFeature`: Object of class `"character"` ~~

`shape`: Object of class `"SamsShape"` ~~

`id`: Object of class `"character"` ~~

`point`: Object of class `"GmlPoint"` ~~

`id`: Object of class `"character"` ~~

`srsName`: Object of class `"character"` ~~

`srsDimension`: Object of class `"integer"` ~~

`axisLabels`: Object of class `"character"` ~~

`uomLabels`: Object of class `"character"` ~~

Extends

`SamsShape` extends `"GmlGeometry"`, directly.

`SamsSamplingFeature` extends `"GmlFeature"`, directly. `SamsSamplingFeature` extends `"GmlFeatureOrNULL"`, by class "GmlFeature", distance 2. `SamsSamplingFeature` extends `"GmlFeatureOrGmlFeaturePropertyOrNULL"`, by class "GmlFeature", distance 2.

Methods

print `signature(x = "SamsSamplingFeature")`: ...

show `signature(object = "SamsSamplingFeature")`: ...

sosCoordinates `signature(obj = "SamsSamplingFeature")`: ...

toString `signature(x = "SamsSamplingFeature")`: ...

print `signature(x = "SamsShape")`: ...

show `signature(object = "SamsShape")`: ...

toString `signature(x = "SamsShape")`: ...

Note

Schema: http://schemas.opengis.net/sampling/2.0/samplingFeature.xsd

Author(s)

Daniel Nuest

References

https://www.opengeospatial.org/standards/om

Examples

```
showClass("SamsSamplingFeature")
showClass("SamsShape")
```

SML *Classes of the Namespace sml*

Description

Classes, construction functions, and accessor functions for elements from the OGC specification "OpenGIS(R) Sensor Model Language (SensorML) Implementation Specification".

The only class at the moment is `"SensorML"` which wraps an `"XMLInternalDocument"` and some additional information. This strongly depends on the SensorML Profile for Discovery to find the respective paramters.

Objects from the Class

Objects can be created by calls to the construction method in the form `SensorML(...)`.

Slots

`xml`**:** Object of class `"XMLInternalDocument"`, holds the XML representation of the sensor description.

`coords`**:** Object of class `"data.frame"`, holds the position of the sensor.

`id`**:** Object of class `"character"`, the main identifier of the sensor.

`name`**:** Object of class `"character"`, a naming identifier of the sensor.

`description`**:** Object of class `"character"`, a normal text description of the sensor.

`boundedBy`**:** Object of class `"matrix"`, the bounding box of the sensor.

Methods

show `signature(object = "SensorML")`: Prints a short statement to the command line.

plot `signature(object = "SensorML")`: Plots the sensor using coercion to an object of class `"Spatial"`.

Author(s)

Daniel Nuest <daniel.nuest@uni-muenster.de>

References

Botts, M., Robin, A. (Eds.), OpenGIS(R) Sensor Model Language (SensorML) Implementation Specification, Open Geospatial Consortium: 07-000.

Houbie, F., Skivee F., Robin A., Jirka S., Broering, A., Nuest D. (2009): OGC(R) Catalogue Services Specification 2.0 - Extension Package for ebRIM Application Profile: SensorML. OGC Discussion Paper. Open Geospatial Consortium: 09-163. http://portal.opengeospatial.org/files/?artifact_id=37944.

See Also

DescribeSensor

Examples

```
showClass("SensorML")

mySOS <- SOS(url = "http://sensorweb.demo.52north.org/sensorwebtestbed/service/kvp",
             binding = "KVP")
mySensor <- describeSensor(sos = mySOS,
procedure = sosProcedures(mySOS)[[1]],
outputFormat = 'text/xml; subtype="sensorML/1.0.1"', # space is needed!
)
class(mySensor)
print(mySensor)

sosId(mySensor)
sosName(mySensor)
sosBoundedBy(mySensor)
sosCoordinates(mySensor)
sosGetCRS(mySensor)

## Not run:
plot(mySensor)

## End(Not run)
```

SOS *Class, and Construction and Accessor Functions for "SOS"*

Description

Base class of a connection to a Sensor Observation Service.

Usage

```
SOS(url, binding = SosDefaultBinding(), version = sosDefaultServiceVersion,
parsers = SosParsingFunctions(), encoders = SosEncodingFunctions(),
dataFieldConverters = SosDataFieldConvertingFunctions(),
timeFormat = sosDefaultTimeFormat, verboseOutput = FALSE,
switchCoordinates = FALSE, useDCPs = TRUE, dcpFilter = SosDefaultDCPs(),
additionalKVPs = list(), ...)

SOS_Test(name = "test", binding = SosDefaultBinding(), version = sos100_version,
         parsers = SosParsingFunctions(), encoders = SosEncodingFunctions(),
         dataFieldConverters = SosDataFieldConvertingFunctions(),
         timeFormat = sosDefaultTimeFormat, verboseOutput = FALSE,
        switchCoordinates = FALSE, useDCPs = TRUE, dcpFilter = SosDefaultDCPs(),
         additionalKVPs = list(), ...)
```

Arguments

url	See the corresponding slot description.
binding	See the corresponding slot description.
version	See the corresponding slot description.
parsers	See the corresponding slot description.
encoders	See the corresponding slot description.
dataFieldConverters	See the corresponding slot description.
timeFormat	See the corresponding slot description.
verboseOutput	See the corresponding slot description.
switchCoordinates	See the corresponding slot description.
useDCPs	See the corresponding slot description.
dcpFilter	See the corresponding slot description.
additionalKVPs	See the corresponding slot description.
...	Additional parameters that are passed on to the getObservation call that is done within this function.
name	Name of the test SOS class.

Details

From the introduction of the specification document: "The goal of SOS is to provide access to observations from sensors and sensor systems in a standard way that is consistent for all sensor systems including remote, in-situ, fixed and mobile sensors."

Value

The construction functions returns an object of class SOS-class.

Objects from the Class

Objects can be created by calls to the construction function of the form SOS(...).

Object from the class can be used in calls to function for metadata retrieval of sensors (link{describeSensor-methods}) and observation data queries (link{getObservation-methods} and link{getObservationById-methods})

Slots

url: Object of class "character": The endpoint of the service, e.g. http://myUrl.org/SOS1/sos.

binding: Object of class "character": The binding, or transport protocol, see SosSupportedBindings for available ones.

version: Object of class "character": The connected service's version, e.g. "1.0.0".

capabilities: Object of class "OwsCapabilities" ~~

parsers: Object of class "list": A list of named functions for parsing of SOS responses.

`encoders`: Object of class `"list"`: A list of named functions for encoding of SOS requests.

`dataFieldConverters`: Object of class `"list"`: A list of named functions to be used by the parsing methods to convert data values to the correct R type, see `SosDataFieldConvertingFunctions` for the default functions and how to add your own converters.

`timeFormat`: Object of class `"character"`: The time format to be used or decoding and encoding time character strings to and from `POSIXt` classes.

`verboseOutput`: Object of class `"logical"`: Trigger parameter for extensive debugging information on the console.

`switchCoordinates`: Object of class `"logical"`: Trigger enabling switching of lat/long during parsing stage.

`useDCPs`: Object of class `"logical"`: Trigger for not using DCP endpoints from the capabilities but just the URL defined when creating the SOS connection. If there is a specific endpoint for a specific binding the user must make sure the configured binding and DCP match.

`dcpFilter`: Object of class `"list"`: Named list of regular expressions to be applied (using `grep`) to DCPs if there is more than one for the chosen binding.

`addtionalKVPs`: A nam of class `"list"`: Named list of key-value-pairs to be appended to an KVP request.

Methods

sosAbstract `signature(obj = "SOS")`: ...

sosBinding `signature(sos = "SOS")`: The protocol used for a connection to the service.

sosCapabilitiesDocumentOriginal `signature(sos = "SOS")`: To retrieve the full original service metadata document.

sosCaps `signature(sos = "SOS")`: ...

sosContents `signature(sos = "SOS")`: ...

sosDataFieldConverters `signature(sos = "SOS")`: ...

sosTime `signature(obj = "SOS")`: ...

sosTime `signature(obj = "SOS")`: ...

sosTitle `signature(obj = "SOS")`: ...

sosOperations `signature(sos = "...")`: ...

sosGetCRS `signature(obj = "character")`: ...

sosGetDCP `signature(sos = "SOS",operation = "character",type = "character")`: Get the distributed computing platform URL for the given operation and method type. If type is missing, the function returns all available DCPs.

sosEncoders `signature(sos = "SOS")`: ...

sosFeatureIds `signature(obj = "list")`: ...

sosFeaturesOfInterest `signature(obj = "list")`: ...

sosGetCRS `signature(obj = "character")`: Get an object of class `sp:::CRS` for a given OGC URN depicting a reference system, like `urn:ogc:def:crs:EPSG:1000`.

sosName `signature(obj = "...")`: ...

sosObservedProperties `signature(sos = "SOS")`: ...

sosOffering `signature(sos = "SOS",offeringId = "character")`: ...

sosOfferingIds `signature(sos = "SOS")`: ...

sosOfferings `signature(sos = "SOS")`: ...

sosOperation `signature(sos = "SOS",operationName = "character")`: ...

sosOperationsMetadata `signature(sos = "SOS")`: ...

sosParsers `signature(sos = "SOS")`: ...

sosProcedures `signature(sos = "SOS")`: Accessor function for the procedures of a SOS (via list in capabilities of GetObservation operation) or a SosObservationOffering.

sosResponseFormats `signature(sos = "SOS")`: TBD: add missing signatures ...

sosResponseMode `signature(sos = "SOS")`: ...

sosResultModels `signature(sos = "SOS")`: ...

sosServiceIdentification `signature(sos = "SOS")`: ...

sosServiceProvider `signature(sos = "SOS")`: ...

sosSrsName `signature(sos = "SOS")`: ...

sosSwitchCoordinates `signature(sos = "SOS")`: ...

sosTimeFormat `signature(sos = "SOS")`: ...

sosUrl `signature(sos = "SOS")`: ...

sosVersion `signature(sos = "SOS")`: ...

sosTime `signature(obj = "SOS")`: Accessor function for the event time period from the GetObservation operations metadata.

sosFilter_Capabilities `signature(sos = "SOS")`: Accessor function for the filter capabilities of a SOS object.

checkRequest `signature(service = "SOS_2.0.0",operation = "SosGetDataAvailability_1.0.0",verbose = "logical")`: ...

checkRequest `signature(service = "SOS_2.0.0",operation = "SosGetFeatureOfInterest_2.0.0",verbose = "logical")`: ...

checkRequest `signature(service = "SOS_2.0.0",operation = "SosGetObservation",verbose = "logical")`: ...

describeSensor `signature(sos = "SOS_2.0.0",procedure = "character")`: ...

getCapabilities `signature(sos = "SOS_2.0.0")`: ...

getDataAvailability `signature(sos = "SOS_2.0.0")`: ...

getFeatureOfInterest `signature(sos = "SOS_2.0.0",featureOfInterest = "ANY")`: ...

getFeatureOfInterest `signature(sos = "SOS_2.0.0",featureOfInterest = "character")`: ...

getObservationById `signature(sos = "SOS_2.0.0",observationId = "character")`: ...

getObservation `signature(sos = "SOS_2.0.0",offering = "character")`: ...

getObservation `signature(sos = "SOS_2.0.0",offering = "SosObservationOffering_2.0.0")`: ...

print `signature(x = "SOS_2.0.0")`: ...

show `signature(object = "SOS_2.0.0")`: ...

sosBinding `signature(sos = "SOS_2.0.0")`: ...

sosRequest `signature(sos = "SOS_2.0.0",request = "OwsServiceOperation",verbose = "logical",inspect = "logical")`: ...

sosUrl `signature(sos = "SOS_2.0.0")`: ...

toString `signature(x = "SOS_2.0.0")`: ...

Author(s)

Daniel Nuest <daniel.nuest@uni-muenster.de>

References

Na, A., Priest, M. (Eds.), Sensor Observation Service, Open Geospatial Consortium Inc., OGC 06-009r6, Version: 1.0

The document is available for download at https://www.opengeospatial.org/standards/sos.

See Also

See also creation function SOS and the package vignette for general description of use.

Examples

```
showClass("SOS")
showClass("SOS_2.0.0")

# create a SOS connection
mySOS <- SOS(url = "http://sensorweb.demo.52north.org/sensorwebtestbed/service/kvp",
             binding = "KVP")

# create the URL to a GET request for GetCapabilities
sosCapabilitiesUrl(mySOS)

# access details of the SOS connection and it's metadata
sosContents(mySOS)
sosTime(mySOS)
sosFeaturesOfInterest(mySOS)
sosBinding(mySOS)

## Not run:
# create a SOS connetion with a specific connection method and time format
mysos <- SOS(url = "http://mysos.org/sos",
    binding = "KVP", timeFormat = "

# turn on verbose output for all methods and functions
SOS(url = "http://mysos.org/sos", verboseOutput = TRUE)

# get the meaning of an exception code
sosExceptionCodeMeaning(ex@exceptionCode)
```

```
# create a CRS object from a URN CRS string
sosGetCRS("urn:ogc:def:crs:EPSG:4217")

# create a SOS for a specific binding at a specific endpoint
SOS(url = "http://localhost:8080/52n-sos/sos/pox", binding = "POX",
useDCPs = FALSE)

# create a SOS using only the DCPs from the capabilities that match a specific
# pattern with the default binding
SOS(url = "http://localhost:8080/52n-sos/sos/service",
dcpFilter = list("POX" = "/pox"))

## End(Not run)
```

SosBindings *Bindings and Connecition Methods of OGC Sensor Observation Service*

Description

The SOS comes with three possible methods of transfering data, HTTP GET, HTTP POST and SOAP.

Details

The **POST** binding is described in the official SOS specification and should be the default method.

The **GET** binding is described by OOTethys in a Best Practice document: https://web.archive.org/web/20120616065001/http://www.oostethys.org/best-practices/best-practices-get. It contains some special encoding for bounding boxes, as the only spatial filter, and time periods, as the only temporal filter.

The **SOAP** binding is not official with regards to the spec, and also not implemented yet.

The connection method can be changed on creation of a SOS object.

References

Wikipedia page for HTTP request methods: https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Request_methods.

See Also

SosSupportedBindings

Examples

```
# HTTP connection methods supported by this sos4R implementation
supported <- SosSupportedBindings()
supported

## Not run:
sos <- SOS("http://sosurl.org/", binding = "KVP")

## End(Not run)
```

SosCapabilities *Class and Construction Function for "SosCapabilities"*

Description

The service metadata document of a Sensor Observation Service.

Usage

```
SosCapabilities(version, updateSequence = NA, owsVersion = "1.1.0",
                identification = NULL, provider = NULL, operations = NULL,
                filterCapabilities = NULL, contents = NULL)
```

Arguments

version	The version of the service.
updateSequence	Service metadata document version, value is "increased" whenever any change is made in complete service metadata document. This can be used to request a certain version of a metadata document. Parameter is found in both request and reponse, but may not be supported by a service.
owsVersion	The used OWS specification version.
identification	The identification section of a capabilities document, object of class OwsServiceIdentification.
provider	The provider section of a capabilities document, object of class OwsServiceProvider.
operations	A list of objects of class OwsOperation in a OperationsMetadata object. The provider section of a capabilities document.
filterCapabilities	An object of class SosFilter_Capabilities contaning the filter capabilities of a service.
contents	The provider section of a capabilities document, object of class SosContents.

Details

This document provides clients with service metadata about a specific service instance, including metadata about the tightly-coupled data served.

The portions of the GetCapabilities response document that are defined in the OWS Common specification are not modified for SOS. The sections of the response that are specific for the SOS are the Filter_Capabilities and the Contents section.

Objects from the Class

Objects can be created by calls to the construction function of the form `SosCapabilities(...)` including the parameter `"owsVersion"` for the respective version of the service metadata document.

Slots

`filterCapabilities:` Object of class `"SosFilter_CapabilitiesOrNULL"` ~~

`identification:` Object of class `"OwsServiceIdentificationOrNULL"` ~~

`provider:` Object of class `"OwsServiceProviderOrNULL"` ~~

`operations:` Object of class `"OwsOperationsMetadataOrNULL"` ~~

`contents:` Object of class `"OwsContentsOrNULL"` ~~

`version:` Object of class `"character"` ~~

`updateSequence:` Object of class `"character"` ~~

`owsVersion:` Object of class `"character"` ~~

Extends

Class `"OwsCapabilities_1.1.0"`, directly. Class `"OwsCapabilities"`, by class "OwsCapabilities_1.1.0", distance 2.

Methods

No methods defined with class "SosCapabilities_1.0.0" in the signature.

Author(s)

Daniel Nuest <daniel.nuest@uni-muenster.de>

References

Section 8.2.3 of the SOS specification: Na, A., Priest, M. (Eds.), Sensor Observation Service, Open Geospatial Consortium Inc., OGC 06-009r6, Version: 1.0

See Also

`SosFilter_Capabilities`, `SosContents`, `OwsCapabilities`

Examples

```
showClass("SosCapabilities_1.0.0")
showClass("SosCapabilities_2.0.0")
```

SosContents-class *Class and Construction Function of "SosContents"*

Description

`SosContents` models the sos:Contents section in a service metadata document.

Usage

```
SosContents(observationOfferings)
```

Arguments

`observationOfferings`
: A list of objects of class `SosObservationOffering`.

Details

The SosContents section extends the generic ows:Contents elements. It contains the `SosObservationOffering`s of a Sensor Observation Service.

Objects from the Class

Objects can be created by calls to the construction funtion in the form `SosContents(...)`.

The virtual class `SosContentsOrNULL` is used to model optional slots in classes containing `SosContents`: No objects may be created from it.

Slots

`observationOfferings`**:** Object of class `"list"` ~~

`xml`**:** Object of class `"XMLAbstractNode"` ~~

Extends

Class `"OwsContents"`, directly. Class `"SosContentsOrNULL"`, directly. Class `"OwsContentsOrNULL"`, by class "OwsContents", distance 2.

Methods

show `signature(object = "SosContents")`: ...

Author(s)

Daniel Nuest <daniel.nuest@uni-muenster.de>

References

See section 8.2.3.2, "Contents Section", of the SOS specification.

See Also

SosObservationOffering, OwsContents

Examples

```
showClass("SosContents")
showClass("SosContentsOrNULL")
```

---

sosConvertString *SOS Conversion functions for Observation Results*

Description

These functions are called by the parsers of om:Observation and om:Measurement documents to convert the actual values to the correct classes.

Usage

```
sosConvertString(x, sos)
sosConvertDouble(x, sos)
sosConvertTime(x, sos)
sosConvertLogical(x, sos)
```

Arguments

x	The object to be converted.
sos	An object of class SOS, whose settings, like formating information, can be utilitzed.

Details

The methods are automatically called from the given SOS's list of conversion functions. This is either default or can be set manually on creation.

If you want to provide you own conversion functions, follow the example below. Always include the common parameters x and sos.

There are functions to access the converters of a SOS (sosDataFieldConverters-methods) and to combine default and your own converters (SosDataFieldConvertingFunctions).

Value

An object of the respective class converted from the parameter x.

Author(s)

Daniel Nuest <daniel.nuest@uni-muenster.de>

See Also

sosDataFieldConverters-methods, SosDataFieldConvertingFunctions

Examples

```
## Not run:
sos <- SOS(url = SosExampleServices()[[2]])
one <- sosConvertDouble("1", sos)
class(one)

# add conversion rules, also possible to override default ones
myConverters <- SosDataFieldConvertingFunctions(
"C" = sosConvertDouble,
"S/m" = sosConvertDouble)
sos <- SOS(url = SosExampleServices()[[2]], dataFieldConverters = myConverters)

# show converters
sosDataFieldConverters(sos)

## End(Not run)
```

---

sosCreate *Convenience Functions for Request Parameter Creations*

Description

These methods can be seen as convenience functions or shortcuts to regularly used parameters in GetObservation requests to a Sensor Observation Service. The remove some complexity and target the most common cases, but also limit flexibility.

Usage

```
sosCreateBBOX(lowLat, lowLon, uppLat, uppLon, srsName, srsDimension = NA_integer_,
              axisLabels = NA_character_, uomLabels = NA_character_,
              propertyName = sosDefaultSpatialOpPropertyName)
sosCreateBBoxMatrix(lowLat, lowLon, uppLat, uppLon)
sosCreateFeatureOfInterest(objectIDs = list(NA), spatialOps = NULL, bbox = NULL,
                           srsName = NA_character_)
sosCreateEventTime(time, operator)
sosCreateEventTimeList(time, operator)
sosCreateTimeInstant(sos, time, frame = as.character(NA),
                     calendarEraName = as.character(NA),
```

```
                      indeterminatePosition = as.character(NA))
sosCreateTimePeriod(sos, begin, end, frame = as.character(NA),
                    calendarEraName = as.character(NA),
                    indeterminatePosition = as.character(NA),
                    duration = as.character(NA), timeInterval = NULL)
sosCreateTime(sos, time, operator = sosDefaultTemporalOperator)
```

Arguments

lowLat	Minimum latitude for bounding box and bounding box matrix.
lowLon	Minimum longitude for bounding box and bounding box matrix.
uppLat	Maximum latitude for bounding box and bounding box matrix.
uppLon	Maximum longitude for bounding box and bounding box matrix.
srsName	Name of the spatial reference system for bounding box, e.g. '"urn:ogc:def:crs:EPSG:4326"'.
srsDimension	Dimensions of the spatial reference system, e.g. '2'.
axisLabels	Labels of the axes of a bounding box as an ordered character vector.
uomLabels	Unit of measurement labels as an ordered character vector for the axes in a bounding box, e.g. '"deg"'.
propertyName	The spatial property name for the bounding box, e.g. '"urn:ogc:data:location"'
objectIDs	Identifiers of a feature of interest list.
spatialOps	An object of class OgcSpatialOps-class which is inserted into the feature of interest element.
bbox	Shortcut to add a feature of interest with a GmlEnvelope-class, object must be a matrix as created by sosCreateBBoxMatrix(...).
time	Object of class "GmlTimeGeometricPrimitive" for sosCreateEventTimeList, or an object of class POSIXt for sosCreateTimePeriod,or an object of class character for sosCreateTime.
operator	The operator to be used for the time in sosCreateEventTimeList, e.g. '"TM_During"'.
sos	An object of class SOS-class for which the element is created. The SOS might for example be required for formatting settings.
frame	Provides a URI reference that identifies a description of the reference system.
calendarEraName	The name of the calendar era.
begin	Object of class POSIXt.
end	Object of class POSIXt.
indeterminatePosition	Inexact temporal positions may be expressed using the optional indeterminatePosition parameter. This takes one of the following values: after, before, now, unknown.
duration	Duration of an interval using ISO 8601 syntax for temporal length.
timeInterval	An object of class "GmlTimeIntervalOrNULL" to be used in a GmlTimePeriod-class.

Value

An object of the respective class, or a list in case of `sosEventTimeList`.

Methods

`signature(time = "GmlTimeGeometricPrimitive")` Create sos:time based on the given `GmlTimeGeometricPrimitive`.

`signature(sos = "SOS", time = "POSIXt")` Create sos:time with time instant based on the given time.

`signature(sos = "SOS", begin = "POSIXt", end = "POSIXt")` Create sos:time with time interval based on the given begin and end times.

Author(s)

Daniel Nuest <daniel.nuest@uni-muenster.de>

See Also

These methods create object of the following classes: GmlTimeInstant-class, GmlTimePeriod-class, SosEventTime-class, SosFeatureOfInterest-class, OgcBBOX-class, matrix-class.

Examples

```
# create a feature of interest based on identifiers
foiIDs <- list("urn:ogc:object:feature:1", "urn:ogc:object:feature:2")
foiObj <- sosCreateFeatureOfInterest(objectIDs = foiIDs[1:2])
print(foiObj)

# create a bounding box matrix and use it to create a spatial feature of interest
bboxMatrix <- sosCreateBBoxMatrix(lowLat = 50.0, lowLon = 7.0,
                                  uppLat = 53.0, uppLon = 10.0)
foiBBox <- sosCreateFeatureOfInterest(bbox = bboxMatrix,
                                      srsName = "urn:ogc:def:crs:EPSG:6.8:4326")
print(foiBBox)

# create a foi with a bounding box
bbox <- sosCreateBBOX(lowLat = 50.0, lowLon = 7.0, uppLat = 53.0, uppLon = 10.0,
                      srsName = "urn:ogc:def:crs:EPSG:6.8:4326",
                      srsDimension = as.integer(2), axisLabels = "lat,lon",
                      uomLabels = "deg,deg", propertyName = "bboxName")
foiBBox2 <- sosCreateFeatureOfInterest(spatialOps = bbox)
print(foiBBox2)

last.period <- sosCreateTimePeriod(sos = SOS_Test(),
begin = (Sys.time() - 3600 * 24 * 7), end = Sys.time())

period <- sosCreateTimePeriod(sos = SOS_Test(),
begin = as.POSIXct("2010/01/01"), end = as.POSIXct("2010/01/07"))
eventTime <- sosCreateEventTimeList(period)

sosCreateTime(sos = SOS_Test(), time = "2007-07-07 07:00::2008-08-08 08:00")
sosCreateTime(sos = SOS_Test(), time = "2007-07-07 07:00/2010-10-10 10:00")
```

```
sosCreateTime(sos = SOS_Test(), time = "::2007-08-05")
sosCreateTime(sos = SOS_Test(), time = "2007-08-05/")
```

SosEventTime *Classes and Construction Functions for sos:eventTime elements.*

Description

Temporal query parameters for GetObservation requests.

Usage

```
SosEventTime(temporalOps)
```

Arguments

temporalOps	An object of class OgcBinaryTemporalOp-class to be wrapped by the sos:eventTime element.

Details

Specifies the time period(s) for which observations are requested. This allows a client to request observations from a specific instant, multiple instances or periods of time in the past, present and future. The supported range is listed in the selected offering capabilities. The objects of these classes are used in the GetObservation (paramter in GetObservation).

A typical example in a POST request:
<eventTime>
<ogc:TM_During>
<ogc:PropertyName>om:samplingTime</ogc:PropertyName>
<gml:TimePeriod>
<gml:beginPosition>2006-11-05T17:18:58.000-06:00</gml:beginPosition>
<gml:endPosition>2006-11-05T21:18:59.000-06:00</gml:endPosition>
</gml:TimePeriod>
</ogc:TM_During>
</eventTime>

In GET binding (SosBindings) the eventTime is simply omitted for getting the latest observation.

It is recommended to use the creation functions as shown in the examples.

Objects from the Classes

Objects can be created by calls to the construction functions of the form `SosEventTime(...)`.

Slots

temporalOps: Object of class "OgcBinaryTemporalOp" for SosEventTime, the temporal operand to be inserted into the event time, or an object of class "character" for SosEventTimeLatest.

Methods

encodeKVP `signature(obj = "SosEventTime", sos = "SOS")`: Encode the given object as a key-value pair.

encodeXML `signature(obj = "SosEventTime", sos = "SOS")`: Encode the given object as XML.

Author(s)

Daniel Nuest <daniel.nuest@uni-muenster.de>

References

See SOS specification, Table 4: "Parameters of GetObservation Request".

See Also

See also SosGetObservation-class, sosCreateEventTimeList-methods.

Examples

```
showClass("SosEventTime")

# create SosEventTime for all times after the given time stamp
timePos <- GmlTimePosition(as.POSIXct("2010-01-01 12:00"))
tOps <- TM_After(time = GmlTimeInstant(timePosition = timePos))
time1 <- SosEventTime(tOps)

# encode it as XML
encodeXML(time1, sos = SOS_Test())

# encode it as KVP
encodeKVP(time1, sos = SOS_Test())
```

SosFeatureOfInterest-class

Class and Construction Function for "SosFeatureOfInterest"

Description

Element in a GetObservation request to a Sensor Observation service to constrain the observations to be returned regarding the observed feature.

Usage

```
SosFeatureOfInterest(objectIDs = list(NA), spatialOps = NULL)
```

Arguments

objectIDs	A list of character identifiers of features in a SOS.
spatialOps	An object of class OgcSpatialOps for spatial filtering.

Details

Specifies the feature for which observations are requested. This can either be represented by a reference to a feature ID advertised in the capabilities document or can be a spatial constraint.

Objects from the Class

Objects can be created by calls to the construction function of the form `SosFeatureOfInterest(...)`.

`SosFeatureOfInterestOrNULL` is a virtual class to model optional slots of containing elements: No objects may be created from it.

Slots

`objectIDs`: Object of class `"list"`: Identifiers of features of interest.

`spatialOps`: Object of class `"OgcSpatialOpsOrNULL"`: A spatial filtering of the result.

Extends

Class `"SosFeatureOfInterestOrNULL"`, directly.

Methods

encodeXML `signature(obj = "SosFeatureOfInterest", sos = "ANY")`: Convert the object to a XML representation.

Author(s)

Daniel Nuest <daniel.nuest@uni-muenster.de>

References

See section 8.4.2 of the SOS specification: Na, A., Priest, M. (Eds.), Sensor Observation Service, Open Geospatial Consortium Inc., OGC 06-009r6, Version: 1.0

See Also

See also `SosGetObservation`, and the convenience creation function `sosCreateFeatureOfInterest-methods`.

Examples

```
showClass("SosFeatureOfInterest")
showClass("SosFeatureOfInterestOrNULL")
```

SosFilter_Capabilities-class
Classes and Construction Functions for "SosFilter_Capabilities" Elements

Description

Additional section in the service metadata document of a Sensor Observation Service, which contains information about the supported filters.

Usage

```
SosFilter_Capabilities(spatial = list(NA_character_), temporal = list(NA_character_),
                       scalar = list(NA_character_), id = list(NA_character_))
```

Arguments

spatial	A character list of names of available spatial filters.
temporal	A character list of names of available temporal filters.
scalar	A character list of names of available scalar filters.
id	A character list of names of available filters on identifiers.

Details

The FilterCapabilities section is used to indicate what types of query parameters are supported by the service. These capabilities refer to the parameters of the GetObservation operation which is the only operation that includes OGC filter-like expressions.

Objects from the Class

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

`SosFilter_CapabilitiesOrNULL` is virtual class: No objects may be created from it.

Slots

`spatial`: Object of class `"list"` with character strings for names of spatial filters.

`temporal`: Object of class `"list"` with character strings for names of temporal filters.

`scalar`: Object of class `"list"` with character strings for names of scalar filters.

`id`: Object of class `"list"` with character strings for names of ID filters.

Extends

Class `"SosFilter_CapabilitiesOrNULL"`, directly.

Methods

show `signature(object = "SosFilter_Capabilities")`: ...

Author(s)

Daniel Nuest <daniel.nuest@uni-muenster.de>

References

See section 8.2.3.1, "FilterCapabilities Section", the SOS specification: Na, A., Priest, M. (Eds.), Sensor Observation Service, Open Geospatial Consortium Inc., OGC 06-009r6, Version: 1.0

See Also

SosCapabilities

Examples

```
showClass("SosFilter_Capabilities")
showClass("SosFilter_CapabilitiesOrNULL")
```

SosGetDataAvailability_1.0.0-class

Class and construction function for "GetDataAvailability" *operation*

Description

See SOS 2.0 Hydrology profile specification, OGC 14-004r1, section 7.4, requirement 12

Objects from the Class

Objects can be created by calls to construction functions:

`DataAvailabilityMember(...)`

`SosGetDataAvailability_1.0.0(...)`

Slots

procedure: Object of class "character" ~~

observedProperty: Object of class "character" ~~

featureOfInterest: Object of class "character" ~~

phenomenonTime: Object of class "GmlTimePeriod" ~~

procedures: Object of class "list" ~~

observedProperties: Object of class "list" ~~

featuresOfInterest: Object of class "list" ~~

offerings: Object of class "list" ~~

service: Object of class "character" ~~

request: Object of class "character" ~~

version: Object of class "character" ~~

Methods

print `signature(x = "DataAvailabilityMember")`: ...

show `signature(object = "DataAvailabilityMember")`: ...

toString `signature(x = "DataAvailabilityMember")`: ...

checkRequest `signature(service = "SOS_2.0.0", operation = "SosGetDataAvailability_1.0.0", verbose = "logical")`: ...

encodeRequestKVP `signature(obj = "SosGetDataAvailability_1.0.0")`: ...

sosName `signature(obj = "SosGetDataAvailability_1.0.0")`: ...

toString `signature(x = "SosGetDataAvailability_1.0.0")`: ...

Extends

Class `SosGetDataAvailability_1.0.0` extends `"OwsServiceOperation"`, directly.

Author(s)

Daniel Nuest <daniel.nuest@uni-muenster.de>

References

OGC 14-004r1, section 7.4, requirement 12

Examples

```
showClass("DataAvailabilityMember")
showClass("SosGetDataAvailability_1.0.0")
```

SosGetFeatureOfInterest_2.0.0-class

Class `"SosGetFeatureOfInterest_2.0.0"`

Description

Representation of a GetFeatureOfInterest operation request that needs to be sent to a Sensor Observation Service to retrieve the features of interest, i.e. the real world features that are observed and for which observations are taken.

Objects from the Class

Objects can be created by calls of the form `new("SosGetFeatureOfInterest_2.0.0", ...)`.

Slots

`featureOfInterest:` Object of class `"character"` ~~

`service:` Object of class `"character"` ~~

`request:` Object of class `"character"` ~~

`version:` Object of class `"character"` ~~

Extends

Class "OwsServiceOperation", directly.

Methods

checkRequest `signature(service = "SOS_2.0.0",operation = "SosGetFeatureOfInterest_2.0.0",verbose = "logical")`: ...

encodeRequestKVP `signature(obj = "SosGetFeatureOfInterest_2.0.0")`: ...

sosName `signature(obj = "SosGetFeatureOfInterest_2.0.0")`: ...

Examples

```
showClass("SosGetFeatureOfInterest_2.0.0")
```

---

sosObservableProperties-methods

Methods for Function `sosObservableProperties` *in Package* **sos4R**

Description

Methods for function `sosObservableProperties` in package **sos4R**. The function allows to retrieve observable properties, e.g. air temperature, wind speed, etc., listed in the capabilities of a Sensor Observation Service and other classes.

Methods

`signature(obj = "list")` List of objects to retreive properties from.

`signature(obj = "OmObservation")` Get observable properties from an object of class `OmObservation`.

`signature(obj = "OmObservationCollection")` Get observable properties from an object of class `OmObservationCollection`, namely the items in the collection.

`signature(obj = "SOS")` Get observable properties for a whole connection to an SOS.

`signature(obj = "SosObservationOffering_2.0.0")` Get observable properties from an object of class `SosObservationOffering_2.0.0`, needed internally for getting properties for a whole SOS.

`signature(obj = "SweCompositePhenomenon")` Get observable properties from an object of class `SweCompositePhenomenon`.

`signature(obj = "SwePhenomenonProperty")` Get observable properties from an object of class `SwePhenomenonProperty`.

SosObservationOffering-class
Classes and Related Functions for "SosObservationOffering"

Description

SosObservationOfferings collect all metadata about a specific offerign in a Sensor Observation Service.

Usage

```
SosObservationOffering(id, name = as.character(NA), time, procedure, observedProperty,
                       featureOfInterest, responseFormat,
                       intendedApplication = as.character(NA),
                       resultModel = as.character(NA),
                       responseMode = as.character(NA), boundedBy = list())
```

Arguments

boundedBy	See the corresponding slot description.
featureOfInterest	See the corresponding slot description.
id	See the corresponding slot description.
intendedApplication	See the corresponding slot description.
name	See the corresponding slot description.
observedProperty	See the corresponding slot description.
procedure	See the corresponding slot description.
responseFormat	See the corresponding slot description.
responseMode	See the corresponding slot description.
resultModel	See the corresponding slot description.
time	See the corresponding slot description.

Details

ObservationOffering provides a mechanism for factoring groups of related observations within a single service instance. A functionally equivalent outcome could be obtained by factoring between different service instances.

Value

The construction functions return an object of the respective class, e.g. SosObservationOffering.

Objects from the Class

Objects can be created by calls to the construction functions of the form `SosObservationOffering(...)`.

Slots

`boundedBy`: Object of class `"list"`: A bounding box that contains all features in this offering.

`featureOfInterest`: Object of class `"list"`: Features or feature collections that represent the identifiable object(s) on which the sensor systems are making observations.

`id`: Object of class `"character"`: Identifier of an offering.

`intendedApplication`: Object of class `"list"`: The intended category of use for this offering such as homeland security or natural resource planning

`name`: Object of class `"character"`: The name of an offering.

`observedProperty`: Object of class `"list"`: The observable/phenomenon that can be requested in this offering.

`procedure`: Object of class `"list"`: A reference to one or more procedures, including sensor systems, instruments, simulators, etc, that supply observations in this offering. The DescribeSensor operation can be called to provide a detailed description of each system.

`responseFormat`: Object of class `"list"`: MIME type of the data that will be returned as the result of a GetObservation request. This is usually text/xml; subtype="om/0.0.0".

`responseMode`: Object of class `"list"`: Indicates what modes of response are supported for this offering. The value of resultTemplate is used to retrieve an observation template that will later be used in calls to GetResult. The other options allow results to appear inline in a resultTag (inline), external to the observation element (out-of-band) or as a MIME attachment (attached).

`resultModel`: Object of class `"list"`: Indicates the namespace-qualified name of the result element that will be included in the document returned from a call to GetObservation for this offering, e.g. ‘`"om:Observation"`’ or ‘`"om:Measurement"`’.

`time`: Object of class `"GmlTimeGeometricPrimitive"`: Time period for which observations can be obtained. This supports the advertisement of historical as well as real-time observations.

Methods

sosTime `signature(obj = "SosObservationOffering")`: Accessor function for the time slot, or to be more precise: the time period for which this offering provides data.

Author(s)

Daniel Nuest <daniel.nuest@uni-muenster.de>

References

See section 6.3, “Observation Offerings”, of the SOS specification: Na, A., Priest, M. (Eds.), Sensor Observation Service, Open Geospatial Consortium Inc., OGC 06-009r6, Version: 1.0

See Also

`SosContents`, `SosCapabilities`

Examples

```
showClass("SosObservationOffering")
showClass("SosObservationOffering_2.0.0")

# explore offerings of an SOS
mySOS <- SOS(url = "http://sensorweb.demo.52north.org/sensorwebtestbed/service/kvp",
             binding = "KVP")
offering1 <- sosOfferings(mySOS)[[1]]

sosId(offering1)
sosName(offering1)
sosTime(offering1)
sosBoundedBy(offering1)
```

---

sosRequest-methods *Send Request to SOS*

Description

This is the main request function for sending and receiving requests respectively documents from a Sensor Observation Service. It's intended for internal use.

Please use the methods for the SOS operations as long as possible: getCapabilities-methods, describeSensor-methods, getObservation-methods, and getObservationById-methods.

Methods

`signature(sos = "SOS", request = "OwsServiceOperation", verbose = "logical", inspect = "logical")`
: Method sends the given operation to the given SOS connection. `verbose` activates extensive debugging to the console. `inspect` prints only the request and response documents to the console.

Supported *Functions to Access Supported Features of the Current sos4R Implementation*

Description

These functions can be used to access the supported parameters for a range of settings of a SOS connection.

Usage

```
SosSupportedOperations(version = sos100_version)
SosSupportedComparisonOperators()
SosSupportedBindings()
SosSupportedGeometryOperands()
SosSupportedResponseFormats()
SosSupportedResponseModes()
SosSupportedResultModels()
SosSupportedSpatialOperators()
SosSupportedTemporalOperators()
SosSupportedServiceVersions()
```

Arguments

version	The SOS specification version.

Details

Supported features, like connection methods and supported response modes, are accessible by functions starting with "SosSupported". See the examples section for a complete list of these functions.

It is encouraged to rather use these methods than manually set character values for compatibility with future versions, e.g. `SosSupportedBindings()[[1]]` instead of directly writing `"GET"`.

Value

A list of supported values for the respective parameter.

Author(s)

Daniel Nuest <daniel.nuest@uni-muenster.de>

See Also

See `Defaults` for default values of parameters.

Examples

```
# The supported operations of the specification
SosSupportedOperations()

# HTTP connection methods supported by this sos4R implementation
SosSupportedBindings()
myBinding <- SosSupportedBindings()[[1]]
myBinding

# Formats, modes and models that can be processed by this implementation
SosSupportedResponseFormats()
SosSupportedResultModels()
SosSupportedResponseModes()
```

```
# Operators and operands for filtering in a GetObservation request
SosSupportedTemporalOperators()
SosSupportedSpatialOperators()
SosSupportedGeometryOperands()
SosSupportedComparisonOperators()
```

SWE *Classes and Construction Functions for the SWE Namespace*

Description

These classes represent elements from the OpenGIS(R) Sensor Model Language (SensorML) Implementation Specification that are used to model observation data in responses from a Sensor Observation Service.

Usage

```
SweCompositePhenomenon(id, name, description = as.character(NA), dimension,
                       components, base = NULL)
SwePhenomenon(id, name, description = as.character(NA))
SwePhenomenonProperty(href = as.character(NA), phenomenon = NULL)
SweTextBlock(tokenSeparator, blockSeparator, decimalSeparator, id = as.character(NA))
```

Arguments

	Arguments of the construction functions are as follows.
	The character string to be used for the id attribute (mandatory).
id name	The character string to be used for the name element (mandatory).
description	The character string to be used for the description element.
dimension	The dimensions of a composite phenomenon (mandatory).
components	The (sub-) components of a composite phenomenon (mandatory).
base	The (optional) base element for a composite phenomenon.
href	A reference to an (online) object instead of a inline property.
phenomenon	The inline phenomenon of a phenomenon property.
tokenSeparator	The character to be used as the token seperator, often ",".
blockSeparator	The character to be used as the block seperator, often ";".
decimalSeparator	The character to be used as the decimal seperator, often ".".

Details

The ...OrNULL classes are used to model optional slots.

Value

The construction functions return an object of the respective class.

Objects from the Class

Objects can be created by calls to the constrction functions of the form `SweCompositePhenomenon( ...)`, `SwePhenomenonProperty( ...)` and so forth.

The following classes are virtual, no objects may be created from them: `SwePhenomenonOrNULL`, `SwePhenomenonPropertyOrNULL`.

Slots

`dimension`**:** Object of class `"integer"`, the value of the dimensions attribute of a composite phenomenon.

`components`**:** Object of class `"list"`, the components of a composite phenomenon.

`base`**:** Object of class `"SwePhenomenonPropertyOrNULL"`, the base of a composite phenomenon, may be `NULL`.

`id`**:** Object of class `"character"`, the value of the id attribute of a (composite) phenomenon.

`name`**:** Object of class `"character"`, the value of the name element of a (composite) phenomenon.

`description`**:** Object of class `"character"`, the value of the description elements of a phenomenon.

`href`**:** Object of class `"character"`, the value of the href attribute of a phenomenon property which references a phenomenon.

`phenomenon`**:** Object of class `"SwePhenomenonOrNULL"`, the actual (inline) phenomenon of a phenomenon property.

`tokenSeparator`**:** Object of class `"character"`, the symbol to be used as the token seperator in a `SweTextBlock`, e.g. in the case of `","` this would result in attribute1,attribute2.

`blockSeparator`**:** Object of class `"character"`, the symbol to be used as the block seperator in a `SweTextBlock`, e.g. in the case of `";"` this would result in attribute1_a,attribute2_a;attribute1_b,attribute2_b.

`decimalSeparator`**:** Object of class `"character"`, the symbol to be used as the decimal seperator in a `SweTextBlock`, e.g. in the case of `"."` this would result in attribute1,42.0,attribute3,23.0.

Extends

SweCompositePhenomenon: Class `"SwePhenomenon"`, directly. Class `"SwePhenomenonOrNULL"`, by class "SwePhenomenon", distance 2.

SwePhenomenon: Class `"SwePhenomenonOrNULL"`, directly.

SwePhenomenonProperty: Class `"SwePhenomenonPropertyOrNULL"`, directly.

Methods

show `signature(object = "SweCompositePhenomenon")`: ...

show `signature(object = "SwePhenomenon")`: ...

show `signature(object = "SwePhenomenonProperty")`: ...

show `signature(object = "SweTextBlock")`: ...

Author(s)

Daniel Nuest <daniel.nuest@uni-muenster.de>

References

See section 9, SWE Common XML Encoding and Examples, of Botts, M., Robin, A. (Eds.), OpenGIS(R) Sensor Model Language (SensorML) Implementation Specification, Open Geospatial Consortium Inc., OGC 07-000

Examples

```
showClass("SweCompositePhenomenon")
showClass("SwePhenomenon")
showClass("SwePhenomenonProperty")
showClass("SwePhenomenonPropertyOrNULL")
showClass("SweTextBlock")
```

SweTextEncoding-class *Class* "SweTextEncoding"

Description

Representation of a text encoding defined in the OGC SWE Common specification. It defines the token, block, and decimal seperators for a text-encoded array of values.

Objects from the Class

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

Slots

tokenSeparator: Object of class "character" ~~

blockSeparator: Object of class "character" ~~

decimalSeparator: Object of class "character" ~~

id: Object of class "character" ~~

Methods

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

Examples

```
showClass("SweTextEncoding")
```

TM_Operators *Classes and Construction Methods for Temporal Operator Classes*

Description

Classes for temporal operators from OpenGIS(R) Filter Encoding used in filters in GetObservation requests.

Usage

```
TM_After(propertyName = sosDefaultTempOpPropertyName, time)
TM_Before(propertyName = sosDefaultTempOpPropertyName, time)
TM_During(propertyName = sosDefaultTempOpPropertyName, time)
TM_Equals(propertyName = sosDefaultTempOpPropertyName, time)
```

Arguments

propertyName	The name of the property that is used to wrap the time.
time	A time instant or period to be used as the temporal operand.

Value

An object of the respective class, so TM_After, TM_Before, TM_During or TM_Equals

Objects from the Class

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

Slots

time: Object of class "GmlTimeInstant"

propertyName: Object of class "character"

Extends

Class "OgcBinaryTemporalOp", directly. Class "OgcBinaryTemporalOpOrNULL", by class "OgcBinaryTemporalOp", distance 2.

Methods

encodeXML signature(obj = "TM_After", sos = "SOS"): ...

show signature(object = "TM_After"): ...

encodeXML signature(obj = "TM_Before", sos = "SOS"): ...

show signature(object = "TM_Before"): ...

encodeXML signature(obj = "TM_During", sos = "SOS"): ...

show signature(object = "TM_During"): ...

encodeXML `signature(obj = "TM_Equals", sos = "SOS")`: ...

show `signature(object = "TM_Equals")`: ...

Author(s)

Daniel Nuest <daniel.nuest@uni-muenster.de>

References

Vretanos, Panagiotis A. (Ed.), OpenGIS(R) Filter Encoding Implementation Specification, OGC 04-095, Version: 1.1.0

See the schema file: http://schemas.opengis.net/sos/1.0.0/ogc4sos.xsd.

See Also

SosGetObservation

Examples

```
showClass("TM_After")
showClass("TM_Before")
showClass("TM_During")
showClass("TM_Equals")

## Not run:
# create times to use for operators
t1 <- sosCreateTimeInstant(sos = weathersos, time = Sys.time())
p1 <- sosCreateTimePeriod(sos = weathersos, begin = as.POSIXct("2010-03-01 12:15"),
                          end = as.POSIXct("2010-03-02 12:15"))

# create temporal operator
afterNow <- TM_After(time = t1)
print(afterNow)
encodeXML(t1, sos)

during <- TM_During(time = p1)
print(during)

## End(Not run)
```

---

xml_document-class *Classes to model S3 classes of package* xml2

Description

Currently the main classes of the XML handling package xml2 are only S3 classes and thus cannot be used in slots for S4 classes. To remedy this, classes can be registered with `setOldClass(..)`. This is done within this package, but should really be part of xml2, see https://github.com/r-lib/xml2/issues/248

Author(s)

Daniel Nuest

Index