SOS.js Documentation

Release 0.1

Paul Breen, Daniel Nüst

September 09, 2015

Contents

SOS.js is a JavaScript library to browse, visualise, and access, data from an Open Geospatial Consortium (OGC) Sensor Observation Service (SOS).

Contents

CHAPTER 1

Overview of the SOS.js Library

The library consists of a number of modules, which along with their dependencies build a layered abstraction for communicating with a SOS.

The core module - `SOS.js`, contains a number of objects that encapsulate core concepts of SOS, such as managing the service connection parameters, the service's capabilities document, methods to access the service's Features of Interest (FOIs), offerings, observed properties etc. It also contains various utility functions, available as methods of the `SOS.Utils` object. The objects of this module are:

- `SOS`
- `SOS.Offering`
- `SOS.Proxy`
- `SOS.Utils`

This module is built on top of OpenLayers, for low-level SOS request/response handling.

The user interface module - SOS.Ui.js, contains the UI components of the library. These components can be used standalone, but are also brought together in the default `SOS.App` object as a (somewhat) generic web application. The objects of this module are:

- `SOS.Plot`
- `SOS.Table`
- `SOS.Map`
- `SOS.Menu`
- `SOS.Info`
- `SOS.App`

This module is built on top of OpenLayers which provides simple mapping for discovery; jQuery for the UI and plumbing; and flot, which is a jQuery plugin, for the plotting.

In addition, there are a number of separate modules that contain UI extension components that are built on top of the above standard components.

- `SOS.MapSet.js`
- `SOS.Plot.Rose.js`
- `SOS.Plot.Stuve.js`

All the styling for the UI components is contained in the library style sheet - `SOS.Styles.css`.

CHAPTER 2

Usage

Here we discuss examples of using the various objects of the library. For fully working examples, see the examples directory in the library distribution.

2.1 SOS

The core SOS object can be used for low-level communication with a SOS. After instantiating a SOS object, the user then interacts with the object via a series of event handling callbacks.

To instantiate a SOS object, we pass it a number of options. Only the URL to the SOS is required, so at its simplest, this will suffice:

```
var options = {
  url: "http://sosmet.nerc-bas.ac.uk:8080/sosmet/sos"
};

var sos = new SOS(options);
```

Typically the first thing that is required after instantiation, is to fetch the capabilities document of the SOS. As this call is asynchronous, we need to setup a callback to handle the `sosCapsAvailable` event, which signifies that the SOS object has received and parsed the capabilities document from the given SOS. We can accomplish this via the following:

```
sos.registerUserCallback({
  event: "sosCapsAvailable",
  scope: this,
  callback: capsHandler
});

sos.getCapabilities();

function capsHandler(evt) {
...
```

whereupon our `capsHandler` function can then inspect the capabilities of the SOS via the available method calls of the SOS object [1]. As a convenience, we can pass the name of our callback function as an argument to the `getCapabilities` call, which will then register this callback function to handle the `sosCapsAvailable` event with a scope of `this`; so identical to the above explicit `registerUserCallback` call:

[1] The parsed capabilities document is stored as a JSON object in the SOS object as `this.SOSCapabilities`. The structure of this document may change in future versions of the library, so direct access is discouraged.

```
sos.getCapabilities(capsHandler);
```

To unregister a callback, we can issue the following:

```
sos.unregisterUserCallback({
  event: "sosCapsAvailable",
  scope: this,
  callback: capsHandler
});
```

Once we have our capabilities document, we can inspect the available offerings and FOIs of the given SOS:

```
var offIds = sos.getOfferingIds();
var offNames = sos.getOfferingNames();
var foiIds = sos.getFeatureOfInterestIds();
```

2.2 SOS.Offering

Once we've identified an offering we're interested in, we can fetch a SOS.Offering object that encapsulates that offering:

```
var offering = sos.getOffering(offId);
var offering = sos.getOfferingByName(offName);
```

or we can fetch an array of SOS.Offering objects pertaining to a given FOI or procedure:

```
var offerings = sos.getOfferingsForFeatureOfInterestId(foiId);
var offerings = sos.getOfferingsForProcedureId(procId);
```

We can inspect the details of a particular offering, via its method calls:

```
var foiIds = offering.getFeatureOfInterestIds();
var procIds = offering.getProcedureIds();
var propIds = offering.getObservedPropertyIds();
var propNames = offering.getObservedPropertyNames();
```

and furthermore we can fetch observations of the offering's observed properties. By default, observations for all the offering's observed properties will be retrieved, however, often we may only want observations for a particular observed property, or subset of observed properties. This can be achieved by filtering the offering's observed properties, thus:

```
// Fetch the air temperature only
offering.filterObservedProperties("air_temperature");

// Fetch the wind data only
offering.filterObservedProperties(["wind_speed", "wind_direction"]);
```

To reset an offering's observed properties list, we unfilter:

```
offering.unfilterObservedProperties();
```

Similarly, we may only be interested in observations from a single FOI from an offering (for example, a single station's observations from a multi-station offering). To achieve this, we simply set the `foiId` property of the SOS.Offering object, thus:

```
offering.foiId = "foi_34579";
```

To retrieve observations for all an offering's FOIs, we simply reset this property:

```
offering.foiId = null;
```

Once we have specified the desired observed property(s) and FOIs, we can fetch observation records, given a date range [2]. This is an asynchronous call, so just like the capabilities call above, we can explicitly setup a callback event handler:

```
offering.registerUserCallback({
  event: "sosObsAvailable",
  scope: this,
  callback: obsHandler
});

offering.getObservations(startDatetime, endDatetime);

function obsHandler(evt) {
...
```

or alternatively, we can use the convenience of passing our callback function as an argument to the `getObservations` call:

```
offering.getObservations(startDatetime, endDatetime, obsHandler);
```

In our observation handler, we can then iterate over the observation records that were returned by the SOS, using the `getCountOfObservations` and `getObservationRecord` method calls. For example, to display the data in an HTML table, we could do something like:

```
for(var i = 0, len = offering.getCountOfObservations(); i < len; i++) {
  var ob = offering.getObservationRecord(i);
  tbody += '<tr>';
  tbody += '<td>' + ob.observedPropertyTitle + '</td>';
  tbody += '<td>' + ob.time + '</td>';
  tbody += '<td>' + ob.result.value + ' ' + ob.uomTitle + '</td>';
  tbody += '</tr>';
}
```

Note that if the requested observations are not pre-filtered by specifying observed property and FOI (see above), then we will probably want to post-filter them. This can be achieved by calling `getFilteredObservationRecord`, along with a suitable filter, instead of calling `getObservationRecord`. For example, if we have requested observations for all FOIs of a given multi-FOI offering, and wish to have a separate table for each FOI, then we could do something like:

```
var foiIds = this.getFeatureOfInterestIds();

for(var i = 0, flen = foiIds.length; i < flen; i++) {
  var filter = {foiId: foiIds[i]};

  for(var j = 0, olen = this.getCountOfObservations(); j < olen; j++) {
    var ob = this.getFilteredObservationRecord(j, filter);

    if(ob) {
      if(tcaption.length < 1) {
        var foi = this.getFeatureOfInterestFromObservationRecord(ob);

        if(foi) {
          tcaption = foi.attributes.name;
        }
```

[2] All dates and times passed to the library must be in an ISO 8601 compliant format. For example, for the 31st of August 2013, that would be `2013-08-31` or `2013-08-31T00:00:00.000Z` etc.

```
      }
      tbody += '<tr>';
      tbody += '<td>' + ob.observedPropertyTitle + '</td>';
      tbody += '<td>' + ob.time + '</td>';
      tbody += '<td>' + ob.result.value + ' ' + ob.uomTitle + '</td>';
      tbody += '</tr>';
    }
  }
}
```

The observation record that is returned by a call to `getObservationRecord` (or `getFilteredObservationRecord`) is an Observations and Measurements om:Measurement resultModel representation, as returned by SOS, with additional convenience members of `time`, `observedPropertyTitle` and `uomTitle`. It has the following structure:

```
{
  samplingTime: {
    timeInstant: {
      timePosition: "2013-08-25T00:00:00.000Z"
    }
  },
  procedure: "urn:ogc:object:feature:Sensor:BAS:bas-met-halley-met",
  observedProperty: "urn:ogc:def:phenomenon:OGC:1.0.30:air_temperature",
  fois: [{
    features: [{
      layer: null,
      lonlat: null,
      data: {
        id: "foi_34579",
        name: "Halley"
      },
      id: "OpenLayers.Feature.Vector_1570",
      geometry: {
        id: "OpenLayers.Geometry.Point_1569",
        x: -26.7,
        y: -75.58
      },
      state: null,
      attributes: {
        id: "foi_34579",
        name: "Halley"
      },
      style: null
    }]
  }],
  result: {
    value: "-40.3",
    uom: "Cel"
  },
  time: "2013-08-25T00:00:00.000Z",
  observedPropertyTitle: "Air Temperature",
  uomTitle: "&deg;C"
}
```

CHAPTER 3

Proxy Configuration

By default, the SOS.js library is configured to use a proxy for communicating with a backend SOS instance. This is to overcome CORS restrictions that web browsers place on javascript code. If you wish to use the SOS.js library without a proxy, then your backend SOS instance must be setup for CORS [1]. Otherwise, you must run a proxy on the web server that hosts SOS.js to marshall all requests from the SOS.js client library to your SOS backend. This is so that to the client (web browser), it looks as though the data from the SOS are a local resource.

If your SOS instance is setup for CORS, then add the following line somewhere near the start of your javascript (for example, in the page's `init` function, before instantiating any other SOS.js objects):

```
SOS.Proxy.disable();
```

This will then work without the need for a proxy.

If your SOS instance isn't setup for CORS, then install the `proxy.cgi` script (distributed with OpenLayers) into a path under your web server that is capable of running CGI binaries. (This web server is where you intend to host SOS.js.) For example, the default path under Apache is /path/to/web-server-root/cgi-bin, which resolves to a URL of http://your.web-server/cgi-bin/proxy.cgi. Edit the `proxy.cgi` script, and add the hostname and port of your SOS instance (and any others you wish to query) to the `allowedHosts` array. For example:

```
allowedHosts = [mysoshost.mydomain:8080, localhost:8080]
```

The SOS.js library should now successfully communicate with your backend SOS instance. (To check, see that the examples pages work.) If you install the `proxy.cgi` script to a different path (it must still be capable of running CGI binaries), then you'll need to tell the `SOS.Proxy` object where to find it. You can do this by adding the following line somewhere near the start of your JavaScript (again, for example, in a page's `init` function):

```
SOS.Proxy.enable({url: "/alternative/path/to/cgi-bin/proxy.cgi?url="});
```

[1] For example, if you have installed the 52°North SOS backend, then you need to configure Tomcat for CORS. Depending on the used version this configuration can be included.

CHAPTER 4

Step-by-step guide for 52°North SOS and SOS.js

This documentation is contributed by https://github.com/simonabadoiu.

This is a short tutorial on how you can make the SOS.js project run with the 52°North SOS.

4.1 52°North SOS Server Installation

In order to use the sos-js library and examples with the 52°North SOS, you need to deploy the SOS on a local server - Apache Tomcat for example. The SOS installation guide which describes two options for deploying the SOS on a local server:

- Build the SOS webapp and deploy it on a local server
- Download the war file and deploy it on a local server

For SOS.js you will need to package the war file using the develop profile (`mvn package -P develop`) so that the SOS 1.0.0 binding is included. Therefore you must build the application and cannot simply download the war file.

4.2 SOS.js Deployment

Now that you have the SOS running on localhost, you can start deploying the sos-js application. Here are the steps that will lead you to an working sos-js application:

- Read the SOS.js readme file on github
- Clone the SOS.js github repository
- Download an HTTP server - for example Apache HTTP Server
- Copy the SOS.js project on the server - for the Apache server, you have to copy the files in the `www` folder
- The latest version of the 52°North SOS is set up for CORS, so you can simply disable the proxy behaviour. In order to do this, you have to call the `SOS.Proxy.disable()` function, see Proxy Configuration.
- Modify the test html documents from the examples folder, in order to make this work with your locally deployed SOS.
  - In almost each html test file, an URL is set for the SOS client. You have to change this URL with one that points to your local SOS.
  - Your local URL could look like this: http://localhost:8080/52n-sos-webapp/sos/kvp

After correctly passing through all this steps, your SOS.js application will look similar with the *SOS.js demo*.

CHAPTER 5

Contribute to the Project

This is the contributor documentation for the 52°North project SOS.js. Great to have you here! Here are a few ways you can help!

5.1 Learn & listen

Best way to get started is to start is getting to know what is going on and how 52°North works:

- Mailing list: http://52north.org/resources/mailing-list-and-forums/
- IRC channel: #52north on irc.freenode.net
- Website: http://52north.org/
- Get involved in 52°North: http://52north.org/about/get-involved/

5.2 Team Members

- @paul-breen
- @nuest
- @justb4

This list might not always be up to date, so also check the repository contributor list.

5.3 Contributor License Agreement

52°North requires all contributors to sign a contributors license agreement (CLA). This is not scary, but essential for open source projects to live beyond lifetime of funding projects and individual contributions.

Find all information on our licensing information page at and our CLA FAQ.

5.4 Adding new features

This section includes advice on how to build new features for the project and what kind of process this comprises.

- Start a discussion on the mailing list about your idea.

- Create an issue to track your work.
- Fork the repository.
- Create a new branch for the feature, e.g. `feature-mynewidea`

Don't get discouraged! We estimate that the response time from the maintainers is around a week.

5.5 Bug triage

Thanks for helping us to improve the software. Here's what you can do:

- You can help report bugs by filing them in the issue tracker: https://github.com/52North/sos-js/issues/new
- You can look through the existing bugs here: https://github.com/52North/sos-js/issues - If you find a bug interesting, help us understand it:
  - Is the bug is reproducible?
  - **Is it reproducible in other environments (browsers)?**
    - What are the steps to reproduce?

5.6 Translations

SOS.js currently does not support different languages. Are you interested to change this? Get in touch!

5.7 Documentation

Code needs explanation, and sometimes those who know the code well have trouble explaining it to someone just getting into it. That is why we need your help. You can directly contribute to the documentation by forking this project and editing the documentation files.

CHAPTER 6

Development Documentation

For general information how to contribute to the project, please see *the contribution page*.

6.1 Testing pages

For quick and simple testing of developments you can use the test clients in the `../examples` folder. Open the file `index-dev.html` to see a list of available pages. The pages load the original source files from a relative path and hence always use the latest version.

6.2 Building the documentation

This projects uses Sphinx to write and build the documentation, all details at http://sphinx-doc.org.

To build documentation install Sphinx and then run `make html` in the root directory `/`.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

CHAPTER 8

LICENSE

This project is licensed under the Apache Software License, Version 2.0. For details see the LICENSE file.

```
Copyright 2014 52°North Initiative for Geospatial Open Source Software GmbH

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

      http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```