sos4R cheat sheet

Contact Daniel Nüst (daniel.nuest@uni-muenster.de)
Version 4 (2011-08-19) for sos4R Version 0.2
Website www.nordholmen.net/sos4r

Connecting to a SOS and Accessing Settings

```
> mySOS <- SOS(url = "http://mySOS:myPort/sos")  > sosUrl(mySOS)          > sosVersion(mySOS)      > sosTimeFormat(mySOS)
> sosMethod(mySOS)      > sosEncoders(mySOS)     > sosParsers(mySOS)      > sosDataFieldConverters(mySOS)
```

Explore Service Capabilities

```
> sosServiceIdentification(mySOS)        > sosServiceProvider(mySOS)       > sosOperationsMetadata(mySOS)
> sosFilter_Capabilities(mySOS)          > sosOfferingIds(mySOS)           > sosGetCRS(mySOS)
> sosResponseFormats(mySOS | myOffering) > sosResponseMode(mySOS | myOffering)     > sosResultModels(mySOS | myOffering)
> myOfferings <- sosOfferings(mySOS)     > sosOfferings(mySOS, name = "name")      > myOffering <- sosOfferings(mySOS)[["id"]]
> sosId(myOffering); sosTitle(mySOS); sosAbstract(mySOS); sosName(offerings | myOffering); plot(mySOS); summary(mySOS)
```

Explore Available Phenomena, Sensors, and Observed Properties based on Offering(s)

```
> sosBoundedBy(myOffering)       > sosBoundedBy(myOffering, bbox = TRUE)   # sp compatible          > sosGetCRS(myOffering)
> sosTime(myOffering)            > sosTime(myOffering, convert = TRUE)     # as POSIXt
> sosProcedures(mySOS | myOffering | myOfferings)                > sosObservedProperties(mySOS | myOffering | myOfferings)
> sosFeaturesOfInterest(mySOS | myOffering | myOfferings)        > plot(myOffering)                > summary(myOffering)
```

Request Observation Data

```
> myObs <- getObservation(sos = mySOS, offering = myOffering)             > getObservation(sos = mySOS, offering = "myId")
> myObs[3:4 | "procedureID" | "observedPropertyID" | "featuresOfInterestID"]       > myObs[[1]]     # subsetting methods
> getObservationById(sos = mySOS, observationId = "myObservationID")
```

Get Result Data

```
> myResult <- sosResult(myObs[[1]] | myObs[1:2])                                  > sosResult(myObs, coordinates = TRUE)
> attributes(myResult[["observedProperty"]])     # field's metadata (e.g. uom)    > sosFeatureIds(myObs | myObs[1:2])
> sosCoordinates(myObs | myObs[1:4])             > sosBoundedBy(myObs[4:5])       > sosUOM(myObs | myObs[[1]] | myResult)
> sosBoundedBy(myObservations[[1]], bbox = TRUE) # sp compatible                 > summary(myObs | myObs[[1]])
```

Subsetting in Requests: Temporal, Spatial, and Result Filtering

```
> lastWeekTP <- sosCreateTimePeriod(sos = mySOS, begin = (Sys.time() - 3600), end = Sys.time()) # using POSIXt classes
> lastDayTI <- sosCreateTimeInstant(sos = mySOS, time = as.POSIXct("2011-01-01"))
> myTime <- sosCreateEventTimeList(lastWeekTP | lastDayTI)                # must wrap time period/instant in event time list
> lastDay <- sosCreateEventTime(time = lastDayTI, operator = SosSupportedTemporalOperators()[[1]])
> year2000 <- sosCreateTime(sos = mySOS, time = "2007-07-07 07:00::2008-08-08 08:00" | "2007-01/2007-02" | "::2009" | "2010/")

> bb <- sosCreateBBOX(lowLat = 50.0, lowLon = 5.0, uppLat = 55.0, uppLon = 10.0, srsName = "urn:ogc:def:crs:EPSG:4326")
> myBBox <- sosCreateFeatureOfInterest(spatialOps = bb)                   # must wrap bounding box in feature element
> myFoi <- sosCreateFeatureOfInterest(objectIDs = list("foiId1", ...))    # request specific features of interest

> filter.pn <- xmlNode(name = "PropertyName", namespace = "ogc")          # namespace placeholder in following calls: '*'
> xmlValue(filter.pn) <- "urn:ogc:def:property:OGC::Temperature"                                  # property to filter
> filter.lit <- xmlNode(name = "Literal", *)                      > xmlValue(filter.l) <- "-2.3"   # filtering value
> filter.op <- xmlNode(name = "PropertyIsGreaterThan", .children = list(filter.pn, filter.lit), *)  # type of comparison
> myFilter <- xmlNode(name = "result", .children = list(filter.op), *)    # add property to a result element

> getObservation(sos = mySOS, offering = myOffering,              # offering (as id or offering object) is mandatory
      eventTime = myTime,                                         # temporal filtering
      procedure = sosProcedures(myOffering)[[1]],                 # specific procedure(s)
      observedProperty = sosObservedProperties(myOffering)        # specific phenomenon(s)
      featureOfInterest = myBBox | myFoi,                         # spatial filtering or specific feature(s)
      result = myFilter,                                          # result filtering
      saveOriginal = TRUE)                                        # saves a copy of the received document
> # other parameters: responseFormat, srsName, resultModel, responseMode, BBOX (for GET only!), latest (52N SOS only!)
```

Request Sensor Description (more detailed information only accessible if sensor description follow the SensorML Profile for Discovery)

```
> myProc = describeSensor(sos = mySOS, procedure = "myProcedureID")       > myProc@xml     # access original document
> sosId(myProc); sosName(myProcedure); sosAbstract(myProc); sosGetCRS(myProc); sosCoordinates(myProc); sosBoundedBy(myProc)
```

Result Coercion

```
> coord <- sosCoordinates(myObs); crs <- sosGetCRS(myObs)         > as(myObs, "SpatialPointsDataFrame") # shortcut
> spdf1 <- SpatialPointsDataFrame(coords = coord[,1:2], data = sosResult(myObs), proj4string = crs)
> spdf2 <- SpatialPointsDataFrame(coords = myResult[,c("lon", "lat")], data = myResult[,c("myVar")], proj4string = crs)
```

Exchange Parsing/Conversion/Encoding-Functions

```
> myERParser <- function(xml) { return("EXCEPTION!!") };  # parsing function named by XML element
> myEncoder <- function(object, sos, verbose) {<...>}     # encoding functions named with transfer method
> myConverters <- SosDataFieldConvertingFunctions("myUnit" = sosConvertDouble, "time" = sosConvertTime)
> # converters named by unit or observed property,
> mySOS2 <- SOS(sosUrl(mySOS, parsers = SosParsingFunctions("ExceptionReport" = myERParser),
      encoders = SosEncodingFunctions("POST" = myEncoder),  dataFieldConverters = myConverters)
```

Debugging (inspect and verbose can be set on **all** SOS operations)

```
> getObservation(sos = mySOS, …, inspect = TRUE)          > describeSensor(sos = mySOS, …, verbose = TRUE)
```

Default Functions, Supported Features, Demos, and Help

```
> SosSupported ~Operations() ~ConnectionMethods() ~ResponseFormats() ~ResultModels() ~TemporalOperators(), … # check features
> SosEncodingFunctions(); SosParsingFunctions(); SosDataFieldConvertingFunctions() # constructors for handling methods
> vignette("sos4R") # extensive documentation     > demo(package = "sos4R") # a list of available demos      > demo("demoname")
> sosChanges() # print the CHANGES to console     > sosCheatSheet() # open the cheat sheet (this document)   > SosDefaults()
```

Legend: | = alternatives of input parameters, please use just one. ~ = concatenate text for the method name.