



deegree owsWatch v2.5

lat/lon GmbH

Aennchenstr. 19
53177 Bonn
Germany
Tel ++49 - 228 - 184 96-0
Fax ++49 - 228 - 184 96-29
info@lat-lon.de
www.lat-lon.de

Dept. of Geography
Bonn University
Meckenheimer Allee 166
53115 Bonn

Tel. ++49 228 732098

Change log

Date	Description	Author
2008-11-06	Document created	Moataz Elmasry
2008-11-21	Remove typos	Judit Mays
2009-10-21	Update for upcoming release	Judit Mays

Table of Contents

1 Introduction.....	4
2 Download/ Installation.....	5
2.1 Prerequisites.....	5
3 Configuration.....	6
3.1 Servlet Configuration.....	6
3.2 Tomcat Configurations.....	6
3.3 owsWatch configuration.....	7
3.3.1 ws:GeneralConfiguration.....	7
3.3.2 ws:ServiceConfiguration.....	8
3.4 owsWatch services.....	8
3.5 Resetting owsWatch.....	9
4 Using owsWatch.....	10
4.1 Monitoring the services.....	10
4.2 Adding/ editing service monitors.....	11
5 Adding and modifying test types.....	13
5.1 Adding a new test type.....	13
5.2 Adding/editing a test type. The client side of the story.....	13
Appendix A Configuration file for owsWatch.....	16
Appendix B Created Tests for the different services....	19
Appendix C Example for a web.xml file for owsWatch. .	22

Index of Tables

Illustration Index

Figure 1: owsWatch main window.....	10
Figure 2: Service Manager window to add/edit tests.....	11

1 Introduction

deegree is a Java Framework offering the main building blocks for Spatial Data Infrastructure (SDIs). Its entire architecture is developed using standards of the Open Geospatial Consortium (OGC) and ISO Technical Committee 211 - Geographic information / Geoinformatics (ISO/TC 211). deegree encompasses OGC Web Services as well as clients. deegree is Free Software protected by the GNU Lesser General Public License (GNU LGPL) and is accessible at www.deegree.org

owsWatch is a web-based application used to monitor the different services of deegree like WMS, WFS, CSW, etc...

owsWatch can be mainly considered an alarm that notifies the user per email or also through its interface in case any of the monitored services is experiencing high traffic that caused response latency or the service suffered an altogether failure. A monitored service means that the user configures a certain request against that service (e.x. GetCapabilities from a WMS) with constraints on that test (e.x. Timeout) to test if the service is live at all times.

Besides owsWatch, deegree comprises a number of additional services and clients. A complete list of deegree components can be found using the following URL and links:

<http://www.lat-lon.de> → Products

Downloads of packaged deegree components can be found via:

<http://www.deegree.org> → Download

2 Download/ Installation

2.1 Prerequisites

To run deegree2 owsWatch you need:

- Java (JRE or JDK) version 1.5.x
- Tomcat 5.5.x

For installation of these components refer to the corresponding documentation at java.sun.com and tomcat.apache.org respectively.

owsWatch is a web application which can be checked out from the SVN¹ under <http://wald.intevation.org/projects/deegree>. The project can then be found in [apps/owswatch/](http://wald.intevation.org/projects/deegree/apps/owswatch/)

¹ SVN is a subversion Framework (revision control system), that allows mutiple programmers to work on the same project and the same code and synchronize their work together without running into code conflict. If you want to know more about SNVs try using the svn tool from <http://subclipse.tigris.org/>

3 Configuration

A servlet is an object that is hosted by a servlet container (like tomcat). It receives requests through HTTP protocol (e.g. send by a browser) and creates a proper response according to that request; just like any computer program that receives an input and produces an output. The main purpose of servlets as objects is that they are mainly designed to be accessed remotely.

3.1 Servlet Configuration

To use the servlet we will need to create a new web application. Create a folder with any name, this will act as the main project folder. In this folder create another folder called "WEB-INF", which will hold the servlet information and inside it shall exist the xml file "web.xml" (deployment descriptor) and the "lib" folder. The lib folder contains all the needed libraries to run the servlet. In the "web.xml" create a `<servlet>` tag with the servlet class `org.deegree.enterprise.servlet.ChartServlet`. Also add two `init-param` elements. The first is "errorMsg" which shall hold the path of an error image, in case the chart could not be created. The second parameter is "configFile" which is the input xml configuration of the servlet. With it you can control how the output chart should look. The configuration file structure will be explained in section 3.2. Please consult Appendix A for a web.xml example that you can use directly in your web application (do not forget to update the paths of your `init-param` elements according to your configuration). Appendix C contains an example web.xml for owsWatch.

3.2 Tomcat Configurations

Apache Tomcat is a web container, or application server developed at the Apache Software Foundation (ASF). Tomcat implements the Java Servlet and the JavaServer Pages (JSP) specifications from Sun Microsystems, providing an environment for Java code to run in cooperation with a web server. It adds tools for configuration and management but can also be configured by editing configuration files that are normally XML-formatted. Tomcat includes its own internal HTTP server.

After Downloading and unpacking tomcat, please read the tomcat documentation on the apache website and install tomcat properly. After installing tomcat you need to create our context in tomcat in order to run our servlet, you need to do the following.

1. In "tomcat/conf" folder you need to create a folder with the path `./Catalina/localhost`
2. under "tomcat/conf/Catalina/localhost" create an xml file called "owswatch.xml", which will act as the web context.

3. The content of this xml file is as follows:

```
<Context displayName="owsWatch" docBase="my_webapp_path" ></Context>
```

where the value of the attribute docBase is the absolute path to your web application folder as explained in section 3.1

4. To start tomcat use your command line tool and run the batch file “./bin/startup.bat” in tomcat directory (Windows), or bin/catalina.sh (Linux)

By this we are done with necessary tomcat configuration. You might want to provide more memory resources than the default of 64M to your Tomcat. Please refer to tomcat documentation or the deegree wiki at <http://wiki.deegree.org/deegreeWiki/HowToDealWithOutOfMemoryError>.

3.3 owsWatch configuration

Basically owsWatch is already preconfigured, so a common user or someone who is starting to use it probably will not need to change the configurations. If you feel that you fall in this category you can just skip this section and jump to Chapter 4 Using owsWatch

In the owsWatch home folder exists the configuration file for the application under WEB-INF/conf/owswatch/owswatch_configuration.xml. We will explain the usage of the different elements in this configuration file

3.3.1 ws:GeneralConfiguration

The ws:GeneralConfiguration element contains general characteristics to the usage of the owsWatch interface, which are:

- **GlobalRefreshRate** can be used to set how often the application shall be refreshed. The time unit used is minute
- **Users** includes block(s) of User which sets access to the different users and their roles (roles is not yet implemented, so all users have the same privileges at the moment)
- **Mail** contains the the Email info like the sender email and the email server address. These information will be used to send the error emails(i.e. They represent the sender) if a service stopped working or produced an error.
- **Location** contains few locations which are used by the application. These locations are:
 - 1) ProtocolLocation: This should be a folder path, where the protocol itself is somehow similar to the log, in which the test results are written to. Note that each Test has its own protocol file and that a new protocol file is created at the beginning of each month without

the erasing the old protocol files, so according to the user needs, he/she might want to regularly cleanup this folder

- 2) ServiceInstanceLocation: This is the path to the xml file that contains the service configurations. This file is actually best edited through the owsWatch and not manually, but it is still a possibility. It will be explained later in this documentation how to edit the file through a GUI and per hand.

3.3.2 ws:ServiceConfiguration

These configurations affect the tests that can be added to monitor the services like the WMS, WFS, etc...

- **TestInterval** contains **Value** elements which is displayed as a drop down box. The time unit is minutes and it indicates how often a certain test should be executed. The drop down box is found in the service configuration (add/edit) dialogue of owsWatch
- **Services** contain several **Service** elements. A Service in this context is to be considered an OGC service and the element contains a description of this service. A service element contains the following attributes:

1- **type**: the OGC service type. The service name could be prefixed with OGC: but it is not a must

2- **version**: of the service

For the moment we will skip explaining the rest of the Service element and we will revisit it, when we discuss adding other OGC services to owsWatch since this part will require some Java and Javascript programming and so it will be proper to explain it at that place. Please consider reading chapter 5: Adding other OGC services

3.4 owsWatch services

A Service in this context means a test of a running OGC service. As we mentioned before, this file could be easily edited through the web application, but for as a reference we shall explain the xml file structure.

The file is to be found under WEB-INF/conf/owswatch/services.xml.

A very important attribute found in the root element is **service_id_sequence**. This attribute is used to assign a unique id to every newly created service. The service ID identifies the protocol files.

The **watch:Config** root element contains multiple ServiceMonitor elements. Each service monitor is responsible for testing a certain OGC service with a

certain test (e.x. GetCapabilities or GetMap). That basically means that multiple tests (i.e. multiple requests in our case) could run against the designated service. This is actually very practical since sometimes there is a problem in the configurations of a service, so that a GetCapabilities request would return a valid result, while another request like a WMS GetMap or CSW GetRecords wouldn't. Each service contains an ID attribute, which identifies it uniquely from other services.

A Service contains the following elements:

- **ACTIVE** says whether the the service should be tested at all or not
- **TIMEOUT** in seconds is the maximum time for running a request. After that a connection time out error will be produced
- **INTERVAL** indicates how often the test shall be executed
- **ONLINERESOURCE** The service of the service to send the request to
- **SERVICENAME** is the title of the test
- **HTTPMETHOD** contains the attribute type which indicates whether this is a GET or POST HTTP request. It also contains the following Elements
 - **REQUEST** name of the request like GetCapabilities and GetMap (Note: the name should be that of an OGC service)
 - **VERSION** of that service
 - **SERVICE** type like WMS; WCS, etc..

3.5 Resetting owsWatch

If you would like to remove everything in owsWatch and go to a start point, follow the following steps

1. In services.xml file, delete all elements under the root element and set the id_sequence attribute to 0.
2. in the protocols folder delete all files except protocol2html.xsl

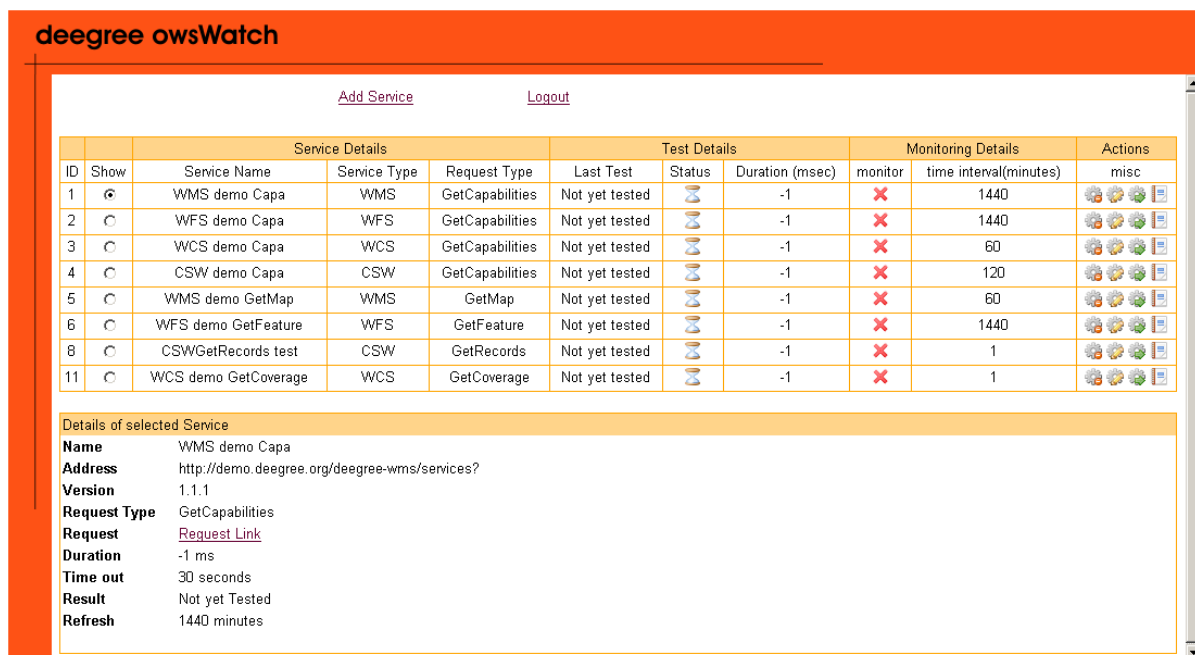
4 Using owsWatch

4.1 Monitoring the services

Start owsWatch by typing its address in your browser address bar. You should have specified the context name of owsWatch in tomcat. As an example we will assume that you named the application owsWatch and you are running tomcat on your local machine on port 8080, so the owsWatch address should be as follows: <http://localhost:8080/owswatch>

When starting the application you will get the login page, type in your user name and password as you specified them in the owsWatch configuration file in Chapter 3.

After logging in you should get an application similar to the one in the next picture.



The screenshot shows the 'deegree owsWatch' interface. At the top, there are links for 'Add Service' and 'Logout'. Below is a table with columns for Service Details, Test Details, Monitoring Details, and Actions. The table lists 11 services, all with a status of 'Not yet tested' and a duration of '-1'. The monitoring details for each service are shown as 'X' in a red box, indicating they are not being monitored. Below the table, there is a section for 'Details of selected Service' for the first service, 'WMS demo Capa'.

Service Details				Test Details			Monitoring Details		Actions	
ID	Show	Service Name	Service Type	Request Type	Last Test	Status	Duration (msec)	monitor	time interval(minutes)	misc
1	<input checked="" type="radio"/>	WMS demo Capa	WMS	GetCapabilities	Not yet tested		-1		1440	
2	<input type="radio"/>	WFS demo Capa	WFS	GetCapabilities	Not yet tested		-1		1440	
3	<input type="radio"/>	WCS demo Capa	WCS	GetCapabilities	Not yet tested		-1		60	
4	<input type="radio"/>	CSW demo Capa	CSW	GetCapabilities	Not yet tested		-1		120	
5	<input type="radio"/>	WMS demo GetMap	WMS	GetMap	Not yet tested		-1		60	
6	<input type="radio"/>	WFS demo GetFeature	WFS	GetFeature	Not yet tested		-1		1440	
8	<input type="radio"/>	CSWGetRecords test	CSW	GetRecords	Not yet tested		-1		1	
11	<input type="radio"/>	WCS demo GetCoverage	WCS	GetCoverage	Not yet tested		-1		1	

Details of selected Service

Name WMS demo Capa
Address <http://demo.deegree.org/deegree-wms/services?>
Version 1.1.1
Request Type GetCapabilities
Request [Request Link](#)
Duration -1 ms
Time out 30 seconds
Result Not yet Tested
Refresh 1440 minutes





Figure 1: owsWatch main window

By clicking on the radio buttons on the left hand side of the page the details on the bottom half of the page to display the information about the clicked service monitor.

If the HTTP request type is a GET request, you will find a link in the details named Request Link, through which you get the exact address used to test the service;

And if the HTTP request is of type post, you will have a text box with a the xml request, which is sent to the service through the POST request.

Next to each service on the right hand side, you should find the following buttons:

- 1- Delete button  used to delete the service altogether
- 2- Edit button  to edit the service details
- 3- Execute button  to run the test
- 4- Protocol button  to show the protocol (test results) of the previous tests of a service

4.2 Adding/ editing service monitors

On the top of the page you will find an Add Service button, click on it to get a new dialogue (The Service Manager) with the details of the new monitor.

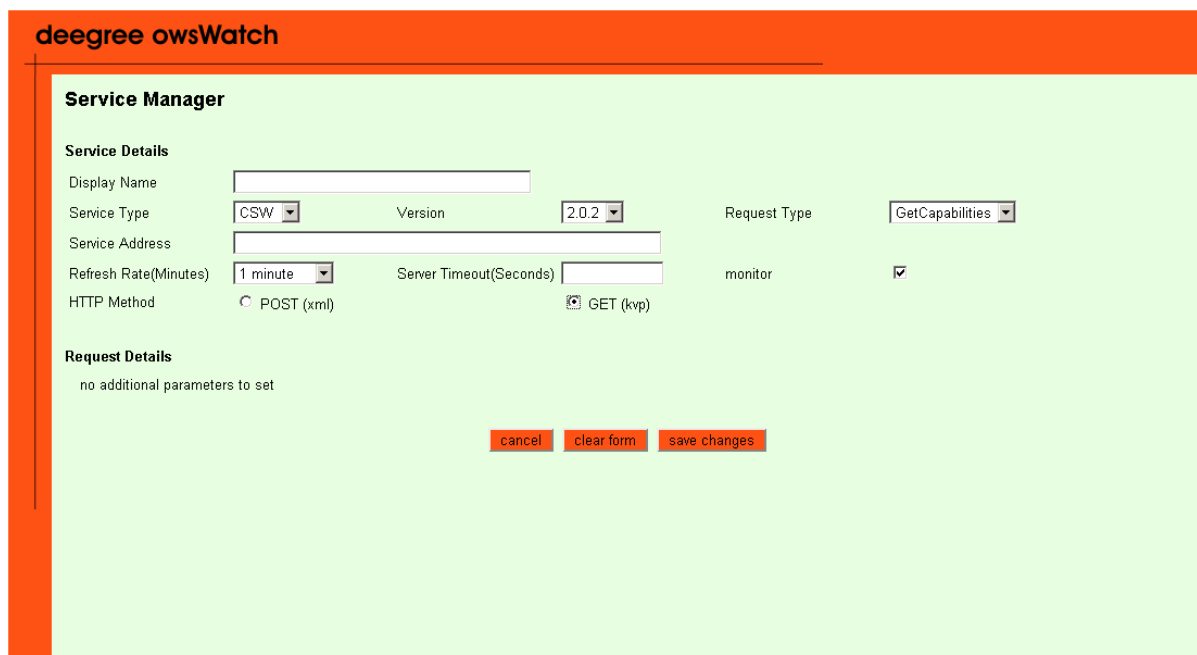


Figure 2: Service Manager window to add/edit tests

Fill in the fields as described in the titles. The following fields exist in the add/edit monitor dialogue:

- **Display Name:** Title of the test
- **Service Type:** WMS, WFS, etc...
- **Version:** of the Service
- **Request Type:** GetCapabilities, GetMap, etc...

- **Service Address:** Address where the requests will be sent. It should look something like the following: `http://www.myserver/servicetype/services?`
- **Refresh Rate** (in minutes): determines how oft the test should be executed
- **Server Timeout** (in seconds): determines how long owsWatch should wait for the service before returning a timeout error
- **Monitor:** decides whether the test should be active or not, i.e. should the service be tested or not
- **Http Method:** Either POST or GET. If you choose POST method a Text area will be shown where the user will be able to enter a specific xml Request. Also there are in some cases preconfigured requests that are added automatically, once you choose POST method, but please not this does not apply for all cases. If you would like to add your default xml requests to a specific test type, please consult the next chapter “Adding and modifying test types”. In post requests there are specific text fields that corresponds to the service and the request type; For example in WMS GetMap request, there will be text fields like BBOX, Layers and Styles.

Also please note that sometimes POST or GET will be dimmed. This means either that either the OGC has no specification for such a request or it is yet not implemented in owsWatch.

Important to note that on editing a service, the fields Service Type and Request Type will be dimmed. This is so, so that same protocol files won't contain different test types. So if you need to change either the Service Type and/or the Request Type of a test you will have to delete the test at hand and create a new one, so that a new protocol file will be created.

5 Adding and modifying test types

Adding a new test type, ex. Adding GetRecords request for CSW is actually easy, but it requires a little bit of Java knowledge, and also some HTML knowledge if you want to add a GET Request for that service. Modifying an existing test type on the other side would require only some HTML knowledge. So lets start with the difficult one first and add a new service.

5.1 Adding a new test type

As explained briefly at the beginning of Chapter 3 on how the servlets work, a request is sent from the client (the Internet browser) to the server (the servlet), which in turn handles the request and sends the answer back to the client.

When a request is sent to the owsWatch main servlet it processes the request, receives the answer from the server and validate it with a validator. This validator will be the only Java class we will have to write.

The validator class must be created in the deegree2 code under org.deegree.portal.owsWatch.validator. The name of the of the validator consists of two parts. First part is the service name (most probably all capital letters), ex. CSW. Second part is the request name, ex. GetRecords. So according to these examples the class name should be CSWGetRecords.java. Please note that this convention is a must otherwise the test type can not be validated. Also this class must inherit the class AbstractValidator.

Now if you just want to check whether the server answered with a valid xml response and this response contains no ServiceException, then you can simply use the method validateAnswer from the AbstarctValidator. The following piece of code is enough for the above mentioned requirement:

```
public class CSWGetCapabilitiesValidator extends AbstractValidator implements
Serializable {

    @Override
    public ValidatorResponse validateAnswer( HttpMethodBase method, int
statusCode ) {
        return super.validateAnswer( method, statusCode );
    }
}
```

But then again, you might need a more specific validator for a certain need, in this case you'll have to have your own implementation in this class.

5.2 Adding/editing a test type. The client side of the story

First of all you have to change the configuration file according to your need. We will continue with the example from 5.1 and suppose we want to add a CSW GetRecords request, then we shall add the following snippet to the

owswatch_configuration.xml within the `wc:ServiceConfiguration` node under the root node:

```
<wc:Service type="OGC:CSW" version="2.0.2">
  <wc:ServiceRequest name="GetCapabilities" isPostable="1" isGettable="1">
    <wc:GetForm>./request_snippets/all_getCapabilities_get.html</wc:GetForm>
    <wc:PostForm>./request_snippets/csw_202_getcapabilities_post.xml</wc:PostForm>
  </wc:ServiceRequest>
  <wc:ServiceRequest name="GetRecords" isPostable="1" isGettable="1">
    <wc:GetForm>./request_snippets/csw_202_getRecords_get.html</wc:GetForm>
    <wc:RequestParameters>
      <wc:Key>outputSchema</wc:Key>
      <wc:Key>resultType</wc:Key>
      <wc:Key>outputFormat</wc:Key>
      <wc:Key>typeNames</wc:Key>
      <wc:Key>ElementSetName</wc:Key>
    </wc:RequestParameters>
  </wc:ServiceRequest>
</wc:Service>
```

In the attributes of the **Service** element there are the specific details of the test type which are the Service and Version.

The child element for the Service is **ServiceRequest**, which contains its name and also two attributes *isPostable* and *isGettable*, which determines whether the request could be sent by POST and/or GET Requests or not.

The element **GetForm** contains a path to an HTML file which contains an HTML snippet that is displayed when the user in owsWatch Service Manager chooses the HTTPMETHOD POST.

```
<table border="0" cellpadding="2" cellspacing="2">
  <tbody>
    <tr>
      <td>
        Output Schema   
      </td>
      <td>
        <select id="outputSchema">
          <option value="http://www.isotc211.org/2005/gmd">
            http://www.isotc211.org/2005/gmd
          </option>
          <option value="dublincore">dublincore</option>
          <option value="http://www.opengis.net/cat/csw/2.0.2">
            http://www.opengis.net/cat/csw/2.0.2
          </option>
        </select>
      </td>
    </tr>
    <tr>
      <td>
        Result Type   
      </td>
      <td>
        <select id=resultType>
          <option value="hits">hits</option>
        </select>
      </td>
    </tr>
    <tr>
      <td>
        Output Format   
    </td>
    <td>
        <select id="outputFormat">
            <option value="text/xml">text/xml</option>
        </select>
    </td>
</tr>
<tr>
    <td>
        Type Name   
    </td>
    <td>
        <input id="typeName" name="typeName" size="20" type="text" />
    </td>
</tr>
<tr>
    <td>
        Elementset Name   
    </td>
    <td>
        <select id="ElementSetName">
            <option value="full">full</option>
            <option value="brief">brief</option>
            <option value="summary">summary</option>
        </select>
    </td>
</tr>
</tbody>
</table>

```

Here you can have the page anyway you like to. As long as all the parameters of the wanted requests are available. Also note that the only HTML controllers allowed for this mechanism are: text field, radio button and list, and that the id of these controllers must correspond to an actual parameter in the CSW GetRecords request. Ex. ElementSetName is the id of a list and also an actual parameter in the GET Request.

The **RequestParameter** element contains the keys which shall be fetched from the html snippet mentioned above (note: keys also correspond to the Ids of the HTML controllers). This is just a sort of a map, which guides the javascript what to fetch from the HTML snippet.

The **PostForm** element contains a path to an xml snippet that contains an actual request of the corresponding service. This shall be used when the user chooses POST in the HTTPMETHOD.

Appendix A Configuration file for owsWatch

\$owsWatch_home\$/WEB-INF/conf/owswatch/owswatch_configuration.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- This file is part of deegree, for copyright/license information, please visit
http://www.deegree.org/license. -->
<wc:Configuration version="1.0.0"
xmlns:wc="http://www.deegree.org/owswatch/config">
  <wc:GeneralConfiguration>
    <!-- Global Refresh Rate unit is minute -->
    <wc:GlobalRefreshRate>1</wc:GlobalRefreshRate>
    <!-- PLEASE set these parameters to values suitable for your environment -->
    <wc:Users>
      <wc:User>
        <wc:UserName>SEC_ADMIN</wc:UserName><!-- DO *NOT* CHANGE THIS NAME -->
        <wc:Password>MY_SECRET_PASSWORD</wc:Password><!-- CHANGE THIS PASSWORD -->
        <wc:FirstName>AdminsFirstName</wc:FirstName>
        <wc:LastName>AdminsLastName</wc:LastName>
        <wc:Email>admin@some-place.org</wc:Email>
        <wc:Roles>admin,user</wc:Roles>
      </wc:User>
    </wc:Users>
    <!-- PLEASE set these parameters to values suitable for your environment -->
    <wc:Mail>
      <wc:mailFrom>info@some-place.org</wc:mailFrom>
      <wc:mailServer>mail.some-place.org</wc:mailServer>
    </wc:Mail>
    <wc:Location>
      <!-- Location of protocol files-->
      <wc:ProtocolLocation>./protocols</wc:ProtocolLocation>
      <!-- Location of service instace file; stores session based data of owsConfig
-->
      <wc:ServiceInstanceLocation>./services.xml</wc:ServiceInstanceLocation>
      <!-- address of the server as written in the email (for the protocol
location) -->
      <!-- PLEASE set this parameter to something suitable for your environment -->
      <wc:ServiceAddress>http://localhost:8080/owswatch</wc:ServiceAddress>
    </wc:Location>
  </wc:GeneralConfiguration>
  <wc:ServiceConfiguration>
    <!-- Service Test Interval unit is minute -->
    <wc:TestInterval>
      <wc:Value>1</wc:Value>
      <wc:Value>5</wc:Value>
      <wc:Value>10</wc:Value>
      <wc:Value>15</wc:Value>
      <wc:Value>30</wc:Value>
      <wc:Value>60</wc:Value>
      <wc:Value>120</wc:Value>
      <wc:Value>1440</wc:Value><!-- once a day -->
      <wc:Value>10080</wc:Value><!-- once a week -->
    </wc:TestInterval>
    <wc:Services>
      <wc:Service type="OGC:WMS" version="1.1.1">
        <wc:ServiceRequest name="GetCapabilities" isPostable="0" isGettable="1">
          <wc:GetForm>./request_snippets/all_getCapabilities_get.html</wc:GetForm>
        </wc:ServiceRequest>
        <wc:ServiceRequest name="GetMap" isPostable="1" isGettable="1">
          <wc:GetForm>./request_snippets/wms_111_getMap_get.html</wc:GetForm>
          <wc:RequestParameters>
            <wc:Key>WIDTH</wc:Key>
            <wc:Key>HEIGHT</wc:Key>
            <wc:Key>LAYERS</wc:Key>
            <wc:Key>STYLES</wc:Key>
            <wc:Key>TRANSPARENT</wc:Key>
            <wc:Key>FORMAT</wc:Key>
            <wc:Key>SRS</wc:Key>
          </wc:RequestParameters>
        </wc:ServiceRequest>
      </wc:Service>
    </wc:Services>
  </wc:ServiceConfiguration>
</wc:Configuration>
```



```

        <wc:Key>BBOX</wc:Key>
    </wc:RequestParameters>
</wc:ServiceRequest>
</wc:Service>
<wc:Service type="OGC:WFS" version="1.1.0">
    <wc:ServiceRequest name="GetCapabilities" isPostable="1" isGettable="1">
        <wc:GetForm>./request_snippets/all_getCapabilities_get.html</wc:GetForm>
        <wc:PostForm>
            ./request_snippets/wfs_110_getcapabilities_post.xml
        </wc:PostForm>
    </wc:ServiceRequest>
    <wc:ServiceRequest name="GetFeature" isPostable="1" isGettable="1">
        <wc:GetForm>./request_snippets/wfs_110_getFeature_get.html</wc:GetForm>
        <wc:RequestParameters>
            <wc:Key>NAMESPACE</wc:Key>
            <wc:Key>TYPENAME</wc:Key>
        </wc:RequestParameters>
    </wc:ServiceRequest>
</wc:Service>
<wc:Service type="OGC:WCS" version="1.0.0">
    <wc:ServiceRequest name="GetCapabilities" isPostable="1" isGettable="1">
        <wc:GetForm>./request_snippets/all_getCapabilities_get.html</wc:GetForm>
        <wc:PostForm>
            ./request_snippets/wcs_100_getcapabilities_post.xml
        </wc:PostForm>
    </wc:ServiceRequest>
    <wc:ServiceRequest name="GetCoverage" isPostable="1" isGettable="1">
        <wc:GetForm>./request_snippets/wcs_100_getcoverage_get.html</wc:GetForm>
        <wc:RequestParameters>
            <wc:Key>coverage</wc:Key>
            <wc:Key>width</wc:Key>
            <wc:Key>height</wc:Key>
            <wc:Key>BBOX</wc:Key>
            <wc:Key>format</wc:Key>
            <wc:Key>crs</wc:Key>
        </wc:RequestParameters>
    </wc:ServiceRequest>
</wc:Service>
<wc:Service type="OGC:CSW" version="2.0.2">
    <wc:ServiceRequest name="GetCapabilities" isPostable="1" isGettable="1">
        <wc:GetForm>./request_snippets/all_getCapabilities_get.html</wc:GetForm>
        <wc:PostForm>
            ./request_snippets/csw_202_getcapabilities_post.xml
        </wc:PostForm>
    </wc:ServiceRequest>
    <wc:ServiceRequest name="GetRecords" isPostable="1" isGettable="1">
        <wc:GetForm>./request_snippets/csw_202_getRecords_get.html</wc:GetForm>
        <wc:RequestParameters>
            <wc:Key>outputSchema</wc:Key>
            <wc:Key>resultType</wc:Key>
            <wc:Key>outputFormat</wc:Key>
            <wc:Key>typeNameNames</wc:Key>
            <wc:Key>ElementSetName</wc:Key>
        </wc:RequestParameters>
    </wc:ServiceRequest>
</wc:Service>
<wc:Service type="OGC:SOS" version="1.0.0">
    <wc:ServiceRequest name="GetCapabilities" isPostable="1" isGettable="1">
        <wc:PostForm>./request_snippets/sos_100_getCapabilities.xml</wc:PostForm>
    </wc:ServiceRequest>
    <wc:ServiceRequest name="DescribeSensor" isPostable="1" isGettable="0">
    </wc:ServiceRequest>
</wc:Service>
</wc:Services>
</wc:ServiceConfiguration>
</wc:Configuration>

```

Appendix B Created Tests for the different services

\$owsWatch_home\$/WEB-INF/conf/owswatch/services.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<watch:Config xmlns:watch="http://www.deegree.org/owswatch/services"
service_id_sequence="8">
  <watch:SERVICEMONITOR id="1">
    <watch:ACTIVE>false</watch:ACTIVE>
    <watch:TIMEOUT>30</watch:TIMEOUT>
    <watch:INTERVAL>1440</watch:INTERVAL>
    <watch:ONLINERESOURCE>
      http://demo.deegree.org/deegree-wms/services?
    </watch:ONLINERESOURCE>
    <watch:SERVICENAME>WMS demo Capa</watch:SERVICENAME>
    <watch:HTTPMETHOD type="GET">
      <watch:REQUEST>GetCapabilities</watch:REQUEST>
      <watch:VERSION>1.1.1</watch:VERSION>
      <watch:SERVICE>WMS</watch:SERVICE>
    </watch:HTTPMETHOD>
  </watch:SERVICEMONITOR>
  <watch:SERVICEMONITOR id="2">
    <watch:ACTIVE>false</watch:ACTIVE>
    <watch:TIMEOUT>30</watch:TIMEOUT>
    <watch:INTERVAL>1440</watch:INTERVAL>
    <watch:ONLINERESOURCE>
      http://demo.deegree.org/deegree-wfs/services?
    </watch:ONLINERESOURCE>
    <watch:SERVICENAME>WFS demo Capa</watch:SERVICENAME>
    <watch:HTTPMETHOD type="GET">
      <watch:REQUEST>GetCapabilities</watch:REQUEST>
      <watch:VERSION>1.1.0</watch:VERSION>
      <watch:SERVICE>WFS</watch:SERVICE>
    </watch:HTTPMETHOD>
  </watch:SERVICEMONITOR>
  <watch:SERVICEMONITOR id="3">
    <watch:ACTIVE>false</watch:ACTIVE>
    <watch:TIMEOUT>31</watch:TIMEOUT>
    <watch:INTERVAL>60</watch:INTERVAL>
    <watch:ONLINERESOURCE>
      http://demo.deegree.org/deegree-wcs/services?
    </watch:ONLINERESOURCE>
    <watch:SERVICENAME>WCS demo Capa</watch:SERVICENAME>
    <watch:HTTPMETHOD type="GET">
      <watch:REQUEST>GetCapabilities</watch:REQUEST>
      <watch:VERSION>1.0.0</watch:VERSION>
      <watch:SERVICE>WCS</watch:SERVICE>
    </watch:HTTPMETHOD>
  </watch:SERVICEMONITOR>
  <watch:SERVICEMONITOR id="4">
    <watch:ACTIVE>false</watch:ACTIVE>
    <watch:TIMEOUT>30</watch:TIMEOUT>
    <watch:INTERVAL>120</watch:INTERVAL>
    <watch:ONLINERESOURCE>
      http://demo.deegree.org/deegree-csw/services?
    </watch:ONLINERESOURCE>
    <watch:SERVICENAME>CSW demo Capa</watch:SERVICENAME>
    <watch:HTTPMETHOD type="GET">
      <watch:REQUEST>GetCapabilities</watch:REQUEST>
      <watch:VERSION>2.0.2</watch:VERSION>
      <watch:SERVICE>CSW</watch:SERVICE>
    </watch:HTTPMETHOD>
  </watch:SERVICEMONITOR>
  <watch:SERVICEMONITOR id="5">
    <watch:ACTIVE>false</watch:ACTIVE>
    <watch:TIMEOUT>30</watch:TIMEOUT>
    <watch:INTERVAL>60</watch:INTERVAL>
```

```

<watch:ONLINERESOURCE>
  http://demo.deegree.org/deegree-wms/services?
</watch:ONLINERESOURCE>
<watch:SERVICENAME>WMS demo GetMap</watch:SERVICENAME>
<watch:HTTPMETHOD type="GET">
  <watch:SERVICE>WMS</watch:SERVICE>
  <watch:LAYERS>Counties</watch:LAYERS>
  <watch:FORMAT>image/gif</watch:FORMAT>
  <watch:HEIGHT>500</watch:HEIGHT>
  <watch:TRANSPARENT>TRUE</watch:TRANSPARENT>
  <watch:REQUEST>GetMap</watch:REQUEST>
  <watch:WIDTH>500</watch:WIDTH>
  <watch:BBOX>-191767.0,3776098.0,1044475.0,4905241.0</watch:BBOX>
  <watch:SRS>EPSG:26912</watch:SRS>
  <watch:STYLES>ColourfulCounties</watch:STYLES>
  <watch:VERSION>1.1.1</watch:VERSION>
</watch:HTTPMETHOD>
</watch:SERVICEMONITOR>
<watch:SERVICEMONITOR id="6">
  <watch:ACTIVE>>false</watch:ACTIVE>
  <watch:TIMEOUT>30</watch:TIMEOUT>
  <watch:INTERVAL>1440</watch:INTERVAL>
  <watch:ONLINERESOURCE>
    http://demo.deegree.org/deegree-wfs/services?
  </watch:ONLINERESOURCE>
  <watch:SERVICENAME>WFS demo GetFeature</watch:SERVICENAME>
  <watch:HTTPMETHOD type="GET">
    <watch:NAMESPACE>xmlns(app=http://www.deegree.org/app)</watch:NAMESPACE>
    <watch:REQUEST>GetFeature</watch:REQUEST>
    <watch:TYPENAME>app:StateBoundary</watch:TYPENAME>
    <watch:VERSION>1.1.0</watch:VERSION>
    <watch:SERVICE>WFS</watch:SERVICE>
  </watch:HTTPMETHOD>
</watch:SERVICEMONITOR>
<watch:SERVICEMONITOR id="7">
  <watch:ACTIVE>>false</watch:ACTIVE>
  <watch:TIMEOUT>10</watch:TIMEOUT>
  <watch:INTERVAL>1</watch:INTERVAL>
  <watch:ONLINERESOURCE>
    http://demo.deegree.org/deegree-wcs/services?
  </watch:ONLINERESOURCE>
  <watch:SERVICENAME>WCS demo GetCoverage</watch:SERVICENAME>
  <watch:HTTPMETHOD type="GET">
    <watch:width>500</watch:width>
    <watch:SERVICE>WCS</watch:SERVICE>
    <watch:REQUEST>GetCoverage</watch:REQUEST>
    <watch:format>jpeg</watch:format>
    <watch:BBOX>420857.5,4504382.5,431410.5,4518364.5</watch:BBOX>
    <watch:crs>EPSG:26912</watch:crs>
    <watch:coverage>saltlakecity</watch:coverage>
    <watch:height>500</watch:height>
    <watch:VERSION>1.0.0</watch:VERSION>
  </watch:HTTPMETHOD>
</watch:SERVICEMONITOR>
<watch:SERVICEMONITOR id="8">
  <watch:ACTIVE>>false</watch:ACTIVE>
  <watch:TIMEOUT>30</watch:TIMEOUT>
  <watch:INTERVAL>1</watch:INTERVAL>
  <watch:ONLINERESOURCE>
    http://demo.deegree.org/deegree-csw/services?
  </watch:ONLINERESOURCE>
  <watch:SERVICENAME>CSWGetRecords test</watch:SERVICENAME>
  <watch:HTTPMETHOD type="GET">
    <watch:ElementSetName>brief</watch:ElementSetName>
    <watch:resultType>hits</watch:resultType>
    <watch:REQUEST>GetRecords</watch:REQUEST>
    <watch:VERSION>2.0.2</watch:VERSION>
    <watch:typeNames>gmd:MD_Metadata</watch:typeNames>
    <watch:outputSchema>csw:profile</watch:outputSchema>
  </watch:HTTPMETHOD>
</watch:SERVICEMONITOR>

```

```
<watch:SERVICE>CSW</watch:SERVICE>  
<watch:outputFormat>text/xml</watch:outputFormat>  
</watch:HTTPMETHOD>  
</watch:SERVICEMONITOR>  
</watch:Config>
```

Appendix C Example for a web.xml file for owsWatch

\$owsWatch_home\$/WEB-INF/web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- This file is part of deegree, for copyright/license information, please visit
http://www.deegree.org/license. -->
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <display-name>owswatch</display-name>
  <description>deegree2 oswatch for OGC WebServices</description>
  <servlet>
    <servlet-name>owsWatch</servlet-name>
    <servlet-class>org.deegree.enterprise.servlet.OWSWatch</servlet-class>
    <init-param>
      <param-name>owsWatchConfiguration</param-name>
      <param-value>WEB-INF/conf/owswatch/owswatch_configuration.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet>
    <servlet-name>wprotocol</servlet-name>
    <servlet-class>org.deegree.enterprise.servlet.ProtocolServlet</servlet-class>
    <init-param>
      <param-name>owsWatchConfiguration</param-name>
      <param-value>WEB-INF/conf/owswatch/owswatch_configuration.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>owsWatch</servlet-name>
    <url-pattern>/owsWatch</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>wprotocol</servlet-name>
    <url-pattern>/wprotocol</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>owsWatch_outerframe.html</welcome-file>
  </welcome-file-list>
</web-app>
```