## OGC Web Service .NET (v1.0) - Installation

First, make sure you have [Microsoft SQL Server System CLR Types](http://www.microsoft.com/downloads/details.aspx?FamilyId=C6C3E9EF-BA29-4A43-8D69-A2BED18FE73C&displaylang=en) installed. Next, copy the contents of [OGC WebService 1.0 Binary](http://ogc.codeplex.com/Release/ProjectReleases.aspx?ReleaseId=24956) into the web application folder. Then configure the connection string. Note that the application uses a data entity connection string, which should look like this:

<add name="SosEntities" connectionString="metadata=res://\*/Sos.Sos.csdl|res://\*/Sos.Sos.ssdl|res://\*/Sos.Sos.msl;provider=System.Data.SqlClient;provider connection string=&quot;Data Source=localhost;Initial Catalog=Odm11;User ID=myuser;Pwd=mypassword;Application Name=SOS;MultipleActiveResultSets=True&quot;" providerName="System.Data.EntityClient" />

The implementation shown here is designed to work “out of the box” with the [ODM 1.1](http://his.cuahsi.org/odmdatabases.html) database. The rest of this document is based on the assumption that you’ve downloaded and configured a sample database to work with.

## Configuration

There are several default settings that can be set at service level and will look like this:

<ServiceConfiguration defaultService="SOS" defaultVersion="1.0.0" defaultConnection="SosEntities" >

Following table describes different service default configuration options

|  |  |
| --- | --- |
| Name | Description |
| defaultService | Default service name to use when it's omitted in URL |
| defaultVersion | Default version number to use when it's omitted in URL |
| defaultConnection | Default connection to use for all service operations |

All services that are exposed by the application are configured in the *ServiceConfiguration* section of the *web.config* file. For example:

<add name="SOS\_GetCapabilities" service="SOS" operation="GetCapabilities" connection="SosEntities" cacheTimeout="60" type="Renci.Ogc.WebService.Sos.GetCapabilitiesOperation, Renci.Ogc.WebService, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />

<add name="SOS\_DescribeSensor" service="SOS" version="1.0.0" operation="DescribeSensor" connection="SosEntities" type="Renci.Ogc.WebService.Sos.DescribeSensorOperation, Renci.Ogc.WebService, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />

<add name="SOS\_GetObservation" service="SOS" version="1.0.0" operation="GetObservation" connection="SosEntities" cacheEnabled="false" type="Renci.Ogc.WebService.Sos.GetObservationOperation, Renci.Ogc.WebService, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />

Following table describes different configuration options

|  |  |  |
| --- | --- | --- |
| Name | Description | Usage |
| name |  | required |
| service | OWS service name | required |
| version | Service version, required for all services except GetCapabilities | optional |
| operation | Opreation/request name for the service | required |
| connection | Connection name, it is possible to configure multiple services to work with different databases | required |
| cacheEnabled | When enabled it caches operation results and will use cached result for the same input parameters | optional, default true |
| cacheTimeout | How long to store the caches for after last request | optional, default 20 minutes |
| type | What type should be used to process this operation request. Type should inherit from BaseOperation class. | required |

Each operation can provide its own default values by specifying defaultValues section. Default values in this section are operation specific but all of them configured in the sam way. For example:

<add name="SOS\_DescribeSensor" service="SOS" version="1.0.0" operation="DescribeSensor" cacheEnabled="false" type="Renci.Ogc.WebService.Sos.DescribeSensorOperation, Renci.Ogc.WebService, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null">

<defaultValues>

<clear />

<add name="OutputFormat" defaultValue="text/xml;subtype=&quot;sensorML/1.0.0&quot;"/>

</defaultValues>

</add>

Service can have zero or more managers to manage different aspects of the service and allows to control specific service behavior.

### Managers configuration

Managers configured inside managers node. For example:

<managers>

<clear />

<add name="UrnManager" type="Renci.Ogc.WebService.Sos.DefaultUrnManager"/>

<add name="UriManager" type="Renci.Ogc.WebService.Sos.DefaultUriManager"/>

</managers>

|  |  |
| --- | --- |
| Name | Description |
| UrnManager | Responsible for create URN names for properties, sensor and feature of interests that used by the service |
| UriManager | Responsible for creating URI links to property, site of feature of interest usually referenced by URN name |

If manager configuration is omitted the default behavior is assumed.

## Entity Data Model Configuration

The included entity data model supports access to the ODM 1.1 database. The application is designed in such a way that if a new entity data model is available, it can be configured to use the new model for the service. All you’ll need to do is re-configure the connection string to use the new model.

A new model can be created by replacing existing queries in the SSDL section of the *Sos.edmx* file. Once all queries are replaced, the application should work as usual. Please note that some of the existing queries contain NULL. This was intentionally done as a way to extend future applications, and to add support for features that are not currently available in the ODM 1.1 database model

## Source Code

The provided source code contains two major projects. First, a class library which is used to serialize and de-serialize service requests. Some classes were extended to simplify current development. We expect that those classes will be extended and improved upon in the future.

The second project is the web service. This has some base functionality that the SOS service is built on.

## Extensibility

At present you can extend this code in two ways.

One, you can inherit from the *BaseOperation* class and implement all functionality from scratch. An important method here is *ProcessRequest*, which is responsible for processing specific service requests. This approach can be taken if you want to implement services other than SOS.

Another way is to extend SOS services by inheritance from one of the operation classes defined in the *Renci.Ogc.WebService.Sos* namespace and then overrides existing functionality. This eliminates the need to implement the whole service by allowing modification to the specific part of the functionality that needs to be replaced or changed.

Finally, after a new class is created it needs to be configured in the *web.config* so the service can use this class. The “type” node of the service must then point to the newly created class (see example below).

<add name="SOS\_GetObservation" service="SOS" version="1.0.0" operation="GetObservation" connection="SosEntities" cacheEnabled="false" **type**="Renci.Ogc.WebService.Sos.GetObservationOperation, Renci.Ogc.WebService, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />

This is a working document and will be modified over time. Please direct any questions regarding this project to [team@sensordatabus.org](mailto:team@sensordatabus.org).