Accessing Data from Sensor Observation Services: the **sos4R** Package

Daniel Nüst*
daniel.nuest@uni-muenster.de
http://www.nordholmen.net/sos4r

December 28, 2010

Abstract

The sos4R package provides easy and simple, yet powerful access to OGC Sensor Observation Service instances. The package supports both encapsulation and abstraction from the service interface for novice users as well as powerful request building for specialists.

sos4R is motivated by the idea to add a missing link between the Sensor Web and tools (geo-)statistical analyses. It implements the core profile of the SOS specification and supports temporal, spatial, and thematical filtering of observations. This document briefly introduces the SOS specification. The package's features are explained extensively: exploration of service metadata, request building with filters, function exchangeability, result data transformation.

The package is published under GPL 2 license within the geostatistics community of 52°North Initiative for Geospatial Open Source Software.

Contents

*Institute for Geoinformatics, University of Muenster, Germany.

1 Introduction

The **sos4R** package provides classes and methods for retrieving data from an OGC Sensor Observation Service (Na, 2007). The goal of this package is to provide easy access with a low entry threshold for everyone to information available via SOSs. The complexity of the service interface shall be shielded from the user as much as possible, while still leaving enough possiblities for advanced users. At the current state, the output is limited to a standard data.frame with attributed columns for metadata. In future releases a tighter integration is planned with upcoming space-time packages regarding data structures and classes. This package uses S4 classes and methods style (Chambers, 1998).

The motivation to write this package was born out of perceiving a missing link between the Sensor Web community (known as Sensor Web Enablement (SWE) Initiative[1] in the OGC realm) and the community of (geo-)statisticians. While the relatively young SWE standards get adopted more by data owners (like governmental organizations), we see a high but unused potential for more open data and spatio-temporal analyses based on it. **sos4R** can help enabling this.

The project is part of the geostatistics community[2] of the 52 °North Initiative for Geospatial Open Source Software[3]. **sos4R** is available, or will be available soon, on CRAN.

[1] http://www.opengeospatial.org/projects/groups/sensorweb
[2] http://52north.org/communities/geostatistics/
[3] http://52north.org/

On the package home page, http://www.nordholmen.net/sos4r/, you can stay updated with the development blog, find example code and services, and download source packages.

This software is released under a GPL 2 license[4] and contributions are very welcome. Please consult section 9 for details.

The package **sos4R** is loaded by

```
> library("sos4R")
```

This document was build for **package version 0.1-08**.

1.1 Related Specifications

The Open Geospatial Consortium[5] (OGC) is an organisation which provides standards for handling geospatial data on the internet, thereby ensuring interoperability.

The Sensor Observation Service (SOS) is such a standard and provides a well-defined interface for data warehousing of measurements and observations made by all kinds of sensors. This vignette describes the classes, methods and functions provided by **sos4R** to query these observations.

Storing and providing data in web services is more powerful than local file copies (with issues like outdating, redundancy, ...). Flexible filtering of data on the service side reduces download size. That is why SOS operations can comprise flexible subsetting in temporal, spatial and thematical domain. For example "Provide only measurements from sensor MySensor-001 for the time period from 01/12/2010 to 31/12/2010 where the air temperature below zero degrees".

In general, the SOS supports two methods of requesting data: (i) HTTP GET as defined in the OOSTethys best practice document[6], and (ii) POST as defined in the standard document. Both request types always returns eXtensible Markup Language (XML) documents as response.

Standards that are referenced, respectively used, by SOS are as follows.

Observations and Measurements (O&M) O&M (Cox, 2007) defines the markup of sensor measurements results. An observation consists of information about the observerd geographic feature, the time of observation, the sensor, the observed phenomenon, and the observation's actual result.

Sensor Model Language (SensorML) SensorML (Botts, 2007) is used for sensor metadata descriptions (calibration information, inputs and outputs, maintainer).

Geography Markup Language (GML) (Portele, 2003) defines markup for geographical features (points, lines, polygons, ...).

SweCommon SWE Common describes data markup and is contained in the SensorML specification.

Filter Encoding Filter Encoding (Vretanos, 2005) defines operators and operands for filtering values.

[4] http://www.gnu.org/licenses/gpl-2.0.html
[5] http://www.opengeospatial.org/
[6] http://www.oostethys.org/best-practices/best-practices-get

OWS Common OGC Web Services Common (Whiteside, 2007) models service related elements that are reusable across several service specifications, like excheption handling.

1.2 Terms and Definitions

The OGC has a particular set of well-defined terms that might differ from usage of words in specific domains. The most important are as follows[7].

Feature of Interest (FOI) The FOI represents the geo-object, for which measurements are made by sensors. It is ordinarily used for the spatial referencing of measuring points, i.e. the geoobject has coordinates like latitude, longitude and height. The feature is project specific and can be anything from a point (e.g. the position of a measuring station) or a real-world object (e.g. the region that is observed).

Observation The observation delivers a measurement (result) for a property (phenomenon) of an observed object (FOI). The actual value is created by a sensor or procedure. The phenomenon was measured at a specific time (sampling time) and the value was generated at a specific point in time (result time). These often coincide so in practice the sampling time is often used as the point in time of an obsrevation.

Offering The offering is a logical collection of related observations (similar to a layer in mapping applications) which a service offers together.

Phenomenon A phenomenon is a property (physical value) of a geographical object, e.g. air temperature, wind speed, concentration of a pollutant in the athmosephere, reflected radiation in a specific frequency band (colours).

Procedure A procedure creates the measurement value of an observation. The source can be a reading from a sensor, simulation or a numerical process.

A more extensive discussion is available in the the O&M specification (Cox, 2007). The Annex B of that document shows the following examples of applicating some terms to a specific domain, earth observations, which are repeated here for elaboration.

O&M	Particulate Matter 2.5 Concentrations	EO
Observation::result	35 ug/m3	observation value, measurement value
Observation::procedure	U.S. EPA Federal Reference Method for PM 2.5	method, sensor
Observation::observedProperty	Particulate Matter 2.5	parameter, variable
Observation::featureOfInterest	troposphere	media (air, water, ...)
Global Change Master Directory "Topic"		

2 Supported Features

The package provides accessor functions for the supported parameters. It is recommended to access options from the lists returned by these functions instead of hardcoding them into scripts.

```
> SosSupportedOperations()
```

[7] Based on http://de.wikipedia.org/wiki/Sensor_Observation_Service

```
[1] "GetCapabilities"    "DescribeSensor"     "GetObservation"
[4] "GetObservationById"
```

```
> SosSupportedServiceVersions()
```

```
[1] "1.0.0"
```

```
> SosSupportedConnectionMethods()
```

```
   GET   POST
 "GET" "POST"
```

```
> SosSupportedResponseFormats()
```

```
[1] "text/xml;subtype=&quot;om/1.0.0&quot;"
[2] "text/xml;subtype=&quot;sensorML/1.0.1&quot;"
```

```
> SosSupportedResponseModes()
```

```
[1] "inline"
```

```
> SosSupportedResultModels()
```

```
[1] "om:Measurement" "om:Observation"
```

```
> SosSupportedSpatialOperators()
```

```
$BBOX
[1] "BBOX"

$Contains
[1] "Contains"

$Intersects
[1] "Intersects"

$Overlaps
[1] "Overlaps"

$BBOX
[1] "BBOX"

$Contains
[1] "Contains"

$Intersects
[1] "Intersects"

$Overlaps
[1] "Overlaps"
```

```
> SosSupportedTemporalOperators()
```

```
$TM_After
[1] "TM_After"

$TM_Before
[1] "TM_Before"

$TM_During
[1] "TM_During"

$TM_Equals
[1] "TM_Equals"

$TM_After
[1] "TM_After"

$TM_Before
[1] "TM_Before"

$TM_During
[1] "TM_During"

$TM_Equals
[1] "TM_Equals"
```

2.1 Supported Services and Implementations

sos4R supports the core profile of the SOS specification. But the possible markups for observations is extremely manifold due to the flexibility of the O&M specification. Sadly, there is no common application profile for certain types of observations, like simple measurements.

Therefore, the undocumented profile of the 52°North **SOS implementation**[8] was used as a guideline. It is not documented outside of the source code. Observations returned by instances of this implementation are most likely to be processed out of the box.

In the author's experience, **OOSThetys SOS implementations**[9] utilize the same or at least very similar profile, so responses of these service instances are also probably parsed without further work.

Please share your experiences with other SOS implementations with the developers and users of **sos4R** (see section **??**Default Options

Two kinds of default values can be found in (function calls in) **sos4R**: (i) default depending on other function parameters, and (ii) global defaults. Global defaults can be inspected (not set!) using the following functions. If you want to use a different value please adapt the respective argument in function calls.

```
> SosDefaultConnectionMethod()

[1] "POST"

> SosDefaults()
```

[8] http://52north.org/communities/sensorweb/sos/
[9] http://www.oostethys.org/

```
$sosDefaultCharacterEncoding
[1] "UTF-8"

$sosDefaultDescribeSensorOutputFormat
[1] "text/xml;subtype=&quot;sensorML/1.0.1&quot;"

$sosDefaultGetCapSections
[1] "All"

$sosDefaultGetCapAcceptFormats
[1] "text/xml"

$sosDefaultGetCapOwsVersion
[1] "1.1.0"

$sosDefaultGetObsResponseFormat
[1] "text/xml;subtype=&quot;om/1.0.0&quot;"

$sosDefaultTimeFormat
[1] "%Y-%m-%dT%H:%M:%OS"

$sosDefaultTempOpPropertyName
[1] "om:samplingTime"

$sosDefaultTemporalOperator
[1] "TM_During"

$sosDefaultSpatialOpPropertyName
[1] "urn:ogc:data:location"

$sosDefaultColumnNameFeatureIdentifier
[1] "feature"

$sosDefaultColumnNameLat
[1] "lat"

$sosDefaultColumnNameLon
[1] "lon"

$sosDefaultColumnNameSRS
[1] "SRS"
```

The process of data download also comprises (i) building requests, (ii) decoding responses, and (iii) applying the correct R data type to the respective data values. This mechanism is explained in detail in see section 5. The package comes with a set of predefined encoders, decoders and converters.

```
> SosEncodingFunctions()
> SosParsingFunctions()
> SosDataFieldConvertingFunctions()
```

3 Creating a SOS connection

The operation `SOS(...)` is a construction method for classes encapsulating a connection to a SOS. It prints out a short statement when the connection was successful and returns an object of class `SOS`.

```
> mySOS = SOS(url = "http://v-swe.uni-muenster.de:8080/WeatherSOS/sos")
Created SOS for URL http://v-swe.uni-muenster.de:8080/WeatherSOS/sos
```

To create a SOS connection you only need the URL of the service (i.e. the URL which can be used for HTTP GET or POST requests).

The optional parameters use default settings (see section 2.1):

- `method`: The transport protocol. Currently available are GET.
- `version`: The service version. Currently allowed are Currenlty available are 1.0.0.
- `parsers`: The list of parsing functions. See section 5.1.
- `encoders`: The list of encoding functions. See section 5.2.
- `dataFieldConverters`: The list of conversion functions. See section 5.3.
- `curlHandle`, `curlOptions`: Settings of the package **RCurl**, which is used for HTTP connections. Please consult the packags specification before using this.
- `timeFormat`: The time format to be used or decoding and encoding time character strings to and from `POSIXt` classes.
- `verboseOutput`: Trigger parameter for extensive debugging information on the console, see section 6.2.

There are accessor methods for the slots of the class.

```
> sosUrl(mySOS)
[1] "http://v-swe.uni-muenster.de:8080/WeatherSOS/sos"
> sosVersion(mySOS)
[1] "1.0.0"
> sosTimeFormat(mySOS)
[1] "%Y-%m-%dT%H:%M:%OS"
> sosMethod(mySOS)
[1] "POST"
```

The following slots are best described in section 5.

```
> sosParsers(mySOS)
> sosDataFieldConverters(mySOS)
```

The default connection method is HTTP POST, but since not all SOS support this a GET connection is possible was well. The latter is partly limited, for example regarding filtering operations. Section 5.3 contains an example of such a connection.

4 SOS Operations

sos4R implements the SOS core profile of version 1.0.0 comprising the operations GetCapabilities, DescribeSensor and GetObservation. This document focusses on the practical usage of the operations, so the reader is refered to the specification document for details.

The methods mirroring the SOS operations all contain debugging parameters `inspect` and `verbose` as described in section 6.2.

4.1 GetCapabilities

The GetCapabilities operations is automatically conducted during the connecting to a SOS instance. If you want to inspect the original capabilities document it can be re-requested using

```
> sosCapabilitiesDocumentOriginal(sos = mySOS)
```

The actual operation can be started with the following function. It returns an object of class `SosCapabilities` which can be accessed later on by the function `sosCaps()` from an object of class `SOS`.

```
> getCapabilities(sos = mySOS)
```

- `sos`: The SOS connection to request the capabilities document from.
- `inspect` and `verbose`: See section 6.2.

4.2 DescribeSensor

The DescribeSensor operation is specified in clause 8.3 of the SOS specification and their response is modeled in Sensor Model Language[10] (SensorML) and Transducer Markup Language[11] (TML) specifications.

> The DescribeSensor operation is useful for obtaining detailed information of sensor characteristics encoded in either SensorML or TML. The sensor characteristics can include lists and definitions of observables supported by the sensor. [...]

The parameters of the operation are the following:

- `sos`: The SOS connection to request a sensor description from.
- `procedure`: The identifier of the sensor, so one of the character strings returned by `sosProcedures(...)`.
- `outputFormat`: The format in which the sensor description is to be returned by the service. The default value is `text/xml;subtype=quot;sensorML/1.0.1quot;`.
- `inspect` and `verbose`: See section 6.2.

```
> sensor.1.1 <- describeSensor(sos = mySOS,
+                  procedure = sosProcedures(obj = mySOS)[[1]][[1]])

Object of class SensorML (wraps unparsed XML, see @xml for details).
```

[10] http://www.opengeospatial.org/standards/sensorml
[11] http://www.opengeospatial.org/standards/tml

4.3 GetObservation

A few utility functions exist to minize a user's amount of work to create usual requests. They accept normal R types as input and return the respective class from **sos4R** with useful default settings. These function's names start with `sosCreate...()` and exist for spatial and temporal filters.

In this section, all matters around requesting data are explained — from extracting query parameters from metadata, and sending the request, till finally extracting data values and coordinates from the response.

4.3.1 Metadata Extraction for Request Building

How can one extract the metadata from a SOS connection and reuse it for queries?

accessor functions, elements of the capabilities, ...

TODO: jedes statement einzeln und erklaeren...

```
> sosContents(mySOS)

Object of class SosContents with observation offerings (names):
         RAIN_GAUGE, LUMINANCE, HUMIDITY, ATMOSPHERIC_PRESSURE, ATMOSPHERIC_TEMPERATURE, W

> sosFilter_Capabilities(mySOS)

Object of class SosFilter_Capabilities;
        Spatial_Capabilities:         gml:Envelope, gml:Point, gml:LineString, gml:Polygon
        Temporal_Capablities:         gml:TimePeriod, gml:TimeInstant ;
        Scalar_Capablities:                 Between, EqualTo, NotEqualTo, LessThan, LessTh
        Id_Capabilities                         FID, EID

> sosServiceIdentification(mySOS)

Object of class OwsServiceIdentification:
        ServiceType:  OGC:SOS ; serviceTypeVersion(s):  1.0.0
        title(s):  IFGI WeatherSOS
        Profile(s):
        Abstract(s):  SOS for weather observations at IFGI, Muenster, Germany (SVN: 9075 @
        Keywords(s):  , temperature, humidity, wind speed, luminance, wind, wind direction
        AccessConstraints(s):  WeatherSOS data is made available under the Open Data Commo

> sosServiceProvider(mySOS)

Object of class OwsServiceProvider:
        Provider name:  52North ; providerSite:  http://52north.org/swe
        Service contact:  (unparsed XML, see @serviceContact for details)

> #sosOfferings(mySOS)
> off.temp <- sosOfferings(mySOS)[["ATMOSPHERIC_TEMPERATURE"]]

Object of class SosObservationOffering;  id:  ATMOSPHERIC_TEMPERATURE , name:  Temperature
        time:  GmlTimePeriod: [ GmlTimePosition [ time: 2008-11-20 15:20:22 ]  -->  GmlTim
        procedure(s):  urn:ogc:object:feature:OSIRIS-HWS:3d3b239f-7696-4864-9d07-15447eae2
        observedProperty(s):  urn:ogc:def:property:OGC::Temperature
```

```
        feature(s)OfInterest:  urn:ogc:object:feature:OSIRIS-HWS:3d3b239f-7696-4864-9d07-1
        responseFormat(s):  text/xml;subtype="om/1.0.0", application/zip , responseMode(s)
        intendedApplication:  NA
        resultModel(s):  ns:Measurement, ns:Observation
        boundedBy:  urn:ogc:def:crs:EPSG:4326, 46.611644 7.6103, 51.9412 13.883498

> # the order of offerings can change in between requests
>
> sosOfferingIds(mySOS)

[1] "RAIN_GAUGE"              "LUMINANCE"
[3] "HUMIDITY"                "ATMOSPHERIC_PRESSURE"
[5] "ATMOSPHERIC_TEMPERATURE" "WIND_SPEED"
[7] "WIND_DIRECTION"

> # Names of offerings list are the Ids (same output as last call)
> # names(sosOfferings(mySOS))
> sosId(off.temp)

[1] "ATMOSPHERIC_TEMPERATURE"

> sosOfferings(mySOS)[1:2]

$RAIN_GAUGE
Object of class SosObservationOffering;  id:  RAIN_GAUGE , name:  Rain
        time:  GmlTimePeriod: [ GmlTimePosition [ time: 2008-11-20 15:35:22 ]  -->  GmlTim
        procedure(s):  urn:ogc:object:feature:OSIRIS-HWS:3d3b239f-7696-4864-9d07-15447eae2
        observedProperty(s):  urn:ogc:def:property:OGC::Precipitation1Hour
        feature(s)OfInterest:  urn:ogc:object:feature:OSIRIS-HWS:3d3b239f-7696-4864-9d07-1
        responseFormat(s):  text/xml;subtype="om/1.0.0", application/zip , responseMode(s)
        intendedApplication:  NA
        resultModel(s):  ns:Measurement, ns:Observation
        boundedBy:  urn:ogc:def:crs:EPSG:4326, 46.611644 7.6103, 51.9412 13.883498

$LUMINANCE
Object of class SosObservationOffering;  id:  LUMINANCE , name:  Luminance
        time:  GmlTimePeriod: [ GmlTimePosition [ time: 2008-11-20 15:20:22 ]  -->  GmlTim
        procedure(s):  urn:ogc:object:feature:OSIRIS-HWS:3d3b239f-7696-4864-9d07-15447eae2
        observedProperty(s):  urn:ogc:def:property:OGC::Luminance
        feature(s)OfInterest:  urn:ogc:object:feature:OSIRIS-HWS:3d3b239f-7696-4864-9d07-1
        responseFormat(s):  text/xml;subtype="om/1.0.0", application/zip , responseMode(s)
        intendedApplication:  NA
        resultModel(s):  ns:Measurement, ns:Observation
        boundedBy:  urn:ogc:def:crs:EPSG:4326, 46.611644 7.6103, 51.9412 13.883498

> sosProcedures(mySOS)

$RAIN_GAUGE
[1] "urn:ogc:object:feature:OSIRIS-HWS:3d3b239f-7696-4864-9d07-15447eae2b93"
[2] "urn:ogc:object:feature:OSIRIS-HWS:efeb807b-bd24-4128-a920-f6729bcdd111"
```

```
$LUMINANCE
[1] "urn:ogc:object:feature:OSIRIS-HWS:3d3b239f-7696-4864-9d07-15447eae2b93"
[2] "urn:ogc:object:feature:OSIRIS-HWS:efeb807b-bd24-4128-a920-f6729bcdd111"

$HUMIDITY
[1] "urn:ogc:object:feature:OSIRIS-HWS:3d3b239f-7696-4864-9d07-15447eae2b93"
[2] "urn:ogc:object:feature:OSIRIS-HWS:efeb807b-bd24-4128-a920-f6729bcdd111"

$ATMOSPHERIC_PRESSURE
[1] "urn:ogc:object:feature:OSIRIS-HWS:3d3b239f-7696-4864-9d07-15447eae2b93"
[2] "urn:ogc:object:feature:OSIRIS-HWS:efeb807b-bd24-4128-a920-f6729bcdd111"

$ATMOSPHERIC_TEMPERATURE
[1] "urn:ogc:object:feature:OSIRIS-HWS:3d3b239f-7696-4864-9d07-15447eae2b93"
[2] "urn:ogc:object:feature:OSIRIS-HWS:efeb807b-bd24-4128-a920-f6729bcdd111"

$WIND_SPEED
[1] "urn:ogc:object:feature:OSIRIS-HWS:3d3b239f-7696-4864-9d07-15447eae2b93"
[2] "urn:ogc:object:feature:OSIRIS-HWS:efeb807b-bd24-4128-a920-f6729bcdd111"

$WIND_DIRECTION
[1] "urn:ogc:object:feature:OSIRIS-HWS:3d3b239f-7696-4864-9d07-15447eae2b93"
[2] "urn:ogc:object:feature:OSIRIS-HWS:efeb807b-bd24-4128-a920-f6729bcdd111"
```

```
> sosProcedures(off.temp)
```

```
[1] "urn:ogc:object:feature:OSIRIS-HWS:3d3b239f-7696-4864-9d07-15447eae2b93"
[2] "urn:ogc:object:feature:OSIRIS-HWS:efeb807b-bd24-4128-a920-f6729bcdd111"
```

```
> # the order of procedures can change in between requests
>
> sosObservedProperties(mySOS)
```

```
$RAIN_GAUGE
$RAIN_GAUGE$observedProperty
[1] "urn:ogc:def:property:OGC::Precipitation1Hour"


$LUMINANCE
$LUMINANCE$observedProperty
[1] "urn:ogc:def:property:OGC::Luminance"


$HUMIDITY
$HUMIDITY$observedProperty
[1] "urn:ogc:def:property:OGC::RelativeHumidity"


$ATMOSPHERIC_PRESSURE
$ATMOSPHERIC_PRESSURE$observedProperty
[1] "urn:ogc:def:property:OGC::BarometricPressure"
```

```
$ATMOSPHERIC_TEMPERATURE
$ATMOSPHERIC_TEMPERATURE$observedProperty
[1] "urn:ogc:def:property:OGC::Temperature"


$WIND_SPEED
$WIND_SPEED$observedProperty
[1] "urn:ogc:def:property:OGC::WindSpeed"


$WIND_DIRECTION
$WIND_DIRECTION$observedProperty
[1] "urn:ogc:def:property:OGC::WindDirection"

> sosObservedProperties(off.temp)

$observedProperty
[1] "urn:ogc:def:property:OGC::Temperature"

> # the order of observed properties can change in between requests
>
> sosBoundedBy(off.temp)

$srsName
[1] "urn:ogc:def:crs:EPSG:4326"

$lowerCorner
[1] "46.611644 7.6103"

$upperCorner
[1] "51.9412 13.883498"

> str(sosBoundedBy(off.temp)) # Nicht so schÃ■n ...

List of 3
 $ srsName    : chr "urn:ogc:def:crs:EPSG:4326"
 $ lowerCorner: chr "46.611644 7.6103"
 $ upperCorner: chr "51.9412 13.883498"
NULL

> sosTime(mySOS)

[[1]]
Object of class OwsRange; spacing:  NA , rangeClosure:  NA
FROM  2008-02-14T11:03:02.000+01:00  TO  2010-12-26T00:15:00.000+01:00

> off.temp.time <- sosTime(off.temp)

GmlTimePeriod: [ GmlTimePosition [ time: 2008-11-20 15:20:22 ]  -->  GmlTimePosition [ tim
```

```
> str(off.temp.time)

Formal class 'GmlTimePeriod' [package "sos4R"] with 9 slots
  ..@ begin        : NULL
  ..@ beginPosition:Formal class 'GmlTimePosition' [package "sos4R"] with 4 slots
  .. .. ..@ time                : POSIXlt[1:1], format: "2008-11-20 15:20:22"
  .. .. ..@ frame               : chr NA
  .. .. ..@ calendarEraName     : chr NA
  .. .. ..@ indeterminatePosition: chr NA
  ..@ end          : NULL
  ..@ endPosition  :Formal class 'GmlTimePosition' [package "sos4R"] with 4 slots
  .. .. ..@ time                : POSIXlt[1:1], format: "2010-12-26 00:15:00"
  .. .. ..@ frame               : chr NA
  .. .. ..@ calendarEraName     : chr NA
  .. .. ..@ indeterminatePosition: chr NA
  ..@ duration     : chr NA
  ..@ timeInterval : NULL
  ..@ frame        : chr NA
  ..@ relatedTimes : list()
  ..@ id           : chr NA
NULL

> # "wirklichen" Startzeitpunkt abfragen
> off.temp.time@beginPosition@time

[1] "2008-11-20 15:20:22"

> class(off.temp.time@beginPosition@time)

[1] "POSIXt"  "POSIXlt"
```

4.3.2 Basic Request and Result Extraction

```
> getObservation(sos = mySOS, ...)
```

`latest`: A boolean parameter to request the latest observation only (see example below) — this is not standard conform.

A request to retrieve the latest measured value is also possible, although not (!) standard conform. 52 °North SOS realizes this specific request by requesting a sampling time with the fixed value latest.

```
> obs.temp.latest <- getObservation(sos = mySOS, offering = off.temp, latest = TRUE)

Finished getObservation to http://v-swe.uni-muenster.de:8080/WeatherSOS/sos
        --> received 2 observation(s) having 2 result values [ 1, 1 ].
```

The returned data is an XML document of type om:Observation, om:Measurement, or om:ObservationCollection which holds a list of the former two. All three of these have corresponding S4 classes, namely `OmObservation`, `OmMeasurement`, or `OmObservationCollection`.

The elements in an `OmObservationCollection` can be accessed just like a normal list (in fact, it just wraps at list of observations at this point).

```
> # ObservationCollection behaves like a list in most cases
> length(obs.temp.latest)
> obs.temp.latest[[1]]
> obs.temp.latest[2:5]
> sosBoundedBy(obs.temp.latest)
```

```
> # ObservationCollection behaves like a list in most cases
> length(obs.temp.latest)
> obs.temp.latest[[1]]
> obs.temp.latest[2:5]
```

Data Values ... sosResult erklÃd'ren,

```
> # sosResult(...) ist die wichtigste Methode
> sosResult(obs.temp.latest[[2]])
> obs.temp.latest.result <- sosResult(obs.temp.latest[1:2])
> # Nur ein ganz normaler data.frame ... Attribute enthalten Metadaten. Diese
> # gehen nach dem "merge" verloren!
> attributes(obs.temp.latest.result[["urn:ogc:def:property:OGC::Temperature"]])
> # Kombination der results mit den Koordinaten
> obs.temp.latest.coords <- sosCoordinates(obs.temp.latest)
> obs.temp.latest.data <- merge(x = obs.temp.latest.result,
+                 y = obs.temp.latest.coords)
> obs.temp.latest.data
```

Spatial Information can be stored in an observation in several ways.
...

```
> # Koordinaten, Features und BoundingBox abfragen
> sosCoordinates(obs.temp.latest)
> sosCoordinates(obs.temp.latest[[1]])
> sosFeatureIds(obs.temp.latest)
> sosBoundedBy(obs.temp.latest)
```

4.3.3 Temporal Filtering

The possibly most typical temporal filter is a period of time for which measurements are of interest.

```
> # temporal interval creation based on POSIXt classes
> lastWeek.period <- sosCreateTimePeriod(sos = mySOS,
+                 begin = (Sys.time() - 3600 * 24 * 7), end = Sys.time())
> lastWeek.eventTime <- sosCreateEventTimeList(lastWeek.period)
```

Please note that the create function also wraps the created objects in a list as expected by the method `getObservation(...)`.

What was the average temperature during the last week?

```
> obs.lastWeek <- getObservation(sos = mySOS, offering = off.temp,
+                 procedure = sosProcedures(off.temp), eventTime = lastWeek.eventTime)
```

```
Finished getObservation to http://v-swe.uni-muenster.de:8080/WeatherSOS/sos
        --> received 1 observation(s) having 406 result values [ 406 ].
```

```
> obs.temp.lastWeek.result <- sosResult(obs.lastWeek)
> summary(obs.temp.lastWeek.result)[4,"urn:ogc:def:property:OGC::Temperature"]

[1] "Mean   :-3.498  "
```

The default temporal operator is "during", but others are supported as well (see section 2). The next example shows how to create a temporal filter for all observations taken **after** a certain point in time. Here the creation function creates just one object of class `SosEventTime` which must be added to a list manually before passing it to `getObservation(...)`.

```
> lastDay.instant <- sosCreateTimeInstant(
+                 time = as.POSIXct(Sys.time() - 3600 * 24), sos = mySOS)
> lastDay.eventTime <- sosCreateEventTime(time = lastDay.instant,
+                 operator = SosSupportedTemporalOperators()[["TM_After"]])
> print(lastDay.eventTime)

Object of class SosEventTime: TM_After: GmlTimePosition [ time: 2010-12-27 18:59:02 ]
```

4.3.4 Spatial Filtering

The possibly most typical spatial filter is a bounding box[12] within which measurements of interest must have been made. Here the creation function returns an object of class `OgcBBOX`, which can be wrapped in an object of class `SosFeatureOfInterest`, which is passed into the get-observation call.

```
> request.bbox <- sosCreateBBOX(lowLat = 10.0, lowLon = 2.0,
+                 uppLat = 50.0, uppLon = 15.0, srsName = "urn:ogc:def:crs:EPSG:4326")
> request.bbox.foi <- sosCreateFeatureOfInterest(spatialOps = request.bbox)
> obs.lastWeek.bbox <- getObservation(sos = mySOS,
+                 offering = off.temp,
+                 featureOfInterest = request.bbox.foi,
+                 eventTime = list(lastDay.eventTime))

Finished getObservation to http://v-swe.uni-muenster.de:8080/WeatherSOS/sos
        --> received 0 observation(s) having 0 result values [ 0 ].

> print(sosCoordinates(obs.lastWeek.bbox))

NULL
```

More advanced spatial filtering, for example based on arbitrary shapes et cetera, is currently not implemented.

4.3.5 Feature Filtering

The feature can not only be used for spatial filtering, but also to query specific FOIs. The following example extracts the identifiers from an offering and then creates an object of class `SosFeatureOfInterest`, which is passed into the get-observation call.

[12] http://en.wikipedia.org/wiki/Bounding_box

```
> off.temp.fois <- sosFeaturesOfInterest(off.temp)
> request.fois <- sosCreateFeatureOfInterest(
+                 objectIDs = list(off.temp.fois[[1]]))
> encodeXML(obj = request.fois, sos = mySOS)

<sos:featureOfInterest>
 <sos:ObjectID>urn:ogc:object:feature:OSIRIS-HWS:3d3b239f-7696-4864-9d07-15447eae2b93</sos
</sos:featureOfInterest>

> obs.lastWeek.fois <- getObservation(sos = mySOS, offering = off.temp,
+                 featureOfInterest = request.fois, eventTime = lastWeek.eventTime)

Finished getObservation to http://v-swe.uni-muenster.de:8080/WeatherSOS/sos
        --> received 1 observation(s) having 406 result values [ 406 ].

> print(sosFeaturesOfInterest(obs.lastWeek.fois))

$OmObservation
Object of class GmlFeatureCollection; id: NA; 1 featureMembers: <S4 object of class "GmlFe
```

4.3.6 Value Filtering

Value Filtering is realized via the slot `result` in a GetObservation request. The filtering in the request is based on comparison operators and operands specified by OGC Filter Encoding (Vretanos, 2005).

The classes and methods of this specification are not yet implemented, but manual definition of the XML elements is possible with the methods of the package **XML**.

The following code example uses a literal comparison of a property:

```
> # result filtering
> filter.value <- -2.3
> filter.propertyname <- xmlNode(name = ogcPropertyNameName,
+                 namespace = ogcNamespacePrefix)
> xmlValue(filter.propertyname) <- "urn:ogc:def:property:OGC::Temperature"
> filter.literal <- xmlNode(name = "Literal", namespace = ogcNamespacePrefix)
> xmlValue(filter.literal) <- as.character(filter.value)
> filter.comparisonop <- xmlNode(name = ogcComparisonOpGreaterThanName,
+                 namespace = ogcNamespacePrefix,
+                 .children = list(filter.propertyname, filter.literal))
> filter.result <- xmlNode(name = sosResultName, namespace = sosNamespacePrefix,
+                 .children = list(filter.comparisonop))
```

Please consult to the extensive documentation of the **XML** package for details. The commands above result in the following output which is inserted into the request without further processing.

```
> print(filter.result)

<sos:result>
 <ogc:PropertyIsGreaterThan>
  <ogc:PropertyName>urn:ogc:def:property:OGC::Temperature</ogc:PropertyName>
```

```
  <ogc:Literal>-2.3</ogc:Literal>
 </ogc:PropertyIsGreaterThan>
</sos:result>
NULL
```

Any object of class `OgcComparisonOpsOrXMLOrNULL`, which includes the class of the object returned by `xmlNode(...)`, i.e. `XMLNode`. These object can be used in the GetObservation request as the `result` parameter.

First, we request the unfiltered values for comparison, then again with the filter applied. The length of the returned results is compared in the end.

```
> # request  values for the last week.
> obs.lastWeek <- getObservation(sos = mySOS,
+                  eventTime = lastWeek.eventTime,
+                  offering = sosOfferings(mySOS)[["ATMOSPHERIC_TEMPERATURE"]])

Finished getObservation to http://v-swe.uni-muenster.de:8080/WeatherSOS/sos
        --> received 1 observation(s) having 406 result values [ 406 ].
Object of class OmObservationCollection with 1 members.

> # request  values for the week with a value higher than 0 degrees.
> obs.lastWeek.filter <- getObservation(sos = mySOS,
+                  eventTime = lastWeek.eventTime,
+                  offering = sosOfferings(mySOS)[["ATMOSPHERIC_TEMPERATURE"]],
+                  result = filter.result)

Finished getObservation to http://v-swe.uni-muenster.de:8080/WeatherSOS/sos
        --> received 1 observation(s) having 131 result values [ 131 ].
Object of class OmObservationCollection with 1 members.

> paste("Filtered:", dim(sosResult(obs.lastWeek.filter))[[1]],
+                                  "-vs.- Unfiltered:", dim(sosResult(obs.lastWeek))[[1]])

[1] "Filtered: 131 -vs.- Unfiltered: 406"
```

4.3.7 Result Exporting

A tighter integration with data structures of packages **sp** or **spacetime** (both available on CRAN) is planned for the future. Please consult the developers for the current status.

As an example the following code creates a `SpatialPointsDataFrame` (can only contain one data value per position!) based on the features of a result.

```
> library("sp")
> obs.lastWeek <- getObservation(sos = mySOS,
+                  offering = off.temp,
+                  procedure = sosProcedures(off.temp),
+                  eventTime = lastWeek.eventTime)

Finished getObservation to http://v-swe.uni-muenster.de:8080/WeatherSOS/sos
        --> received 1 observation(s) having 406 result values [ 406 ].
```

```
> # Create SpatialPointsDataFrame from result features
> coords <- sosCoordinates(obs.lastWeek[[1]])
> crs <- sosGetCRS(obs.lastWeek[[1]])
> spdf <- SpatialPointsDataFrame(coords = coords[,1:2],
+                 data = data.frame(coords[,4]), proj4string = crs)
> str(spdf)
```

```
Formal class 'SpatialPointsDataFrame' [package "sp"] with 5 slots
  ..@ data       :'data.frame':        1 obs. of  1 variable:
  .. ..$ coords...4.: Factor w/ 1 level "urn:ogc:object:feature:OSIRIS-HWS:3d3b239f-7696-4
  ..@ coords.nrs : num(0)
  ..@ coords     : num [1, 1:2] 51.94 7.61
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : NULL
  .. .. ..$ : chr [1:2] "lat" "lon"
  ..@ bbox       : num [1:2, 1:2] 51.94 7.61 51.94 7.61
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2] "lat" "lon"
  .. .. ..$ : chr [1:2] "min" "max"
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slots
  .. .. ..@ projargs: chr "+init=epsg:4326"
```

4.4 GetObservationById

The operation GetObservationById is defined in clause 10.1 of the SOS specification and not part of the core profile. But it is implemented as it is quite simple. The response is the same as described in the previous section. Optional parameters are the same as in GetObservation requests.

```
> obs <- getObservationById(sos = mySOS, observationId = "o00001")
> obs
```

5 Changing Handling Functions

TODO: explain approach, mention available non-exchangeable functions in the subsections

fixed order, exchangeable components

explain include/exclude mechanism

The process of data download also comprises (i) building requests, (ii) decoding responses, and (iii) applying the correct R data type to the respective data values. This mechanism is explained in detail in see section 5.3.

5.1 Parsers/Decoders

The terms parsing and decoding are used as names for the process of processing an XML document to create an R object.

TBD

5.2 Encoders

TBD

5.3 Data Converters

A list of named functions to be used by the parsing methods to convert data values to the correct R type, which are mostly based on the unit of measurement[13] code.

The conversion functions always take two parameters: `x` is the object to be converted, `sos` is the service where the request was received from.

The available functions are basically wrappers for coercion functions, for example `as.double()`. The only method exploiting the second argument is the one for conversion of time stamps which uses the time format saved with the object of class `SOS` in a call to `strptime`.

```
[1] 2
```

```
[1] "2"
```

```
[1] "character"
[1] "character"
```

```
[1] "2.0"
```

```
[1] 2
```

```
[1] "numeric"
[1] "numeric"
```

```
[1] "1"
```

```
[1] NA
```

```
[1] "logical"
[1] "logical"
```

```
[1] "2010-01-01T12:00:00.000"
```

```
[1] "2010-01-01 12:00:00 CET"
```

```
[1] "POSIXt"  "POSIXct"
[1] "POSIXt"  "POSIXct"
```

The full list of currently supported units can be seen below. It mostly contains common numerical units which are converted to type `double`.

```
 [1] "urn:ogc:data:time:iso8601"       "urn:ogc:property:time:iso8601"
 [3] "urn:ogc:phenomenon:time:iso8601" "time"
 [5] "m"                               "s"
 [7] "g"                               "rad"
 [9] "K"                               "C"
[11] "cd"                              "%"
[13] "ppth"                            "ppm"
[15] "ppb"                             "pptr"
[17] "mol"                             "sr"
```

[13] http://en.wikipedia.org/wiki/Units_of_measurement

```
[19] "Hz"                                "N"
[21] "Pa"                                "J"
[23] "W"                                 "A"
[25] "V"                                 "F"
[27] "Ohm"                               "S"
[29] "Wb"                                "Cel"
[31] "T"                                 "H"
[33] "lm"                                "lx"
[35] "Bq"                                "Gy"
[37] "Sv"                                "gon"
[39] "deg"                               "'"
[41] "''"                                "l"
[43] "L"                                 "ar"
[45] "t"                                 "bar"
[47] "u"                                 "eV"
[49] "AU"                                "pc"
[51] "degF"                              "hPa"
[53] "mm"                                "nm"
[55] "cm"                                "km"
[57] "m/s"                               "kg"
[59] "mg"                                "uom"
[61] "urn:ogc:data:feature"
```

The following connection shows a typical workflow of connecting to a new SOS for the first time, what the errors for missing converters look like, and how to add them to the SOS connection.

```
> # GET Verbindung
> MBARI <- SOS("http://mmisw.org/oostethys/sos",
+                 method = SosSupportedConnectionMethods()[["GET"]])

Created SOS for URL http://mmisw.org/oostethys/sos

> myOff <- sosOfferings(MBARI)[[1]]
> myProc <- sosProcedures(MBARI)[[1]]
> mbariObs1 <- try(getObservation(sos = MBARI, offering = myOff,
+                                 procedure = myProc))

Finished getObservation to http://mmisw.org/oostethys/sos
        --> received 1 observation(s) having 100 result values [ 100 ].

> warnings()

Warnmeldungen:
1: In sub(object$syntax$docexpr, val, chunk[pos[1L]]) :
  Argument 'replacement' hat einen LÃd'nge > 1 und nur das erste Element wird benutzt
2: In FUN(X[[7L]], ...) :
  swe:Quantity given without unit of measurement: Salinity
3: In .valParser(values = obj[[sweValuesName]], fields = .fields,  ... :
  No converter for the unit of measurement  S/m  with the definition  http://mmisw.org/ont
4: In .valParser(values = obj[[sweValuesName]], fields = .fields,  ... :
```

```
  No converter found! Skipping field  ConductivityNo converter found! Skipping field  http
5: In .valParser(values = obj[[sweValuesName]], fields = .fields,  ... :
  No converter found for the given field Salinity, http://mmisw.org/ont/cf/parameter/sea_w
6: In .valParser(values = obj[[sweValuesName]], fields = .fields,  ... :
  No converter found! Skipping field  SalinityNo converter found! Skipping field  http://m
7: In FUN(X[[7L]], ...) :
  swe:Quantity given without unit of measurement: Salinity
8: In .handleExceptionReport(sos, .response) :
  Object of class OwsExceptionReport; version: 1.0.0, lang: NA,  1 exceptions (code @ loca
        InvalidRequest @ NA : The request was sent in an unknown format or is invalid! Ple
```

There are warnings about unknown units of measurement. The example below creates conversion functions for these and subsequently results in more fields in the final result.

```
> # Create converters for missing units and definitions, then reconnect:
> myConverters <- SosDataFieldConvertingFunctions(
+                 "S/m" = sosConvertDouble,
+                 "http://mmisw.org/ont/cf/parameter/sea_water_salinity" = sosConvertDoubl
> MBARI <- SOS("http://mmisw.org/oostethys/sos",
+                method = SosSupportedConnectionMethods()[["GET"]],
+                dataFieldConverters = myConverters)

Created SOS for URL http://mmisw.org/oostethys/sos

> mbariObs2 <- getObservation(sos = MBARI, offering = myOff, procedure = myProc)

Finished getObservation to http://mmisw.org/oostethys/sos
        --> received 1 observation(s) having 100 result values [ 100 ].

> names(sosResult(mbariObs1))

[1] "esecs"        "Latitude"     "Longitude"    "NominalDepth" "Temperature"

> names(sosResult(mbariObs2))

[1] "esecs"        "Latitude"     "Longitude"    "NominalDepth" "Temperature"
[6] "Conductivity" "Salinity"
```

6 Exception Handling

When working with sos4R, two kinds of errors must be handled: service exceptions and package errors. The former can occur when a request is invalid or a service encounters internal exceptions. The latter can mean a bug or illegal settings within the package. To understand both types of errorenous states, this sections explains the contents of the exception reports returned by the service and the functionalities to investigate the inner workings of the package.

6.1 OWS Service Exceptions

The service exceptions returned by a SOS are described in OGC Web Services Common (Whiteside, 2007) clause 8. The classes to handle the returned exceptions in sos4R are `OwsExceptionReport`, which contains a list of exception reports, and `OwsException`, which contains slots for the parameters exception text(s), exception code, and locator. These are defined as follows and can be implementation specific.

ExceptionText Text describing specific exception represented by the exceptionCode.

exceptionCode Code representing type of this exception.

locator Indicator of location in the client's operation request where this exception was encountered.

The standard exception codes and meanings are accessible by calling

```
> OwsExceptionsData()
```

directly in **sos4R** and are shown in table 1.

	exceptionCode	meaningOfCode
1	OperationNotSupported	Request is for an operation that is not supported by this server
2	MissingParameterValue	Operation request does not include a parameter value, and this server did r
3	InvalidParameterValue	Operation request contains an invalid parameter value
4	VersionNegotiationFailed	List of versions in 'AcceptVersions' parameter value in GetCapabilities ope
5	InvalidUpdateSequence	Value of (optional) updateSequence parameter in GetCapabilities operation
6	OptionNotSupported	Request is for an option that is not supported by this server
7	NoApplicableCode	No other exceptionCode specified by this service and server applies to this

Table 1: Exception Data Table

```
> response <- try(getObservationById(sos = mySOS, observationId = "doesNotExist"))

Object of class OwsExceptionReport; version: 1.0.0, lang: NA,  1 exceptions (code @ locato
        InvalidRequest @ NA : The request was sent in an unknown format or is invalid! Ple
```

If an exception is received then it is also saved as a warning message.

6.2 Inspect Requests and Verbose Printing

The package offers two levels of inspection of the ongoing operations indicated by two boolean parameters, `inspect` and `verbose`. These are available in all service operation calls. The option `verboseOutput` when using the method `SOS(...)` turns on the verbose setting for all subsequent requests made to the created connection unless deactivated in an operation call.

inspect prints the raw requests and responses to the console. An example is shown below.

verbose prints not only the requests, but also debugging statements which are too extensive to show for this document.

```
> off1 <- sosOfferings(mySOS)[[1]]
> getObservation(sos = mySOS,
+                 offering = off1, latest = TRUE,
+                 procedure = sosProcedures(off1)[[1]],
+                 inspect = TRUE)

*** POST! REQUEST:
<sos:GetObservation xsi:schemaLocation="http://www.opengis.net/sos/1.0 http://schemas.open
 <sos:offering>RAIN_GAUGE</sos:offering>
 <sos:eventTime>
  <ogc:TM_Equals>
   <ogc:PropertyName>om:samplingTime</ogc:PropertyName>
   <gml:TimeInstant>
    <gml:timePosition>latest</gml:timePosition>
   </gml:TimeInstant>
  </ogc:TM_Equals>
 </sos:eventTime>
 <sos:procedure>urn:ogc:object:feature:OSIRIS-HWS:3d3b239f-7696-4864-9d07-15447eae2b93</so
 <sos:observedProperty>urn:ogc:def:property:OGC::Precipitation1Hour</sos:observedProperty>
 <sos:responseFormat>text/xml;subtype=&quot;om/1.0.0&quot;</sos:responseFormat>
</sos:GetObservation>
*** RESPONSE size: 3968 )
** RESPONSE DOC:
<?xml version="1.0" encoding="UTF-8"?>
<om:ObservationCollection xmlns:om="http://www.opengis.net/om/1.0" xmlns:gml="http://www.o
  <gml:boundedBy>
    <gml:Envelope srsName="urn:ogc:def:crs:EPSG:4326">
      <gml:lowerCorner>51.9412 7.6103</gml:lowerCorner>
      <gml:upperCorner>51.9412 7.6103</gml:upperCorner>
    </gml:Envelope>
  </gml:boundedBy>
  <om:member>
    <om:Observation gml:id="ot_3505678">
      <om:samplingTime>
        <gml:TimePeriod xsi:type="gml:TimePeriodType">
          <gml:beginPosition>2010-12-26T00:15:00.000+01:00</gml:beginPosition>
          <gml:endPosition>2010-12-26T00:15:00.000+01:00</gml:endPosition>
        </gml:TimePeriod>
      </om:samplingTime>
      <om:procedure xlink:href="urn:ogc:object:feature:OSIRIS-HWS:3d3b239f-7696-4864-9d07-
      <om:observedProperty>
        <swe:CompositePhenomenon gml:id="cpid0" dimension="1">
          <gml:name>resultComponents</gml:name>
          <swe:component xlink:href="urn:ogc:data:time:iso8601"/>
          <swe:component xlink:href="urn:ogc:def:property:OGC::Precipitation1Hour"/>
        </swe:CompositePhenomenon>
      </om:observedProperty>
```

```
      <om:featureOfInterest>
        <gml:FeatureCollection>
          <gml:featureMember>
            <sa:SamplingPoint gml:id="urn:ogc:object:feature:OSIRIS-HWS:3d3b239f-7696-4864
              <gml:name>weather @ roof of the ifgi, MS, Germany</gml:name>
              <sa:sampledFeature xlink:href="urn:ogc:def:nil:OGC:unknown"/>
              <sa:position>
                <gml:Point>
                  <gml:pos srsName="urn:ogc:def:crs:EPSG:4326">51.9412 7.6103</gml:pos>
                </gml:Point>
              </sa:position>
            </sa:SamplingPoint>
          </gml:featureMember>
        </gml:FeatureCollection>
      </om:featureOfInterest>
      <om:result>
        <swe:DataArray>
          <swe:elementCount>
            <swe:Count>
              <swe:value>1</swe:value>
            </swe:Count>
          </swe:elementCount>
          <swe:elementType name="Components">
            <swe:DataRecord>
              <swe:field name="Time">
                <swe:Time definition="urn:ogc:data:time:iso8601"/>
              </swe:field>
              <swe:field name="feature">
                <swe:Text definition="urn:ogc:data:feature"/>
              </swe:field>
              <swe:field name="urn:ogc:def:property:OGC::Precipitation1Hour">
                <swe:Quantity definition="urn:ogc:def:property:OGC::Precipitation1Hour">
                  <swe:uom code="mm"/>
                </swe:Quantity>
              </swe:field>
            </swe:DataRecord>
          </swe:elementType>
          <swe:encoding>
            <swe:TextBlock decimalSeparator="." tokenSeparator="," blockSeparator=";"/>
          </swe:encoding>
          <swe:values>2010-12-26T00:15:00.000+01:00,urn:ogc:object:feature:OSIRIS-HWS:3d3b
        </swe:DataArray>
      </om:result>
    </om:Observation>
  </om:member>
</om:ObservationCollection>

Finished getObservation to http://v-swe.uni-muenster.de:8080/WeatherSOS/sos
        --> received 1 observation(s) having 1 result values [ 1 ].
Object of class OmObservationCollection with 1 members.
```

7 Getting Started

The **demos** are a good way to get started with the package. Please be aware that the used SOSs might be temporarily unavailable.

```
> demo(package = "sos4R")
```

Additionally, there is a list of services on the project homepage (http://www.nordholmen.net/sos4r/data/) and a few SOS URLs are available via the function `SosExampleServices()`.

```
> SosExampleServices()

$`52 North SOS: Weather Data, station at IFGI, Muenster, Germany`
[1] "http://v-swe.uni-muenster.de:8080/WeatherSOS/sos"

$`52 North SOS: Water gauge data for Germany`
[1] "http://v-sos.uni-muenster.de:8080/PegelOnlineSOSv2/sos"

$`52 North SOS: Air Quality Data for Europe`
[1] "http://v-sos.uni-muenster.de:8080/AirQualityEurope/sos"

$`OOTethys SOS: Marine Metadata Interoperability Initiative (MMI)`
[1] "http://mmisw.org/oostethys/sos"

$`OOTethys SOS: Gulf of Maine Ocean Observing System SOS`
[1] "http://www.gomoos.org/cgi-bin/sos/oostethys_sos.cgi"
```

8 Getting Support

If you want to ask questions about using the software, please go first to the 52°North **forum** for the geostatistics community at http://geostatistics.forum.52north.org/ and check if a solution is described there. If you are a frequent user please consider subscribing to the geostatistics **mailing list** (http://list.52north.org/mailman/listinfo/geostatistics) which is linked to the forum.

9 Developing sos4R

Code Repository

You can download and browse the source of **sos4R** directly from the 52°North repository:

- **SVN resource URL**: https://svn.52north.org/svn/geostatistics/main/sos4R. Please read the documentation (especially the posting guide) of the 52°North repositories[14]. Anonymous access for download is possible.

[14] http://52north.org/resources/source-repositories/

- **Web access**: https://svn.52north.org/cgi-bin/viewvc.cgi/main/sos4R/?root=geostatistics

See the **developer documentation** at the 52 °North Wiki for detailed information on how to use the checked out source project: https://wiki.52north.org/bin/view/Geostatistics/Sos4R. You will find a detailed description of the folder and class structure, the file naming scheme, and an extensive list of tasks for future development.

Please get in touch with the community lead[15] of the geostatistics community if you want to **become a contributor**.

10 Acknowledgements

The project was generously supported by the 52 °North Student Innovation Prize for Geoinformatics 2010.

11 References

Botts, M., 2007, OGC Implementation Specification 07-000: OpenGIS Sensor Model Language (SensorML)- Open Geospatial Consortium, Tech. Rep.

Chambers, J.M., 2008, Software for Data Analysis, Programming with R. Springer, New York.

Cox, S., 2007, OGC Implementation Specification 07-022r1: Observations and Measurements - Part 1 - Observation schema. Open Geospatial Consortium. Tech. Rep.

Cox, S., 2007, OGC Implementation Specification 07-022r3: Observations and Measurements - Part 2 - Sampling Features. Open Geospatial Consortium. Tech. Rep.

Na, A., Priest, M., Niedzwiadek, H. and Davidson, J., 2007, OGC Implementation Specification 06-009r6: Sensor Observation Service, http://portal.opengeospatial.org/files/?artifact_id=26667, Open Geospatial Consortium, Tech. Rep.

Portele, C., 2003, OGC Implementation Specification 07-036: OpenGIS Geography Markup Language (GML) Encoding Standard, version: 3.00. Open Geospatial Consortium, Tech. Rep.

Vretanos, P.A., 2005, OGC Implementation Specification 04-095: OpenGIS Filter Encoding Implementation Specification. Open Geospatial Consortium, Tech. Rep.

Whiteside, A., Greenwood, J., 2008, OGC Implementation Specification 06-121r9: OGC Web Services Common Specification. Open Geospatial Consortium, Tech. Rep.

[15] http://52north.org/communities/geostatistics/community-contact