GeoServer User Manual

Release 2.5.x

GeoServer

January 10, 2014

GeoServer is an open source software server written in Java that allows users to share and edit geospatial data. Designed for interoperability, it publishes data from any major spatial data source using open standards.

This User Manual is a comprehensive guide to all aspects of using GeoServer. Whether you are a novice or a veteran user of this software, we hope that this documentation will be a helpful reference.

CHAPTER 1

Introduction

This section is for more information on GeoServer, its background, and what it can do for you.

For those who wish to get started with GeoServer right away, feel free to skip to the *Installation* section.

1.1 Overview

GeoServer is an open source software server written in Java that allows users to share and edit geospatial data. Designed for interoperability, it publishes data from any major spatial data source using open standards.

Being a community-driven project, GeoServer is developed, tested, and supported by a diverse group of individuals and organizations from around the world.

GeoServer is the reference implementation of the Open Geospatial Consortium (OGC) Web Feature Service (WFS) and Web Coverage Service (WCS) standards, as well as a high performance certified compliant Web Map Service (WMS). GeoServer forms a core component of the Geospatial Web.

1.2 History

GeoServer was started in 2001 by The Open Planning Project (TOPP), a non-profit technology incubator based in New York. TOPP was creating a suite of tools to enable open democracy and to help make government more transparent. The first of these was GeoServer, which came out of a recognition that a suite of tools to enable citizen involvement in government and urban planning would be greatly enhanced by the ability to share spatial data.

The GeoServer founders envisioned a Geospatial Web, analogous to the World Wide Web. With the World Wide Web, one can search for and download text. With the Geospatial Web, one can search for and download spatial data. Data providers would be able to publish their data straight to this web, and users could directly access it, as opposed to the now indirect and cumbersome methods of sharing data that exist today.

Those involved with GeoServer founded the GeoTools project, an open source GIS Java toolkit. Through GeoTools, support for Shapefiles, Oracle databases, ArcSDE integration, and much more was added.

Around the same time as GeoServer was founded, The OpenGIS Consortium (now the Open Geospatial Consortium) was working on the Web Feature Service standard. It specifies a protocol to make spatial data directly available on the web, using GML (Geographic Markup Language), an interoperable data format. A Web Map Service was also created, a protocol for creating and displaying map images created from spatial data.

Other projects became interrelated. Refractions Research created PostGIS, a free and open spatial database, which enabled GeoServer to connect to a free database. Also, MetaCarta created OpenLayers, an open source browser-based map viewing utility. Together, these tools are all have enhanced the functionality of GeoServer.

GeoServer can now output data to many other spatial data viewers, such as Google Earth, a popular 3-D virtual globe. In addition, GeoServer is currently working directly with Google in order to allow GeoServer data to be searchable on Google Maps. Soon a search for spatial data will be as easy as a Google search for a web page. Thus GeoServer is continuing on its mission to make spatial data more accessible to all.

1.3 Getting Involved

There are many ways that one can help out with the GeoServer project. GeoServer fully embraces an open source development model that does not see a split between user and developer, producer and consumer, but instead sees everyone as a valuable resource in a collaborative quest to build something better than any of us could alone.

1.3.1 Development

Helping to develop GeoServer is the obvious way to help out. Developers usually start with bug fixes and small patches, and then move into larger contributions as they learn the system. Our developers are more than happy to help out - try the Developers mailing list below - as you learn and get acquainted. We try our hardest to keep our code clean and well documented. You can find the project on github.

1.3.2 Documentation

One of the best and most needed ways to help out is with the documentation. Our official documentation is contained as part of our official code repository in order to maintain a uniform look and feel.

1.3.3 Mailing lists

GeoServer maintains two email lists: GeoServer Users and GeoServer Developers. These lists are publicly available and are a great resource for those who are new to GeoServer, who need a question answered, or who are interested in contributing code. The Users list is mainly for those who have questions relating to the use of GeoServer, and the Developers list is for more code-specific and roadmap-based discussions. If you see a question asked on these lists that you know the answer to, please respond!

1.3.4 IRC

GeoServer has an IRC channel, #geoserver, on the Freenode network. GeoServer developers frequent this channel, and so it is a great way to give and receive information in real time.

1.3.5 Bug tracking

If you have a problem when working with GeoServer, then please let us know through the mailing lists. GeoServer uses JIRA , a bug tracking website, to manage issue reports; you'll need to create an account first. As GeoServer is open source, everyone is encouraged to fix bugs and submit patches. Even if you are not a core developer, we welcome patches through JIRA, or pull requests to github.

1.3.6 Translation

We would like GeoServer available in as many languages as possible, just as we want spatial data to be available to all. The two areas of GeoServer to translate are the text for the *Web Administration Interface* and this documentation. Eventually we would even like to set up GeoServer community sites in different languages. If you are interested in this please let us know via the mailing lists.

1.3.7 Suggest improvements

If you have suggestions as to how we can make GeoServer better, we would love to hear them. You can contact us through the mailing lists linked above or in the IRC.

1.3.8 Spread the word

A further way to help out the GeoServer project is to spread the word about it. Word of mouth information sharing is more powerful than any amount spent on marketing, and the more people who use our software, the better it will become.

1.3.9 Fund improvements

A final way to help out is to push for GeoServer to be used in your own organization. A number of commerical organizations offer support for GeoServer, and any improvements made due to that funding will benefit the entire GeoServer community.

1.4 License

GeoServer is free software and is licensed under the `GNU General Public License`.:

```
GeoServer, open geospatial information server
Copyright (C) 2001 - 2011 The Open Planning Project dba OpenPlans
http://openplans.org

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version (collectively, "GPL").

As an exception to the terms of the GPL, you may copy, modify,
propagate, and distribute a work formed by combining GeoServer with the
Eclipse Libraries, or a work derivative of such a combination, even if
such copying, modification, propagation, or distribution would otherwise
violate the terms of the GPL. Nothing in this exception exempts you from
complying with the GPL in all respects for all of the code used other
than the Eclipse Libraries. You may include this exception and its grant
of permissions when you distribute GeoServer.  Inclusion of this notice
with such a distribution constitutes a grant of such permissions.  If
you do not wish to grant these permissions, remove this paragraph from
your distribution. "GeoServer" means the GeoServer software licensed
under version 2 or any later version of the GPL, or a work based on such
software and licensed under the GPL. "Eclipse Libraries" means Eclipse
Modeling Framework Project and XML Schema Definition software
distributed by the Eclipse Foundation and licensed under the Eclipse
```

```
Public License Version 1.0 ("EPL"), or a work based on such software and
licensed under the EPL.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Suite 500, Boston, MA 02110-1335  USA
```

This product includes software developed by the Apache Software Foundation (http://www.apache.org/) licensed under the `Apache License Version 2.0` and `Apache License Version 1.1`.

CHAPTER 2

Installation

There are many ways to install GeoServer on your system. This section will discuss the various installation paths available.

2.1 Windows

There are a few ways to install GeoServer on Windows. The simplest way is to use the Windows installer, but you can also perform a manual installation with the OS-independent binary.

Note: To run GeoServer as part of a servlet container such as Tomcat, please see the *Web archive (WAR)* section.

2.1.1 Windows Installer

The Windows installer provides an easy way to set up GeoServer on your system. With no configuration files to edit or command line setting, everything can be done through the Windows GUI.

1. Navigate to the GeoServer Download page at http://geoserver.org/display/GEOS/Download.
2. Select the version of GeoServer that you wish to download. If you're not sure, select the Stable version at http://geoserver.org/display/GEOS/Stable.
3. Click on the link for the Windows installer.

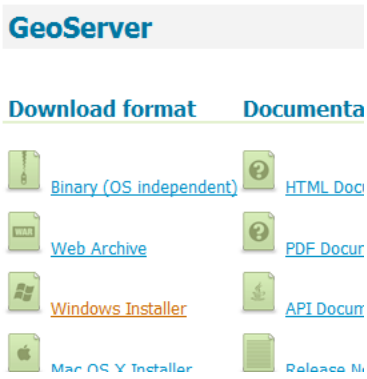


Figure 2.1: *Downloading the Windows installer*

4. After downloading, double-click on the file to launch.
5. At the Welcome screen, click *Next*.

Figure 2.2: *Welcome screen*

6. Read the *License* and click *I Agree*.

Figure 2.3: *GeoServer license*

7. Select the directory of the installation, then click *Next*.
8. Select the Start Menu directory name and location, then click *Next*.
9. Enter the path to a **valid Java Runtime Environment (JRE)**. GeoServer requires a valid JRE in order to run, so this step is required. The installer will inspect your system and attempt to automatically populate this box with the path contained in your `%JAVA_HOME%` variable. If this variable is undefined, you may not have a JRE on your system. In this case, you can download a JRE at http://www.oracle.com/technetwork/java/javase/downloads/index.html. Once the JRE is downloaded and installed, restart the GeoServer installer. When finished, click *Next*.

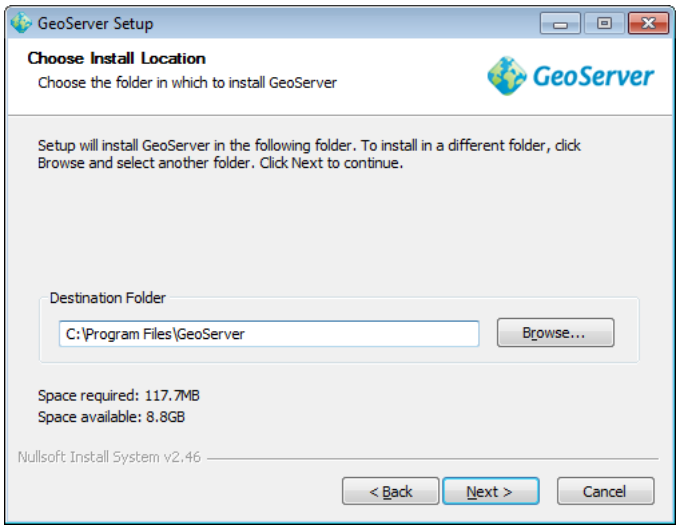


Figure 2.4: *GeoServer install directory*

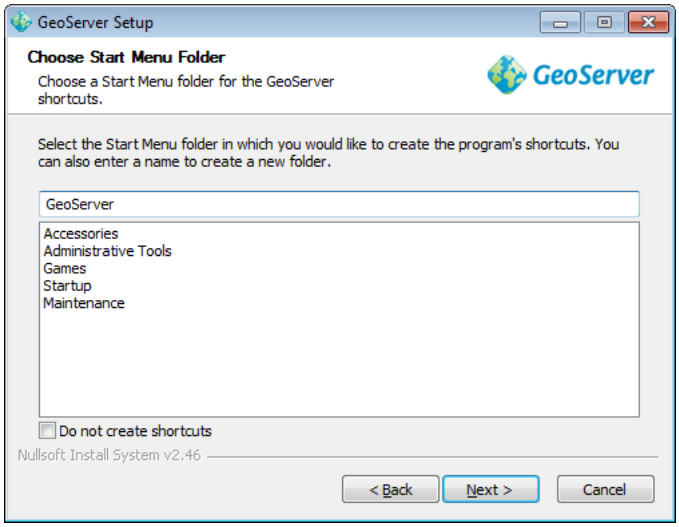


Figure 2.5: *Start menu location*

Note: An example of a valid path would be `C:\Program Files\Java\jre6`.

Warning: Don't include the `\bin` in the JRE path. So if `javaw.exe` is located at `C:\Program Files\Java\jre6\bin\javaw.exe`, set the path to be `C:\Program Files\Java\jre6`.

Note: Oracle Java SE 6 or newer is strongly recommended. (As of GeoServer 2.2.x, Oracle JRE 5 is no longer supported.) A Java Development Kit (JDK) is not required to run GeoServer. For more information about Java and GeoServer, please see the section on *Java Considerations*.

Figure 2.6: *Selecting a valid JRE*

10. Enter the path to your GeoServer data directory or select the default. Previous GeoServer users may already have a data directory that they wish to use. If this is your first time using GeoServer, you should select the *Default data directory*. When finished, click *Next*.

Figure 2.7: *Setting a GeoServer data directory*

11. Enter the username and password for administration of GeoServer. GeoServer's *Web Administration Interface* requires authentication for management, and what is entered here will become those administrator credentials. The defaults are *admin* / *geoserver*. It is recommended to change these from the defaults, but this is not required. When finished, click *Next*.

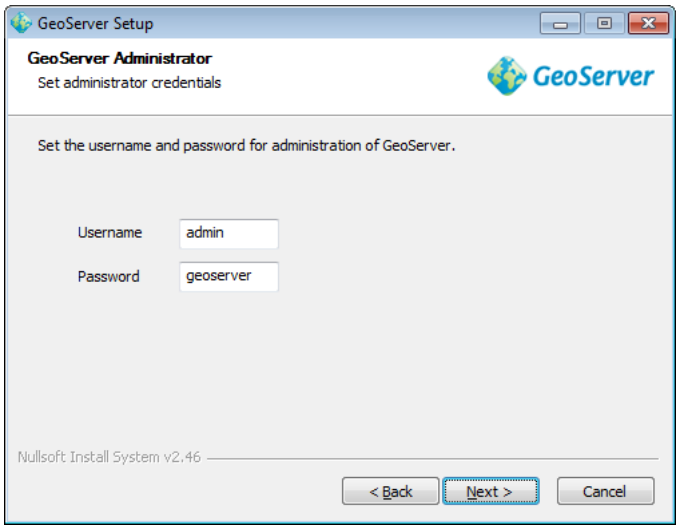


Figure 2.8: *Setting the username and password for GeoServer administration*

12. Enter the port that GeoServer will respond on. This affects the location of the GeoServer *Web Administration Interface*, as well as the endpoints of the GeoServer *Web Map Service* and *Web Feature Service*. The default port is *8080*, although any valid unused port will work. When finished, click *Next*.

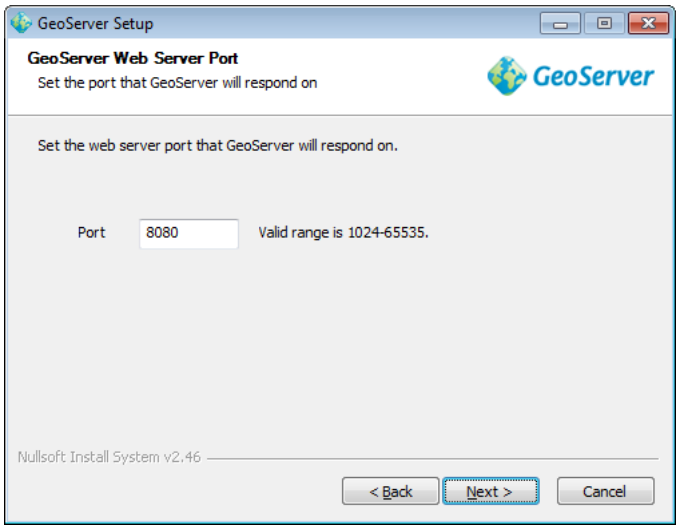


Figure 2.9: *Setting the GeoServer port*

13. Select whether GeoServer should be run manually or installed as a service. When run manually, GeoServer is run like a standard application under the current user. When installed as a service, GeoServer in integrated into Windows Services, and thus is easier to administer. If running on a server, or to manage GeoServer as a service, select *Install as a service*. Otherwise, select *Run manually*. When finished, click *Next*.

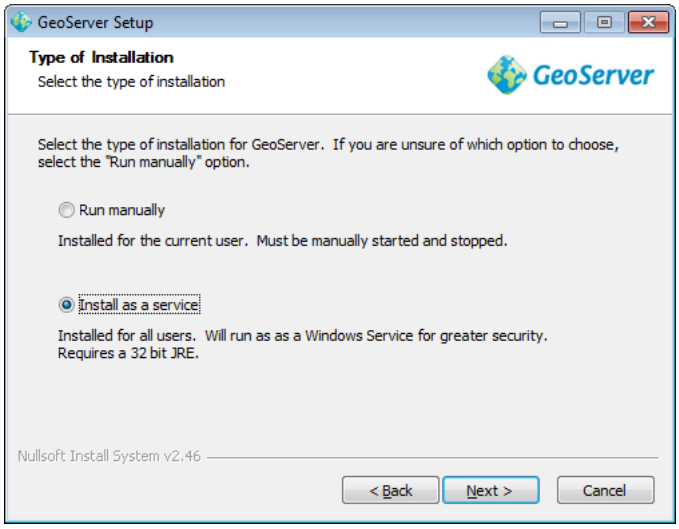


Figure 2.10: *Installing GeoServer as a service*

14. Review your selections and click the *Back* button if any changes need to be made. Otherwise, click *Install*.

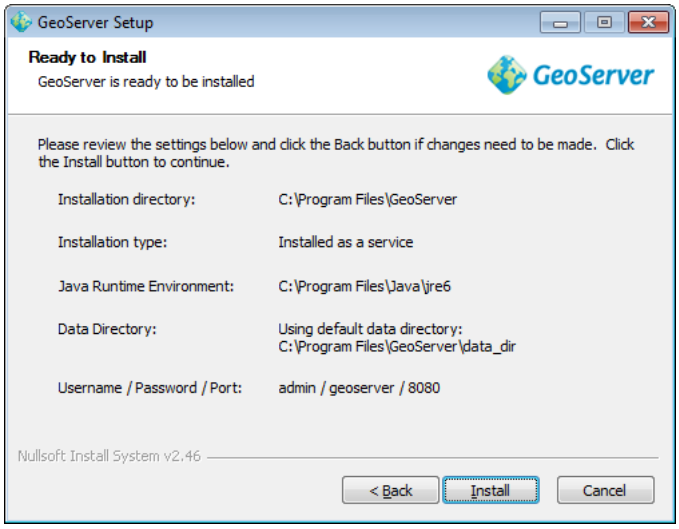


Figure 2.11: *Verifying settings*

15. GeoServer will install on your system. When finished, click *Finish* to close the installer.
16. If you installed GeoServer as a service, it is already running. Otherwise, you can start GeoServer by going to the Start Menu, and clicking *Start GeoServer* in the GeoServer folder.
17. Navigate to `http://[SERVER_URL]:[PORT]/geoserver/` (Ex: `http://localhost:8080/geoserver/`) to access the GeoServer *Web Administration Interface*.

If you see GeoServer in your browser, then congratulations, GeoServer is successfully installed!

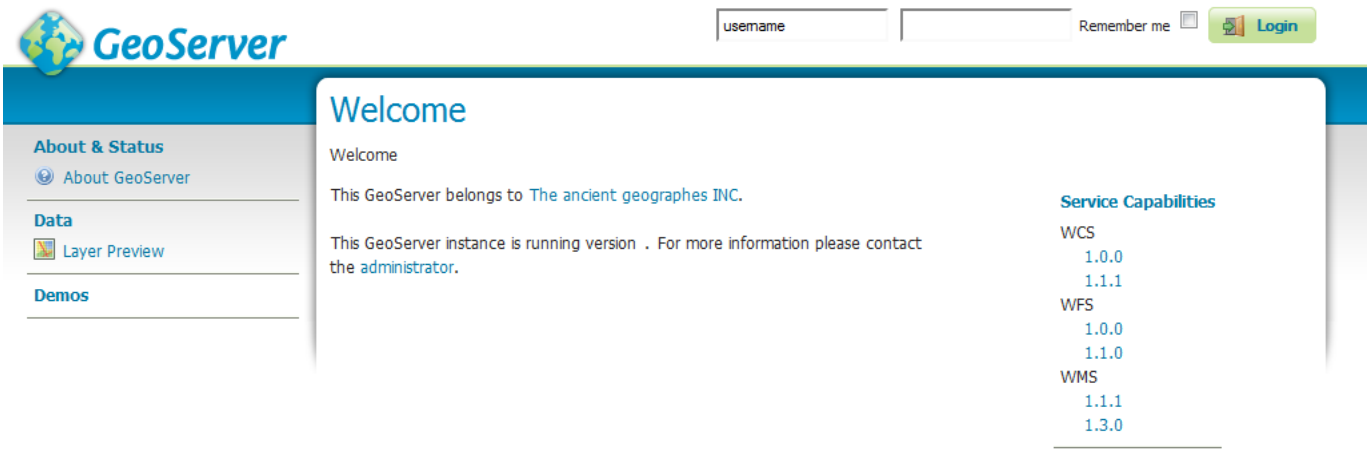


Figure 2.12: *GeoServer installed and running successfully*

2.1.2 Windows Binary

Note: This section is for the OS-independent binary. Please see the section on the *Windows Installer* for the wizard-based installer for Windows.

The most common way to install GeoServer is using the OS-independent binary. This version is a GeoServer web application (webapp) bundled inside Jetty, a lightweight servlet container system. It has the advantages of working very similarly across all operating systems plus being very simple to set up.

Installation

1. Navigate to the GeoServer Download page and pick the appropriate version to download.
2. Select *OS-independent binary* on the download page.
3. Download the archive, and unpack to the directory where you would like the program to be located. A typical place would be `C:\Program Files\GeoServer`.

Setting environment variables

You will need to set the `JAVA_HOME` environment variable if it is not already set. This is the path to your JDK/JRE such that `%JAVA_HOME%\bin\java.exe` exists. You can download a JRE at http://www.oracle.com/technetwork/java/javase/downloads/index.html.

Note: Oracle Java SE 6 or newer is strongly recommended. (As of GeoServer 2.2.x, Oracle JRE 5 is no longer supported.) A Java Development Kit (JDK) is not required to run GeoServer. For more information about Java and GeoServer, please see the section on *Java Considerations*.

1. Navigate to *Control Panel* → *System* → *Advanced* → *Environment Variables*.
2. Under *System variables* click *New*.
3. For *Variable name* enter `JAVA_HOME`. For *Variable value* enter the path to your JDK/JRE.
4. Click OK three times.

Note: You may also want to set the `GEOSERVER_HOME` variable, which is the directory where GeoServer is installed, and the `GEOSERVER_DATA_DIR` variable, which is the location of the GeoServer data directory (usually `%GEOSERVER_HOME\data_dir`). The latter is mandatory if you wish to use a data directory other than the one built in to GeoServer. The procedure for setting these variables is identical to the above. Note that the specified data directory should be a valid *GeoServer Data Directory*.

Running

Note: This can be done either via Windows Explorer or the command line.

1. Navigate to the `bin` directory inside the location where GeoServer is installed.
2. Run `startup.bat`. A command-line window will appear and persist. This window contains diagnostic and troubleshooting information. This window should not be closed, or else GeoServer will shut down.
3. To access the *Web Administration Interface*, navigate to `http://localhost:8080/geoserver`.

Stopping

Either close the persistent command-line window, or run the `shutdown.bat` file inside the `bin` directory.

Uninstallation

1. Stop GeoServer (if it is running)
2. Delete the directory where GeoServer is installed.

2.2 Mac OS X

There are a few ways to install GeoServer on OS X. The simplest way is to use the OS X installer, but you can also perform a manual installation with the OS-independent binary.

Note: To run GeoServer as part of a servlet container such as Tomcat, please see the *Web archive (WAR)* section.

2.2.1 Mac OS X Installer

1. Navigate to the GeoServer Download page and click your preferred GeoServer version–Stable, Latest or Nightly.
2. On the resulting page, download the *Mac OS X Installer* format of your preferred GeoServer version.
3. Double click on the `.dmg` file to start the download.
4. Drag the GeoServer icon to the Applications folder.
5. Navigate to your applications folder, and double click on the GeoServer icon.

Figure 2.13: *Starting the Mac OSX Installer of GeoServer*

Note: Accept any security warnings regarding GeoServer as an application downloaded from the Internet.

6. In the resulting GeoServer console window, start GeoServer by going to *Server→Start*.

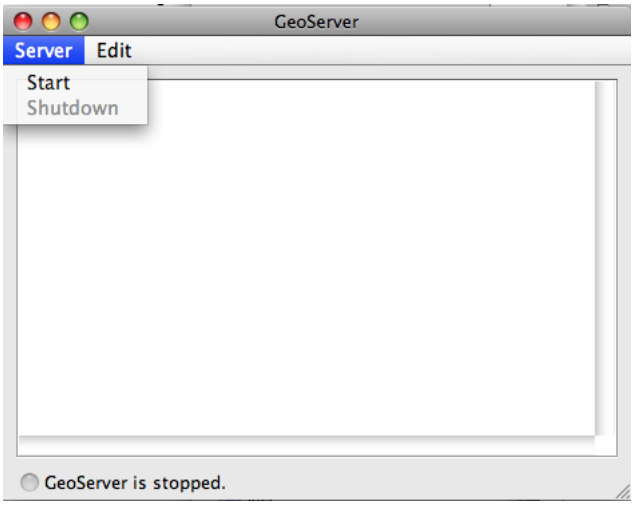


Figure 2.14: *Starting GeoServer*

7. The console window will log GeoServer's loading. Once GeoServer is completely started, a browser window will open at the URL `http://localhost:8080/geoserver`. Welcome to GeoServer!

2.2.2 Mac OS X Binary

Note: This section is for the OS-independent binary. Please see the section on the *Mac OS X* for the wizard-based installer for OS X.

The most common way to install GeoServer is using the OS-independent binary. This version is a GeoServer web application (webapp) bundled inside Jetty, a lightweight servlet container system. It has the advantages of working very similarly across all operating systems plus being very simple to set up.

Installation

1. Navigate to the GeoServer Download page and click your preferred GeoServer version–Stable, Latest or Nightly.
2. On the resulting page, download and save the *Binary (OS independent)* format of your preferred GeoServer version.

   Note: Download GeoServer wherever you find appropriate. In this example we download the GeoServer archive to the Desktop. If GeoServer is in a different location, simply replace `Desktop` in the following command to your own folder path.

3. After saving the Geoserver archive, move to the location of your download, by first opening a terminal window (*Applications→Utitlies→Terminal*) and then typing the following command:

   ```
   cd Desktop/
   ```

4. Confirm that you are in the right directory by listing its contents. You should see your specific GeoServer archive (e.g., `GeoServer-2.0-RC1-bin.zip`) by typing:

   ```
   ls -l
   ```

5. Unzip `geoserver-2.0-RC1.zip` to `/usr/local/geoserver` with the following two commands:

   ```
   unzip $geoserver-2.0-RC1.zip  .
   sudo mv geoserver-2.0-RC1/  geoserver
   ```

   Note: Notice the **.** in the first command. This means the archive will unzip in the current directory.

6. Add an environment variable to save the location of GeoServer by typing the following command:

   ```
   echo "export GEOSERVER_HOME=/usr/local/geoserver" >> ~/.profile
   . ~/.profile
   ```

7. Make yourself the owner of the `geoserver` folder. Type the following command in the terminal window, replacing `USER_NAME` with your own username :

   ```
   sudo chown -R USER_NAME /usr/local/geoserver/
   ```

8. Start GeoServer by changing into the directory `geoserver/bin` and executing the `startup.sh` script:

   ```
   cd geoserver-1.7.0/bin
   sh startup.sh
   ```

9. Visit `http://localhost:8080/geoserver` in a web browser.

2.3 Linux

Warning: Under construction

GeoServer requires Java to be installed on your system. Oracle Java SE 6 or newer is strongly recommended. (As of GeoServer 2.2.x, Oracle JRE 5 is no longer supported.) A Java Development Kit (JDK) is not required to run GeoServer. For more information about Java and GeoServer, please see the section on *Java Considerations*.

The most common way to install GeoServer is using the OS-independent binary. This version is a GeoServer web application (webapp) bundled inside Jetty, a lightweight servlet container system. It has the advantages of working very similarly across all operating systems plus being very simple to set up.

2.3.1 Debian

Setup a local geoserver instance with tomcat7 in Debian wheezy/sid.

Preparation

1. Follow instructions from *Web archive (WAR)* until step 3 and return here again.
2. Root permissions are needed, to make some changes to the linux system. Gain superuser rights by executing on a terminal: `su`
3. If not already available on your debian system, install the tomcat7 servlet container with your favourite package administration tool (synaptic, apt-get or aptitude). This tutorial uses aptitude: `aptitude install tomcat7`

Installation

1. Copy as user root the geoserver web application archive into tomcat7's webapp directory: `cp geoserver.war /var/lib/tomcat7/webapps`
2. Tomcat should recognize the WAR archive and immediately start to deploy the web application. This process takes some time and depends on your hardware used. Congratulations, your local geoserver is now up and running.

2.4 Web archive (WAR)

GeoServer is packaged as a standalone servlet for use with existing servlet container applications such as Apache Tomcat and Jetty.

Note: GeoServer has been mostly tested using Tomcat, and therefore these instructions may not work with other container applications.

2.4.1 Java

GeoServer requires Java to be installed on your system. Oracle Java SE 6 or newer is strongly recommended. (As of GeoServer 2.2.x, Oracle JRE 5 is no longer supported.) A Java Development Kit (JDK) is not

required to run GeoServer. For more information about Java and GeoServer, please see the section on *Java Considerations*.

2.4.2 Installation

1. Navigate to the GeoServer Download page and pick the appropriate version to download.
2. Select *Web archive* on the download page.
3. Download and unpack the archive. Copy the file `geoserver.war` to the directory that contains your container application's webapps.
4. Your container application should unpack the web archive and automatically set up and run GeoServer.

   Note: A restart of your container application may be necessary.

2.4.3 Running

Use your container application's method of starting and stopping webapps to run GeoServer.

1. To access the *Web Administration Interface*, open a browser and navigate to `http://container_application_URL/geoserver`. For example, with Tomcat running on port 8080 on localhost, the URL would be `http://localhost:8080/geoserver`.

2.4.4 Uninstallation

1. Stop the container application.
2. Remove the GeoServer webapp from the container application's webapps directory.

2.5 Upgrading

The general GeoServer upgrade process involves installing the new version on top of the old and ensuring it points at the same data directory used by the previous version. See *Migrating a Data Directory between different versions* for more details.

This section contains details about upgrading to specific GeoServer versions.

2.5.1 Upgrade to 2.2

Security configuration

GeoServer 2.2 comes with a significant retrofit of the *Security* subsystem. The changes focus mostly on authentication and user management. On upgrade GeoServer will update configuration in the `security` directory. The specific changes are described *here*.

Obtaining a master password

Starting with Geoserver 2.2 a master password is needed. This password is used to log in as `root` user and to protect the Geoserver key store.

During the upgrade process, Geoserver tries to find a proper master password. The following rules apply

- The default admin password `geoserver` is not allowed.
- The minimal length of the password is 8 characters.

The algorithm for finding a password:

1. Look for an existing user called `admin`. If there is such a user and the password obeys the rules above, use it.
2. Look for a user having the role `ROLE_ADMINISTRATOR`. If there is such a user and the password obeys the rules above, use it.
3. Generate a random password with 8 characters of length

The algorithm stores a file `masterpw.info` into the `security` directory. If an existing password of a user is used, the content of this file is like

```
This file was created at 2012/08/11 15:57:52

Master password is identical to the password of user: admin

Test the master password by logging in as user "root"

This file should be removed after reading !!!.
```

If a master password was generated, the content is like

```
This file was created at 2012/08/11 15:57:52

The generated master password is: pw?"9bWL

Test the master password by logging in as user "root"

This file should be removed after reading !!!.
```

After reading this file, remember the master password and remove this file.

RESTconfig security and administrative access

The security changes also include a new type of access mode for layer level security that allows for controlling administrative access to workspaces. In this context administrative access includes access via the web admin ui, or the RESTconfig api. For more details see *Layer security*.

A side effect of this change can have consequences for RESTconfig api users. Previously access via REST was controlled by specifying constraints on url patterns as described *here*. Administrative workspace/layer security now adds a second level of access control. Therefore in order for a user to access resources via REST that user must also have sufficient administrative privileges.

By default administrative access for workspaces/layers is granted to the `ROLE_ADMINISTRATOR` role. So if REST security defines url level constraints that involve roles with lesser privileges, access to resources will be denied. The most common case of this is when users make the REST api accessible anonymously.

The solution to this problem is to reduce the administrative access role to that required by REST url security. In the case where access to the REST api is granted anonymously this is **not recommended**. Allowing a server to be administered anonymously is a huge security hole.

CHAPTER 3

Getting Started

This section contains short tutorials for common tasks in GeoServer, to get new users using the system quickly and easily.

3.1 Web Administration Interface Quickstart

The *Web Administration Tool* is a web-based application used to configure all aspects of GeoServer, from adding and publishing data to changing service settings.

The web admin tool is accessed via a web browser at `http://<host>:<port>/geoserver` (for a default installation on the local host the link is http://localhost:8080/geoserver/web). When the app starts it displays the public Welcome page.

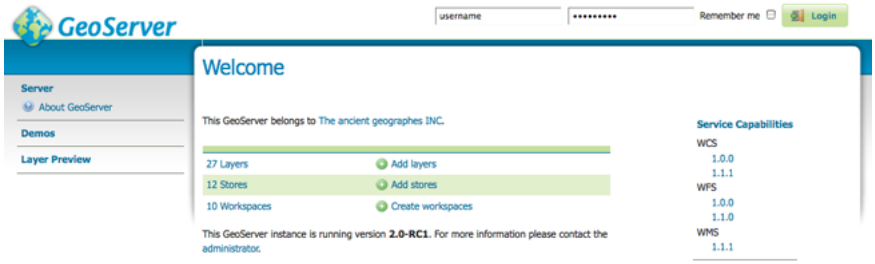


Figure 3.1: *Welcome Page*

3.1.1 Logging In

In order to change any server settings or configure data a user must first be authenticated. Navigate to the upper right hand corner to log into GeoServer. The default username and password is `admin` and `geoserver`. These can be changed by editing the `security/users.properties` file in the *GeoServer Data Directory*.

Figure 3.2: *Login*

Once logged in, the Welcome screen changes to show the available admin functions. These are available from links under the sections on the left-hand menu.

3.1.2 Server

The *Server* section provides access to GeoServer environment information. It is a combination of diagnostic and configuration tools, and can be particularly useful for debugging.

The *Server Status* page shows a summary of server configuration parameters and run-time status.

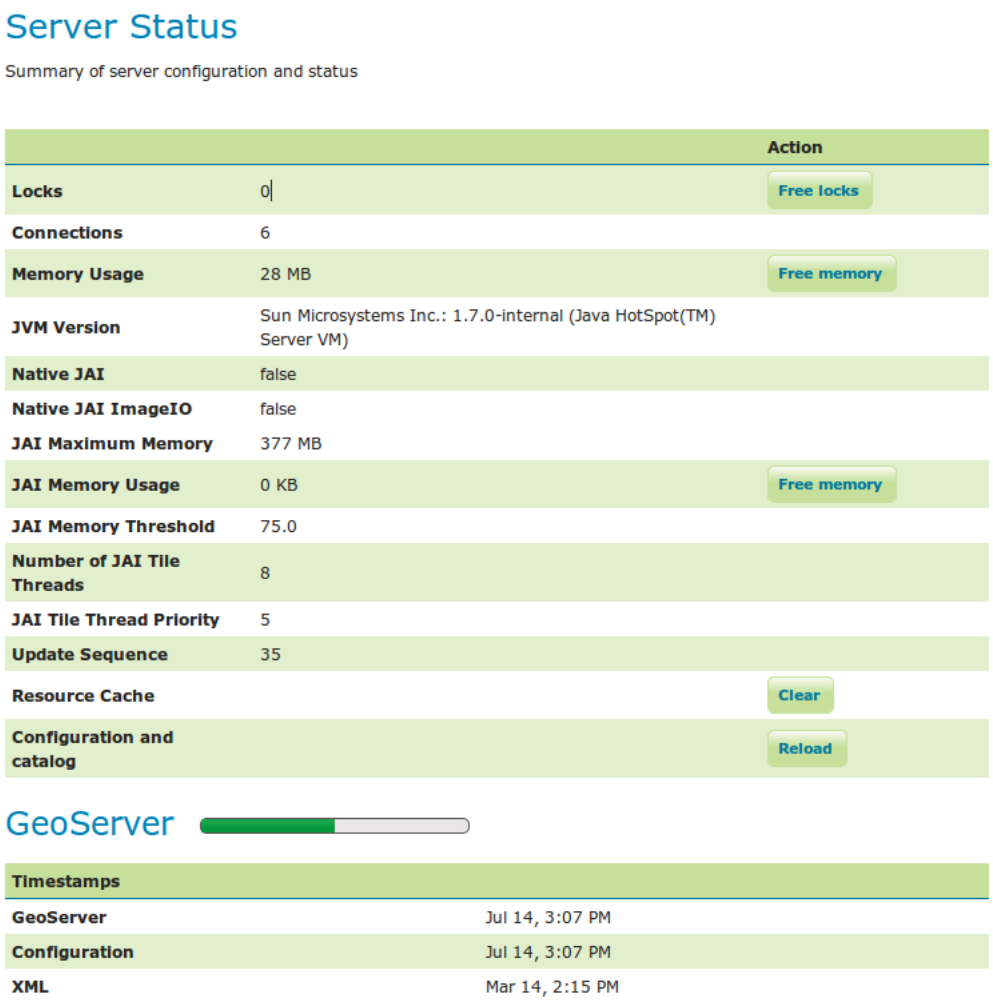


Figure 3.3: *Status Page*

The *Contact Information* page sets the public contact information in the Capabilities document of the WMS server.

The *Global Settings* page configures messaging, logging, character and proxy settings for the entire server.

The *JAI Settings* page is used to configure several JAI parameters, used by both WMS and WCS operations.

The *About GeoServer* section provides links to the GeoServer documentation, homepage and bug tracker.

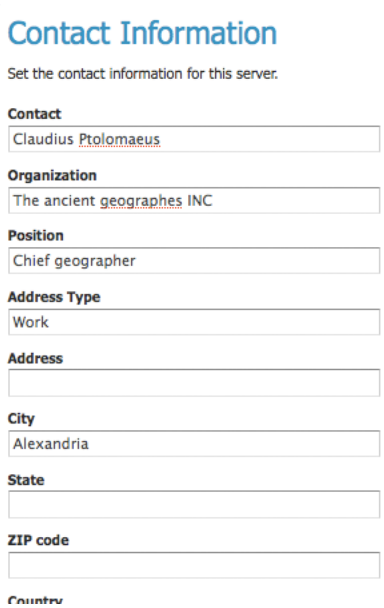


Figure 3.4: *Contact Page*

3.1.3 Services

The *Services* section is for advanced users needing to configure the request protocols used by GeoServer. The Web Coverage Service (*WCS*) page manages metadata information, common to WCS, WFS and WMS requests. The Web Feature Service (*WFS*) page permits configuration of features, service levels, and GML output. The Web Map Service (*WMS*) page sets raster and SVG options.

3.1.4 Data

The *Data* links directly to a data type page with edit, add, and delete functionality. All data types subsections follow a similar workflow. As seen in the *Styles* example below, the first page of each data type displays a view page with an indexed table of data.

Each data type name links to a corresponding configuration page. For example, all items listed below Workspace, Store and Layer Name on the *Layers* view page, link to its respective configuration page.

In the data type view panel, there are three different ways to locate a data type–sorting, searching, and scrolling .

To alphabetically sort a data type, click on the column header.

For simple searching, enter the search criteria in the search box and hit Enter.

To scroll through data type pages, use the arrow button located on the bottom and top of the view table.

As seen in the *Stores* example below, the buttons for adding and removing a data type can be found at the top of the view page.

To add a new data, select the *Add* button, and follow the data type specific prompts. To delete a data type In order to remove a data type, click on the data type's corresponding check box and select the *Remove* button. (Multiple data types, of the same kind, can be checked for batch removal.)

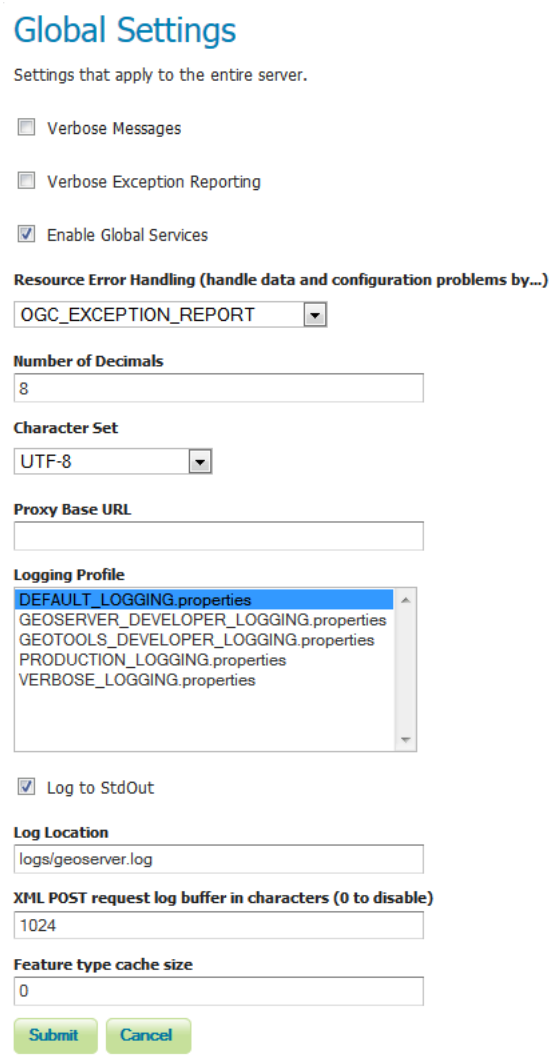


Figure 3.5: *Global Settings Page*

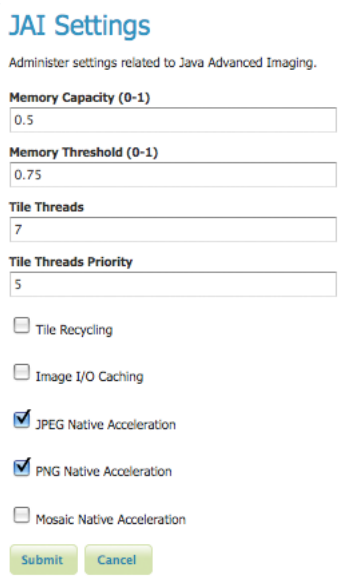


Figure 3.6: *JAI Settings*

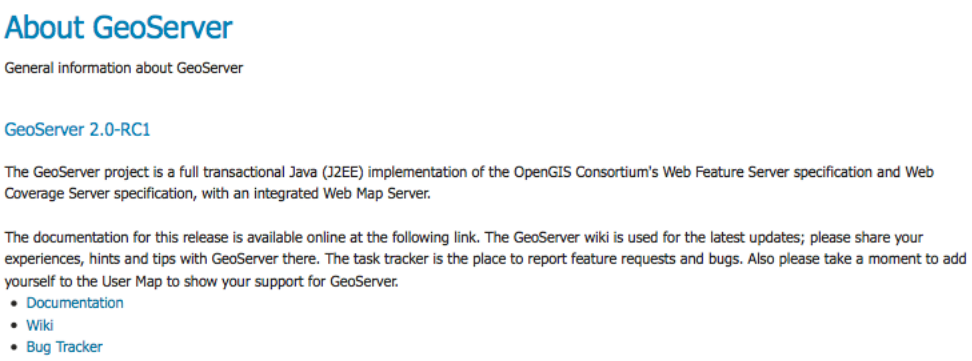


Figure 3.7: *About Section*

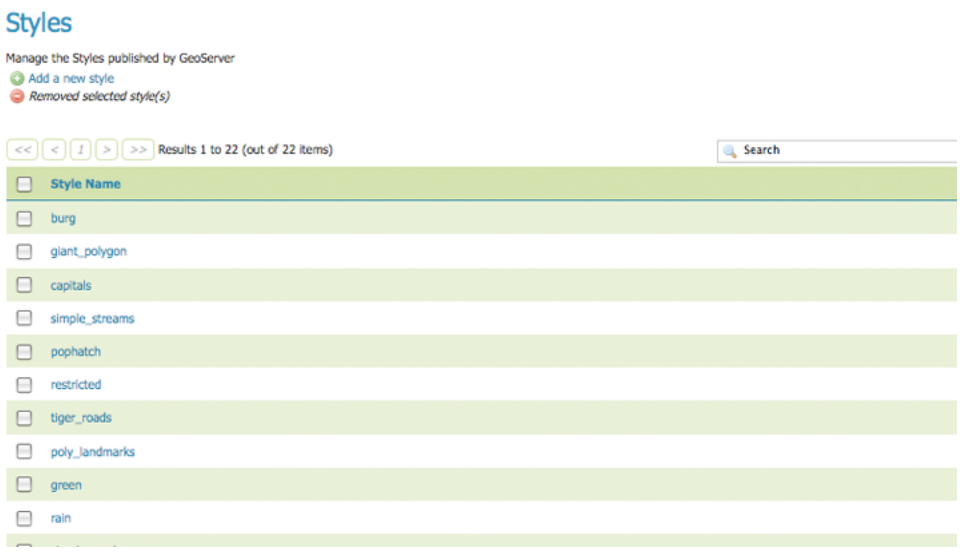


Figure 3.8: *Styles View page*

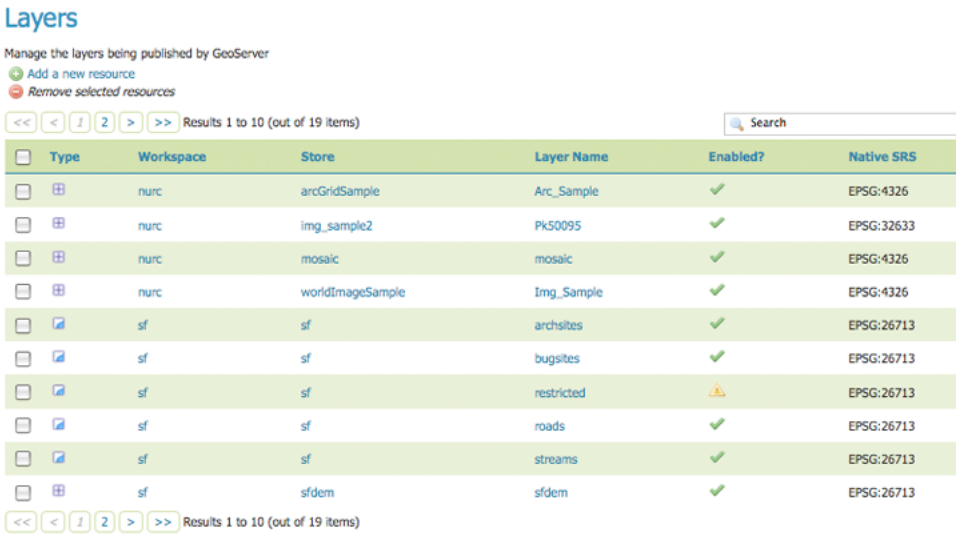


Figure 3.9: *Layers View*

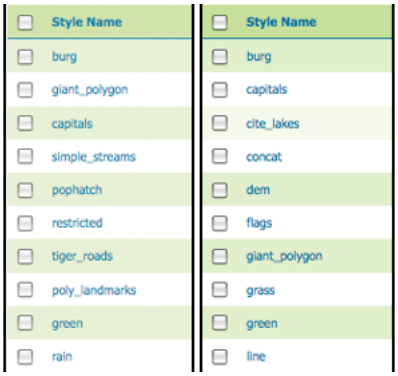


Figure 3.10: *On the left an unsorted column; on the right a sorted column.*

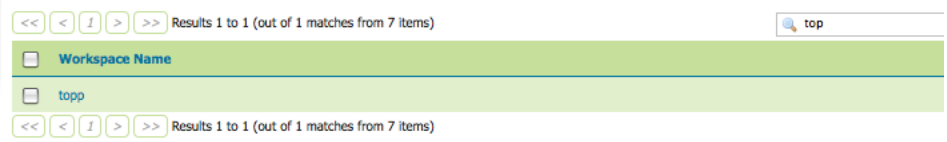


Figure 3.11: *Search results for the query "top".*

Figure 3.12: *Page scroll for data types.*

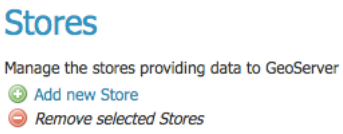


Figure 3.13: *Buttons to add and remove Stores*

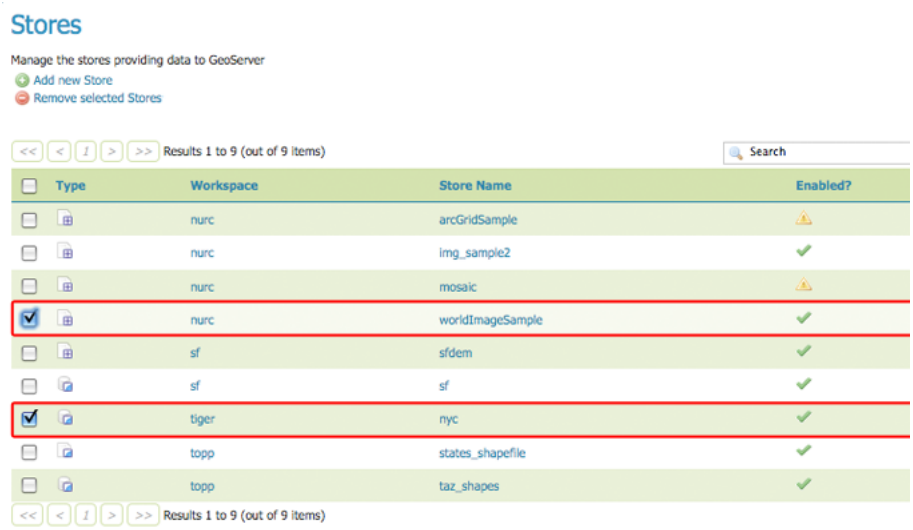


Figure 3.14: *Stores checked for deletion*

3.1.5 Demos

The *Demos* page contains links to example WMS, WCS and WFS requests for GeoServer as well as a link listing all SRS info known to GeoServer. You do not need to be logged into GeoServer to access this page.

Figure 3.15: *Demos page*

3.1.6 Layer Preview

The *Layer Preview* page provides layer previews in various output formats, including the common OpenLayers and KML formats. This page helps to visually verify and explore the configuration of a particular layer.

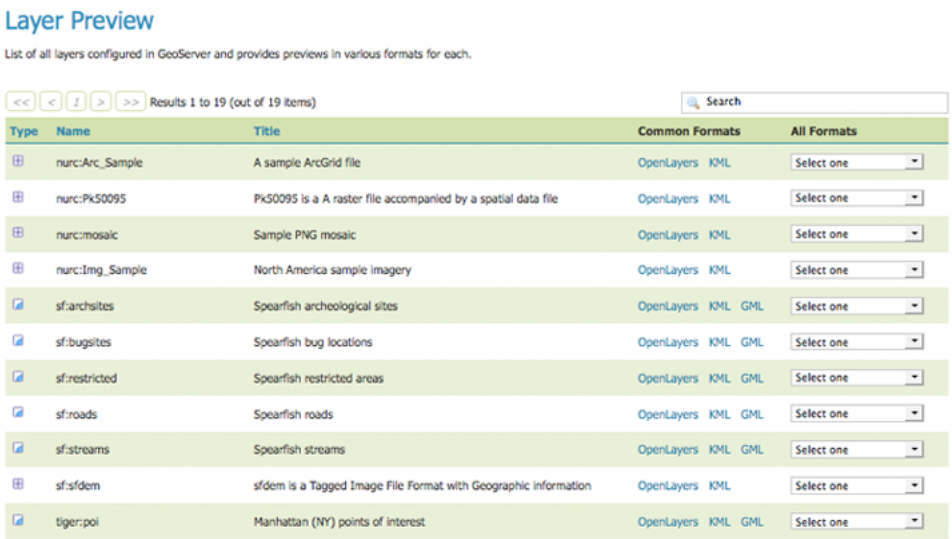


Figure 3.16: *Layer Preview page*

Each layer row consists of a *Type*, *Name*, *Title*, and available formats for viewing. The *Type* column shows an icon indicating the layer datatype. *Name* displays the Workspace and Layer Name of a layer, while *Title* displays the brief description configured in the *Edit Layer Data* panel. *Common Formats* include OpenLayers, KML, and GML where applicable, while the *All Formats* include additional output formats for further use or data sharing.

Figure 3.17: *Single Layer preview row*

3.2 Publishing a Shapefile

This tutorial walks through the steps of publishing a Shapefile with GeoServer.

Note: This tutorial assumes that GeoServer is running at http://localhost:8090/geoserver/web.

3.2.1 Getting Started

1. Download the file `nyc_roads.zip`. This archive contains a Shapefile of roads from New York City that will be used during in this tutorial.
2. Unzip the *nyc_roads.zip*. The extracted folder `nyc_roads` contains the following four files:

```
nyc_roads.shp
nyc_roads.shx
nyc_roads.dbf
nyc_roads.prj
```

#. Move the `nyc_roads` folder into `<GEOSERVER_DATA_DIR>/data`, where `<GEOSERVER_DATA_DIR>` is the root of the GeoServer data directory. If no changes have been made to the GeoServer file structure, the path is `geoserver/data_dir/data/nyc_roads`.

3.2.2 Create a New Workspace

The first step is to create a *workspace* for the Shapefile. A workspace is a container used to group similar layers together.

1. In a web browser navigate to http://localhost:8080/geoserver/web.
2. Log into GeoServer as described in *Logging In*.
3. Navigate to *Data→Workspaces*.

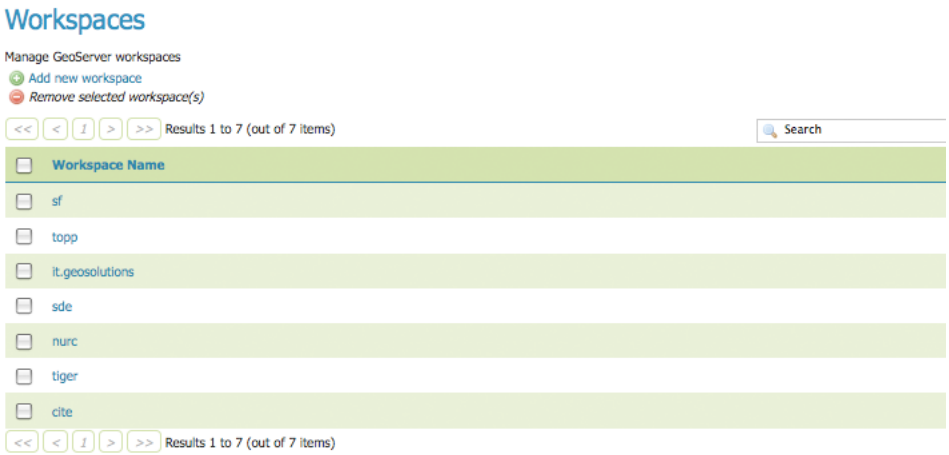


Figure 3.18: *Workspaces page*

4. To create a new workspace click the *Add new workspace* button. You will be prompted to enter a workspace *Name* and *Namespace URI*.

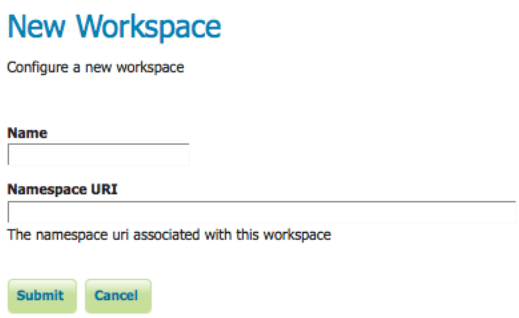


Figure 3.19: *Configure a New Worksapce*

5. Enter the *Name* as `nyc_roads` and the *Namespace URI* as `http://opengeo.org/nyc_roads`. A workspace name is a identifier describing your project. It must not exceed ten characters or contain spaces. A Namespace URI (Uniform Resource Identifier) is typically a URL associated with your project, perhaps with an added trailing identifier indicating the workspace.

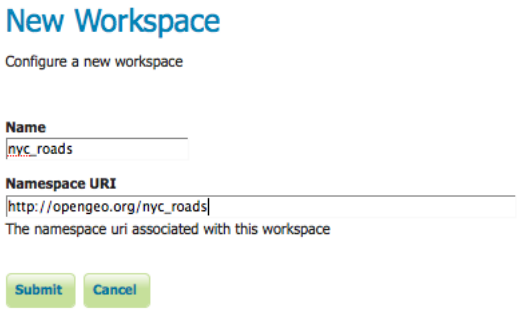


Figure 3.20: *NYC Roads Workspace*

6. Click the *Submit* button. The `nyc_roads` workspace will be added to the *Workspaces* list.

3.2.3 Create a Data Store

1. Navigate to *Data→Stores*.
2. In order to add the nyc_roads Shapefile, you need to create a new Store. Click on the *Add new store* button. You will be redirected to a list of the data sources supported by GeoServer.

3. Select *Shapefile - ESRI(tm) Shapefiles (.shp)*. The *New Vector Data Source* page will display.
4. Begin by configuring the *Basic Store Info*. Select the workspace `nyc_roads` from the drop down menu. Enter the *Data Source Name* as `NYC Roads`. and enter a brief *Description* (such as "Roads in New York City").
5. Under *Connection Parameters* specify the location *URL* of the Shapefile as `file:data/nyc_roads/nyc_roads.shp`.

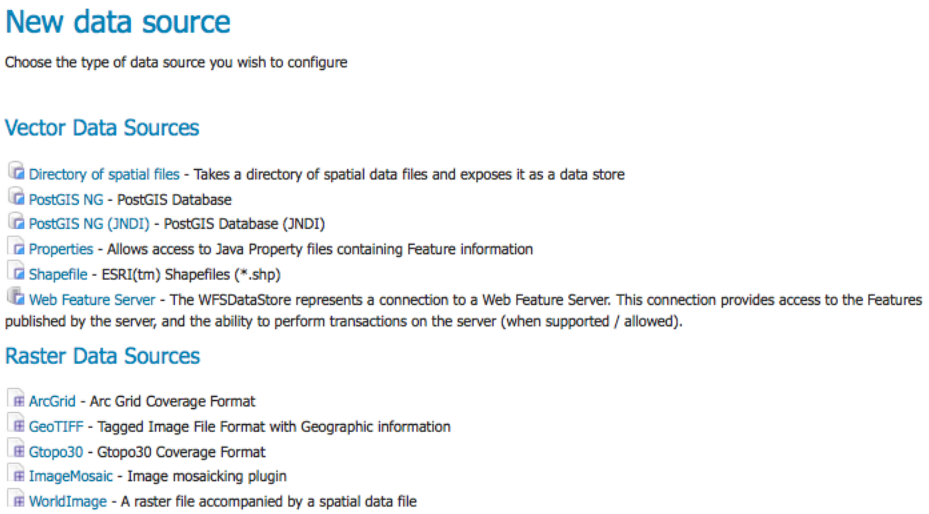


Figure 3.21: *Data Sources*

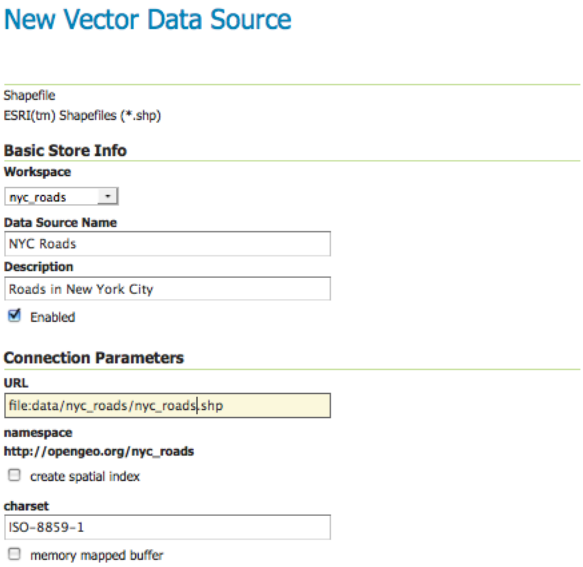


Figure 3.22: *Basic Store Info and Connection Parameters*

6. Click *Save*. You will be redirected to the *New Layer chooser* page in order to configure the `nyc_roads` layer.

3.2.4 Create a Layer

1. On the *New Layer chooser* page, select the layer `nyc_roads`.

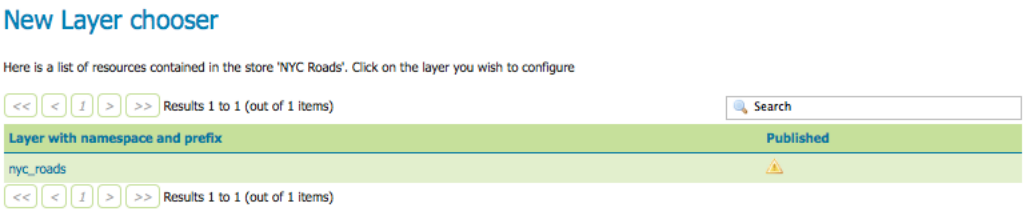


Figure 3.23: *New Layer chooser*

2. The *Edit Layer* page defines the Data and Publishing parameters for a layer. Enter a short *Title* and an *Abstract* for the `nyc_roads` layer.

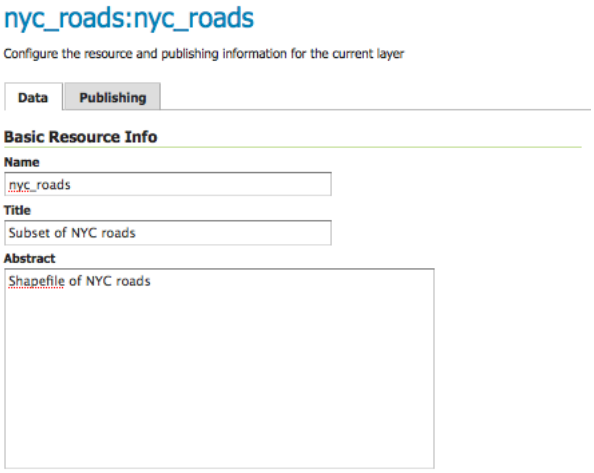


Figure 3.24: *Basic Resource Information*

3. Generate the layer's *bounding boxes* by clicking the *Compute from data* and then *Compute from Native bounds*.

4. Set the layer's style by switching to the *Publishing* tab.
5. Select the *line* style from the *Default Style* drop down list.

6. Finalize the layer configuration by scrolling to the bottom of the page and clicking *Save*.

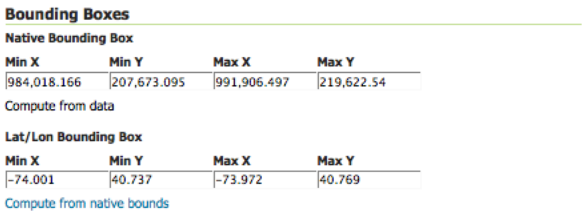


Figure 3.25: *Generate Bounding Boxes*

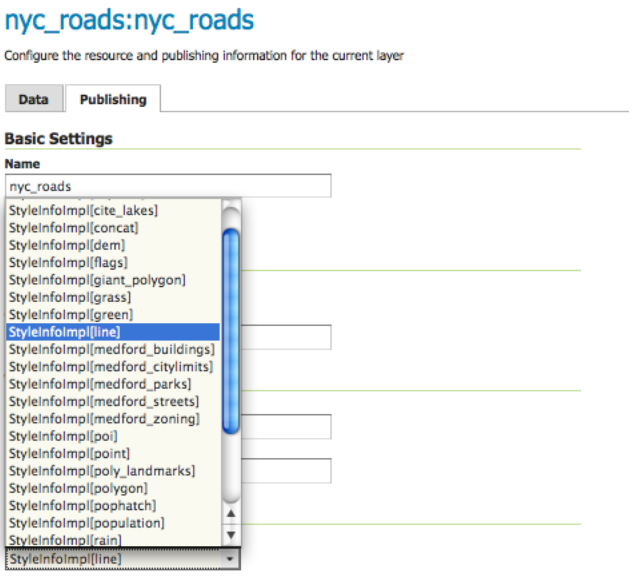


Figure 3.26: *Select Default Style*

3.2.5 Preview the Layer

1. In order to verify that the `nyc_roads` layer is published correctly you can preview the layer. Navigate to the *Layer Preview* screen and find the `nyc_roads:nyc_roads` layer.

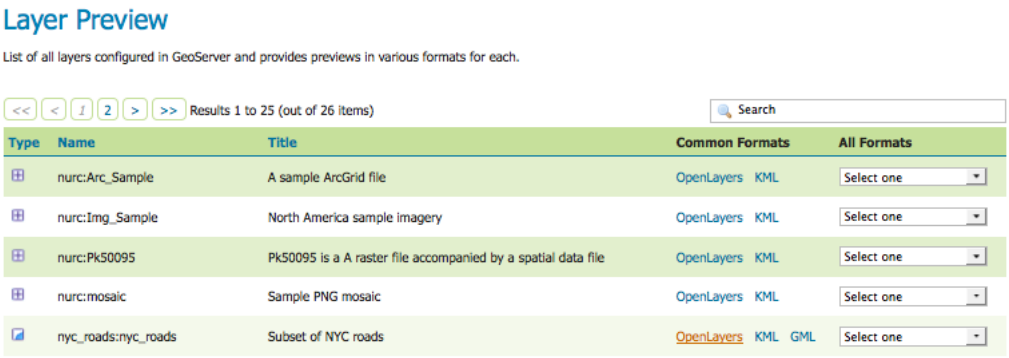


Figure 3.27: *Layer Preview*

2. Click on the *OpenLayers* link in the *Common Formats* column.
3. Success! An OpenLayers map loads in a new page and displays the Shapefile data with the default line style. You can use the Preview Map to zoom and pan around the dataset, as well as display the attributes of features.

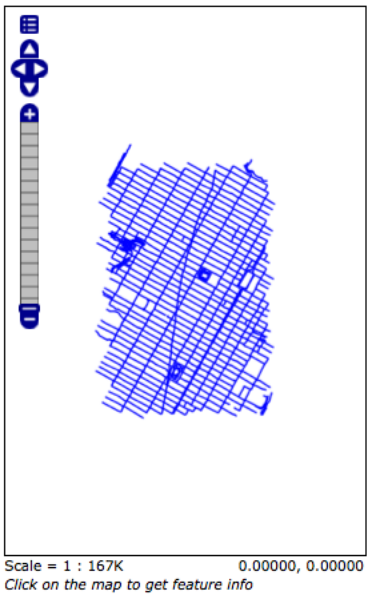


Figure 3.28: *Preview map of nyc_roads*

3.3 Publishing a PostGIS Table

This tutorial walks through the steps of publishing a PostGIS table with GeoServer.

Note: This tutorial assumes that GeoServer is running at http://localhost:8080/geoserver.

Note: This tutorial assumes PostGIS has been previously installed on the system.

3.3.1 Getting Started

1. Download the zip file `nyc_buildings.zip`. It contains a PostGIS dump of a dataset of buildings from New York City that will be used during in this tutorial.
2. Create a PostGIS database called "nyc". This can be done with the following command line:

   ```
   createdb -T template_postgis nyc
   ```

   If the PostGIS install is not set up with the "postgis_template" then the following sequence of commands will perform the equivalent:

   ```
   ...
   ```

3. Unzip `nyc_buildings.zip` to some location on the file system. This will result in the file `nyc_buildings.sql`.
4. Import `nyc_buildings.sql` into the `nyc` database:

   ```
   psql -f nyc_buildings.sql nyc
   ```

3.3.2 Create a Data Store

The first step is to create a *data store* for the PostGIS database "nyc". The data store tells GeoServer how to connect to the database.

1. In a web browser navigate to http://localhost:8080/geoserver.
2. Navigate to *Data→Stores*.

3. Create a new data store by clicking the `PostGIS NG` link.
4. Enter the *Basic Store Info*. Keep the default *Workspace*, and enter the *Data Source Name* as `nyc_buildings` and a brief *Description*.

5. Specify the PostGIS database *Connection Parameters*

`dbtype`	postgisng
`host`	localhost
`post`	5432
`database`	nyc
`schema`	public
`user`	postgres
`passwd`	enter postgres password
`validate connections`	enable with check box

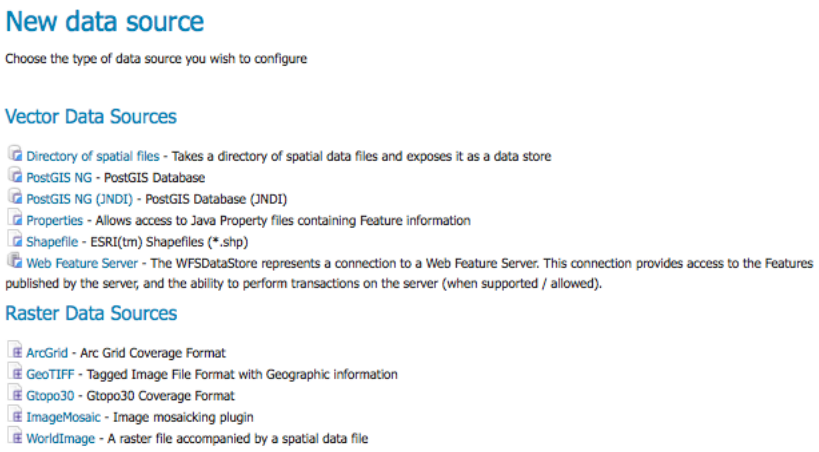


Figure 3.29: *Adding a New Data Source*

Figure 3.30: *Basic Store Info*

Note: The **username** and **password** parameters are specific to the user who created the postgis database. Depending on how PostgreSQL is configured the password parameter may be unnecessary.

---

6. Click *Save*.

3.3.3 Create a Layer

1. Navigate to *Data→Layers*.
2. Click *Add a new resource*.
3. From the *New Layer chooser* drop-down menu, select `cite:nyc_buidings`.
4. On the resulting layer row, select the layer name `nyc_buildings`.
5. The *Edit Layer* page defines the Data and Publishing parameters for a layer. Enter a short *Title* and an *Abstract* for the `nyc_buildings` layer.
6. Generate the layer's *bounding boxes* by clicking the *Compute from data* and then *Compute from Native bounds*.

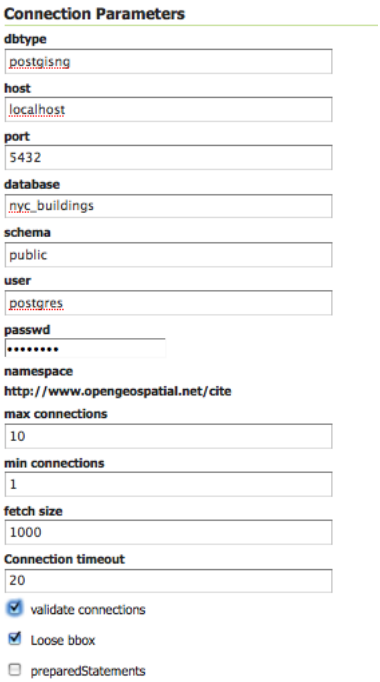


Figure 3.31: *Connection Parameters*

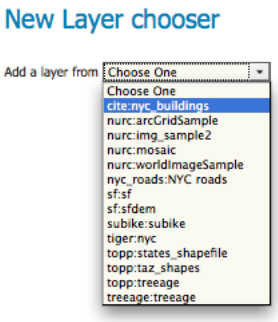


Figure 3.32: *New Layer drop down selection*

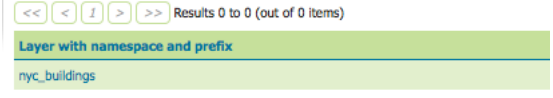


Figure 3.33: *New Layer row*

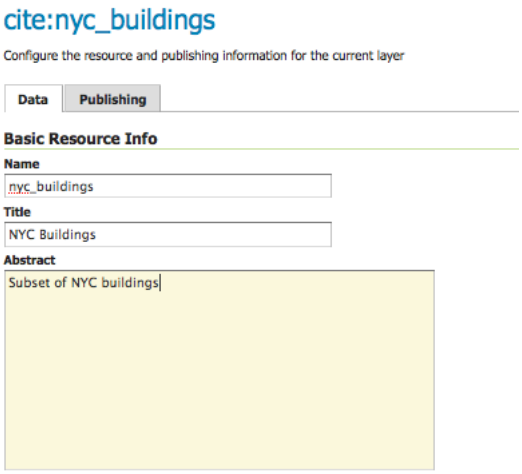


Figure 3.34: *Basic Resource Info*

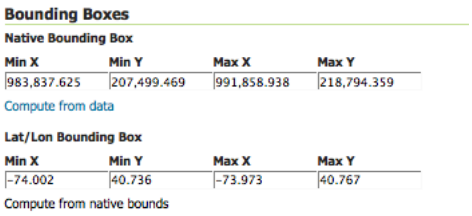


Figure 3.35: *Generate Bounding Boxes*

7. Set the layer's style by switching to the *Publishing* tab.
8. Select the *polygon* style from the *Default Style* drop down list.

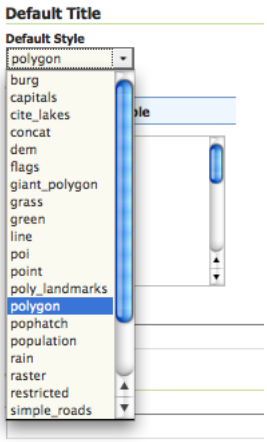


Figure 3.36: *Select Default Style*

9. Finalize the layer configuration by scrolling to the bottom of the page and clicking *Save*.

3.3.4 Preview the Layer

1. In order to verify that the `nyc_buildings` layer is published correctly you can preview the layer. Navigate to the *Layer Preview* screen and find the `cite:nyc_buildings` layer.

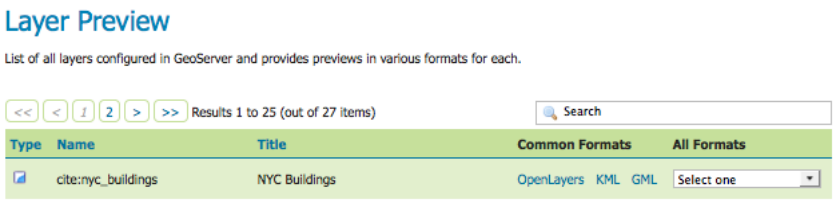


Figure 3.37: *Layer Preview*

2. Click on the *OpenLayers* link in the *Common Formats* column.
3. Success! An OpenLayers map loads in a new page and displays the layer data with the default polygon style. You can use the Preview Map to zoom and pan around the dataset, as well as display the attributes of features.

3.4 Styling a Map

When a new dataset is added to GeoServer the layer for it is usually assigned a very basic style. To properly visualize the data a style specific to that data must be created.

Figure 3.38: *Preview map of nyc_buildings*

This tutorial walks through the steps to create a new style in GeoServer and provides an introduction to the Styled Layer Descriptor (SLD) styling language.

Note: It is assumed that the tutorials *Publishing a Shapefile* and *Publishing a PostGIS Table* have been completed.

3.4.1 Getting started

Before continuing with this tutorial it is *strongly* recommended that the section *Introduction to SLD* be first read.

3.4.2 Creating a new style

CHAPTER 4

GeoServer Data Directory

The GeoServer **data directory** is the location in the file system where GeoServer stores its configuration information. The configuration defines things such as what data is served by GeoServer, where it is stored, and how services such as WFS and WMS interact with and serve the data. The data directory also contains a number of support files used by GeoServer for various purposes.

For production use, it is a good idea to define an external data directory for GeoServer instances, to make it easier to upgrade. To learn how to create a data directory for a GeoServer installation see the *Creating a New Data Directory* section. *Setting the Data Directory* describes how to configure GeoServer to use an existing data directory.

Since GeoServer provides both interactive and programmatic interfaces to manage confiuration information, in general users do not need to know about the internal structure of the data directory. As background, an overview is provided in the *Structure of the Data Directory* section.

4.1 Creating a New Data Directory

The easiest way to create a new data directory is to copy one that comes with a standard GeoServer installation.

If GeoServer is running in **Standalone** mode the data directory is located at `<installation root>/data_dir`.

Note: On Windows systems the `<installation root>` is located at `C:\Program Files\GeoServer <VERSION>`.

If GeoServer is running in **Web Archive** mode inside of a servlet container, the data directory is located at `<web application root>/data`.

Once the data directory has been found copy it to a new external location. To point a GeoServer instance at the new data directory proceed to the next section *Setting the Data Directory*.

4.2 Setting the Data Directory

Setting the location of the GeoServer data directory is dependent on the type of GeoServer installation. Follow the instructions below specific to the target platform.

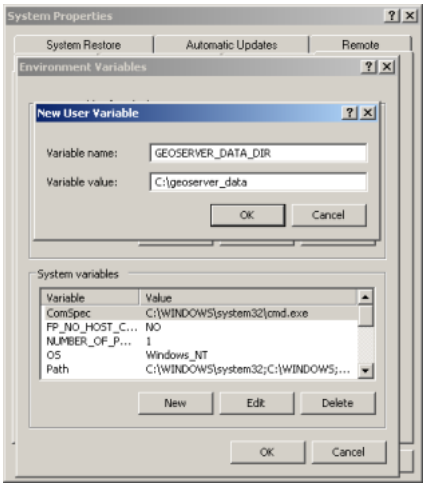
Note: If the location of the GeoServer data directory is not set explicitly, the directory `data_dir` under the root of the GeoServer installation is used.

4.2.1 Windows

On Windows platforms the location of the GeoServer data directory is controlled by the `GEOSERVER_DATA_DIR` environment variable. The method of setting this variable depends on the Windows version.

Windows XP

1. From the Desktop or Start Menu right-click the `My Computer` icon and select `Properties`.
2. On the resulting dialog select the `Advanced` tab and click the `Environment Variables` button.
3. Click the `New` button and create a environment variable called `GEOSERVER_DATA_DIR` and set it to the desired location.



Windows Vista

To be documented.

4.2.2 Linux

On Linux platforms the location of the GeoServer data directory is controlled by the `GEOSERVER_DATA_DIR` environment variable. Setting the variable can be achieved with the following command (in a bash shell):

```
% export GEOSERVER_DATA_DIR=/var/lib/geoserver_data
```

Place the command in the `.bash_profile` or `.bashrc` file (again assuming a bash shell). Ensure that this done for the user running GeoServer.

4.2.3 Mac OS X

Binary Install

For the binary install of GeoServer on Mac OS X, the data directory is set in the same way as for Linux.

Mac OS X Install

For the Mac OS X install, set the `GEOSERVER_DATA_DIR` environment variable to the desired directory location. See this page for details on how to set an environment variable in Mac OS X

4.2.4 Web Archive

When running a GeoServer WAR inside a servlet container the data directory can be specified in a number of ways. The recommended method is to set a **servlet context parameter**. An alternative is to set a **Java system property**.

Servlet context parameter

To specify the data directory using a servlet context parameter, create the following `<context-param>` element in the `WEB-INF/web.xml` file for the GeoServer application:

```
<web-app>
  ...
  <context-param>
    <param-name>GEOSERVER_DATA_DIR</param-name>
    <param-value>/var/lib/geoserver_data</param-value>
  </context-param>
  ...
</web-app>
```

Java system property

It is also possible to specify the data directory location with a Java system property. This method can be useful during upgrades, as it avoids the need to set the data directory after every upgrade.

> **Warning:** Using a Java system property will typically set the property for all applications running in the servlet container, not just GeoServer.

The method of setting the Java system property is dependent on the servlet container:

For **Tomcat**:

Edit the file `bin/setclasspath.sh` under the root of the Tomcat installation. Specify the `GEOSERVER_DATA_DIR` system property by setting the `CATALINA_OPTS` variable:

```
CATALINA_OPTS="-DGEOSERVER_DATA_DIR=/var/lib/geoserver_data"
```

For **Glassfish**:

Edit the file `domains/<<domain>>/config/domain.xml` under the root of the Glassfish installation, where `<<domain>>` refers to the domain that the GeoServer web application is deployed under. Add a `<jvm-options>` element inside the `<java-config>` element:

```
...
<java-config>
   ...
  <jvm-options>-DGEOSERVER_DATA_DIR=/var/lib/geoserver_data</jvm-options>
</java-config>
...
```

4.3 Structure of the Data Directory

This section gives an overview of the structure and contents of the GeoServer data directory.

This is not intended to be a complete reference to the GeoServer configuration information, since generally the data directory configuration files should not be accessed directly. Instead, the *Web Administration Interface* can be used to view and modify the configuration manually, and for programmatic access and manipulation the *REST configuration* API should be used.

The directories that do contain user-modifiable content are: `logs`, `palettes`, `templates`, `user-projection`, and `www`.

The following figure shows the structure of the GeoServer data directory:

```
<data_directory>/

   global.xml
   logging.xml
   wms.xml
   wfs.xml
   wcs.xml

   data/
   demo/
   geosearch/
   gwc/
   layergroups/
   logs/
   palettes/
   plugIns/
   security/
   styles/
   templates/
   user_projections/
   workspaces/
     |
     +- workspace dirs...
        |
        +- datastore dirs...
           |
           +- layer dirs...
   www/
```

4.3.1 The .xml files

The top-level `.xml` files contain information about the services and various global options for the server instance.

File	Description
`global.xml`	Contains settings common to all services, such as contact information, JAI settings, character sets and verbosity.
`logging.xml`	Specifies logging parameters, such as logging level, logfile location, and whether to log to stdout.
`wcs.xml`	Contains the service metadata and various settings for the WCS service.
`wfs.xml`	Contains the service metadata and various settings for the WFS service.
`wms.xml`	Contains the service metadata and various settings for the WMS service.

4.3.2 workspaces

The `workspaces` directory contain metadata about the layers published by GeoServer. It contains a directory for each defined **workspace**. Each workspace directory contains directories for the **datastores** defined in it. Each datastore directory contains directories for the **layers** defined for the datastore. Each layer directory contains a `layer.xml` file, and either a `coverage.xml` or a `featuretype.xml` file depending on whether the layer represents a *raster* or *vector* dataset.

4.3.3 data

The `data` directory can be used to store file-based geospatial datasets being served as layers. (This should not be confused with the main "GeoServer data directory".) This directory is commonly used to store shapefiles and raster files, but can be used for any data that is file-based.

The main benefit of storing data files under the `data` directory is portability. Consider a shapefile stored external to the data directory at a location `C:\gis_data\foo.shp`. The `datastore` entry in `catalog.xml` for this shapefile would look like the following:

```
<datastore id="foo_shapefile">
   <connectionParams>
     <parameter name="url" value="file://C:/gis_data/foo.shp" />
   </connectionParams>
 </datastore>
```

Now consider trying to port this data directory to another host running GeoServer. The location `C:\gis_data\foo.shp` probably does not exist on the second host. So either the file must be copied to this location on the new host, or `catalog.xml` must be changed to reflect a new location.

This problem can be avoided by storing `foo.shp` in the `data` directory. In this case the `datastore` entry in `catalog.xml` becomes:

```
<datastore id="foo_shapefile">
  <connectionParams>
    <parameter name="url" value="file:data/foo.shp"/>
  </connectionParams>
</datastore>
```

The `value` attribute is rewritten to be relative to the `data` directory. This location independence allows the entire data directory to be copied to a new host and used directly with no additional changes.

4.3.4 demo

The `demo` directory contains files which define the *sample requests* available in the *Sample Request Tool* (http://localhost:8080/geoserver/demoRequest.do). See the *Demos* page for more information.

4.3.5 geosearch

The `geosearch` directory contains information for regionation of KML files.

4.3.6 gwc

The `gwc` directory holds the cache created by the embedded GeoWebCache service.

4.3.7 layergroups

The `layergroups` directory contains configuration information for the defined layergroups.

4.3.8 logs

The `logs` directory contains configuration information for logging profiles, and the default `geoserver.log` log file. See also *Advanced log configuration*.

4.3.9 palettes

The `palettes` directory is used to store pre-computed **Image Palettes**. Image palettes are used by the GeoServer WMS as way to reduce the size of produced images while maintaining image quality. See also *Paletted Images*.

4.3.10 security

The `security` directory contains the files used to configure the GeoServer security subsystem. This includes a set of property files which define *access roles*, along with the services and data each role is authorized to access. See the *Security* section for more information.

4.3.11 styles

The `styles` directory contains Styled Layer Descriptor (SLD) files which contain styling information used by the GeoServer WMS. For each file in this directory there is a corresponding entry in `catalog.xml`:

```
<style id="point_style" file="default_point.sld"/>
```

See the *Styling* section for more information about styling and SLD .

4.3.12 templates

The `templates` directory contains files used by the GeoServer **templating** subsystem. Templates are used to customize the output of various GeoServer operations. See also *Freemarker Templates*.

4.3.13 user_projections

The `user_projections` directory contains a file called `epsg.properties` which is used to define custom spatial reference systems that are not part of the official EPSG database. See also *Custom CRS Definitions*.

4.3.14 www

The `www` directory is used to allow GeoServer to serve files like a regular web server. The contents of this directory are served at `http://<host:port>/geoserver/www`. While not a replacement for a full blown web server, this can be useful for serving client-side mapping applications. See also *Serving Static Files*.

4.4 Migrating a Data Directory between different versions

4.4.1 Minor and major version numbers

There should generally be no problems or issues migrating data directories between major and minor versions of GeoServer (i.e. from 2.0.0 to 2.0.1 and vice versa, or from 1.6.x to 1.7.x and vice versa).

4.4.2 Migrating between GeoServer 1.7.x and 2.0.x

When using GeoServer 2.0.x with a data directory from the 1.7.x branch, modifications will occur to the directory **immediately** that will **make the data directory incompatible with 1.7.x**! Below is a list of changes made to the data directory.

Files and directories added

```
wfs.xml
wcs.xml
wms.xml
logging.xml
global.xml
workspaces/*
layergroups/*
styles/*.xml
```

Files renamed

- `catalog.xml` renamed to `catalog.xml.old`
- `services.xml` renamed to `services.xml.old`

4.4.3 Reverting from GeoServer 2.0.x to 1.7.x

In order to revert the directory to be compatible with 1.7.x again:

1. Stop GeoServer.
2. Delete the following files and directories:

   ```
   wfs.xml
   wcs.xml
   wms.xml
   logging.xml
   global.xml
   workspaces/*
   layergroups/*
   styles/*.xml
   ```

3. Rename `catalog.xml.old` to `catalog.xml`.
4. Rename `services.xml.old` to `services.xml`.

4.4.4 Migrating between GeoServer 2.1.x and 2.2.x

The security changes that ship with GeoServer 2.2 require modifications to the `security` directory of the GeoServer data directory.

Files and directories added

```
security/*.xml
security/masterpw.*
security/geoserver.jceks
security/auth/*
security/filter/*
security/masterpw/*
security/pwpolicy/*
security/role/*
security/usergroup/*
```

Files renamed

- `security/users.properties` renamed to `security/users.properties.old`

4.4.5 Reverting from GeoServer 2.2.x and 2.1.x

In order to restore the GeoServer 2.1 configuration:

1. Stop GeoServer.
2. Rename `users.properties.old` to `users.properties`.
3. Additionally (although not mandatory) delete the following files and directories:

```
security/
  config.xml
  geoserver.jceks
  masterpw.xml
  masterpw.digest
  masterpw.info
  auth/
  filter/
  masterpw/
  pwpolicy/
  role/
  usergroup/
```

4.4.6 Migrating between GeoServer 2.2.x and 2.3.x

The security improvements that ship with GeoServer 2.3 require modifications to the `security` directory of the GeoServer data directory.

Files and directories added

```
security/filter/roleFilter/config.xml
```

Files modified

```
security/filter/formLogout/config.xml
security/config.xml
```

Backup files

```
security/filter/formLogout/config.xml.2.2.x
security/config.xml.2.2.x
```

4.4.7 Reverting from GeoServer 2.3.x

In order to restore the GeoServer 2.2 configuration:

1. Stop GeoServer.
2. Copy `security/config.xml.2.2.x` to `security/config.xml`.
3. Copy `security/filter/formLogout/config.xml.2.2.x` to `security/filter/formLogout/config.xml`
4. Additionally (although not mandatory) delete the following files and directories:

```
security/
  filter/
    roleFilter/
       config.xml
    formLogout/
       config.xml.2.2.x
  config.xml.2.2.x
```

CHAPTER 5

Web Administration Interface

The Web Administration Interface is a web-based tool for configuring all aspects of GeoServer, from adding data to changing service settings. In a default GeoServer installation, this interface is accessed via a web browser at http://localhost:8080/geoserver/web. However, this URL may vary depending on your local installation.

5.1 Interface basics

This section will introduce the basic concepts of the web administration interface (generally abbreviated to "web admin" .)

5.1.1 Welcome Page

For most installations, GeoServer will start a web server on localhost at port 8080, accessible at the following URL:

```
http://localhost:8080/geoserver/web
```

Note: This URL is dependent on your installation of GeoServer. When using the WAR installation, for example, the URL will be dependent on your container setup.

When correctly configured, a welcome page will open in your browser.

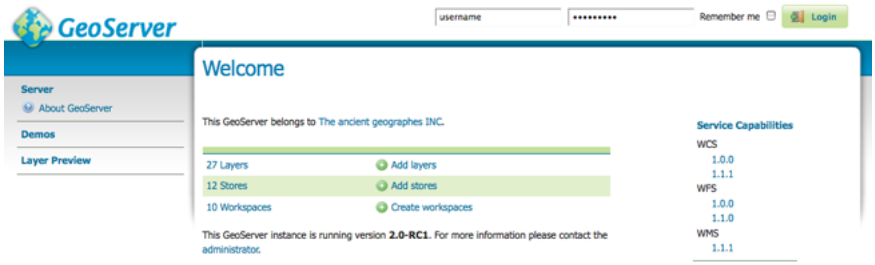


Figure 5.1: *Welcome Page*

The welcome page contains links to various areas of the GeoServer configuration. The *About GeoServer* section in the *Server* menu provides external links to the GeoServer documentation, homepage, and bug

tracker. The page also provides login access to the geoserver console. This security measure prevents unauthorized users from making changes to your GeoServer configuration. The default username and password is `admin` and `geoserver`. These can be changed only by editing the `security/users.properties` file in the *GeoServer Data Directory*.

Figure 5.2: *Login*

Regardless of authorization access, the web admin menu links to the *Demo* and *Layer Preview* portion of the console. The *Demos* page contains links to various information pages, while the *Layer Preview* page provides spatial data in various output formats.

When logged on, additional options will be presented.

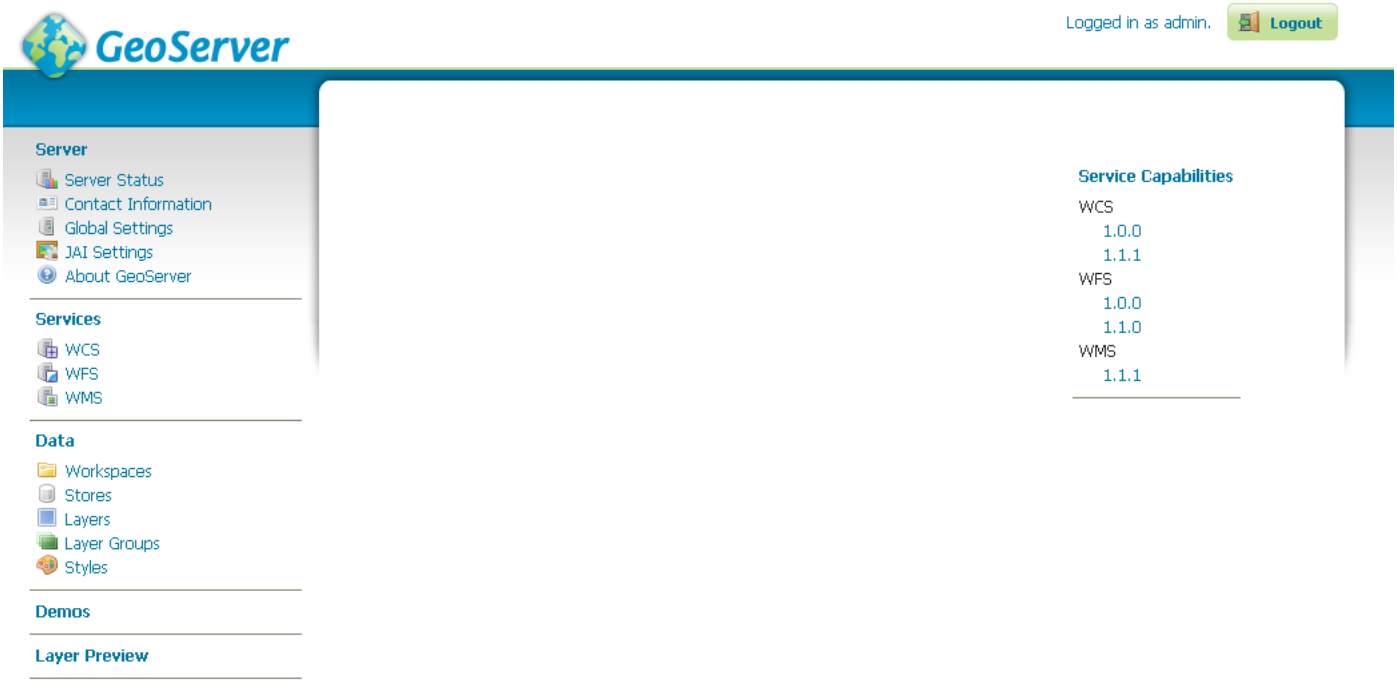


Figure 5.3: *Additional options when logged in*

Geoserver Web Coverage Service (WCS), Web Feature Service (WFS), and Web Map Service (WMS) configuration specifications can be accessed from this welcome page as well. For further information, please see the section on *Services*.

5.2 Server

The Server section of the *Web Administration Interface* provides access to GeoServer configuration and diagnostic tools, which may be useful for debugging.

5.2.1 Status

The Server Status page provides a summary of server configuration parameters and run-time status. It provides a useful diagnostic tool in a testing environment.

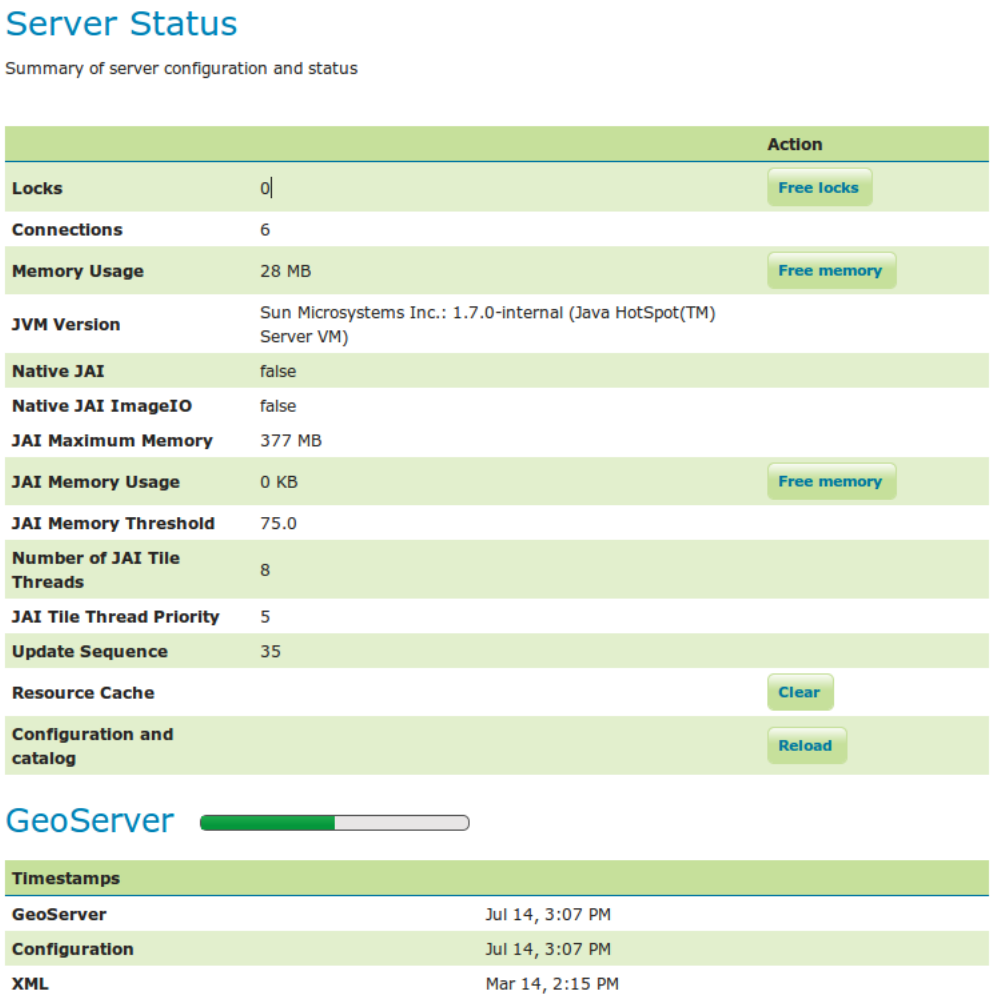


Figure 5.4: *Status Page*

Status Field Descriptions

The following table describes the current status indicators.

Option	Description
Locks	A WFS has the ability to lock features to prevent more than one person from updating the feature at one time. If data is locked, edits can be performed by a single WFS editor. When the edits are posted, the locks are released and features can be edited by other WFS editors. A zero in the locks field means all locks are released. If locks is non-zero, then pressing "free locks," releases all feature locks currently help by the server, and updates the field value to zero.
Connections	Refers to the numbers of vector stores, in the above case 4, that were able to connect.
Memory Usage	The amount of memory current used by GeoServer. In the above example, 55.32 MB of memory is being used. Clicking on the "Free Memory" button, cleans up memory marked for deletion by running the garbage collector.
JVM Version	Denotes which version of the JVM (Java Virtual Machine) is been used to power the server. Here the JVM is Apple Inc.: 1.5.0_16.
Native JAI	GeoServer uses Java Advanced Imaging (JAI) framework for image rendering and coverage manipulation. When properly installed (true), JAI makes WCS and WMS performance faster and more efficient.
Native JAI ImageIO	GeoServer uses JAI Image IO (JAI) framework for raster data loading and image encoding. When properly installed (true), JAI Image I/O makes WCS and WMS performance faster and more efficient.
JAI Maximum Memory	Expresses in bytes the amount of memory available for tile cache, in this case 33325056 bytes. The JAI Maximum Memory value must be between 0.0 and {0}
JAI Memory Usage	Run-time amount of memory is used for the tile cache. Clicking on the "Free Memory" button, clears available JAI memory by running the tile cache flushing.
JAI Memory Threshold	Refers to the percentage, e.g. 75, of cache memory to retain during tile removal. JAI Memory Threshold value must be between 0.0 and 100.
Number of JAI Tile Threads	The number of parallel threads used by to scheduler to handle tiles
JAI Tile Thread Priority	Schedules the global tile scheduler priority. The priority value is defaults to 5, and must fall between 1 and 10.
Update Sequence	Refers to the number of times (60) the server configuration has been modified
Resource cache	GeoServer does not cache data, but it does cache connection to stores, feature type definitions, external graphics, font definitions and CRS definitions as well. The "Clear" button forces those caches empty and makes GeoServer reopen the stores and re-read image and font information, as well as the custom CRS definitions stored in *${GEOSERVER_DATA_DIR}/user_projections/epsg.properties*.
Configuration and catalog	GeoServer keeps in memory all of its configuration data. If for any reason that configuration information has become stale (e.g., an external utility has modified the configuration on disk) the "Reload" button will force GeoServer to reload all of its configuration from disk.

Timestamps Field Descriptions

Option	Description
GeoServer	Currently a placeholder. Refers to the day and time of current GeoServer install.
Configuration	Currently a placeholder. Refers to the day and time of last configuration change.
XML	Currently a placeholder.

5.2.2 Contact Information

The Contact Information is used in the Capabilities document of the WMS server, and is publicly accessible. Please complete this form with the relevant information.

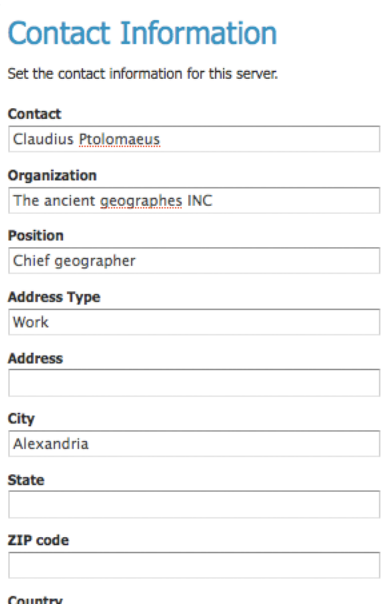


Figure 5.5: *Contact Page*

Contact Information Fields

Field	Description
Contact	Contact information for webmaster
Organization	Name of the organization with which the contact is affiliated
Position	Position of the contact within their organization
Address Type	Type of address specified, such as postal
Address	Actual street address
City	City of the address
State	State or province of the address
Zip code	Postal code for the address
Country	Country of the address
Telephone	Contact phone number
Fax	Contact Fax number
Email	Contact email address

5.2.3 Global Settings

The Global Setting page configures messaging, logging, character, and proxy settings for the entire server.

Verbose Messages

Verbose Messages, when enabled, will cause GeoServer to return XML with newlines and indents. Because such XML responses contain a larger amount of data, and in turn requires a larger amount of bandwidth, it is recommended to use this option only for testing purposes.

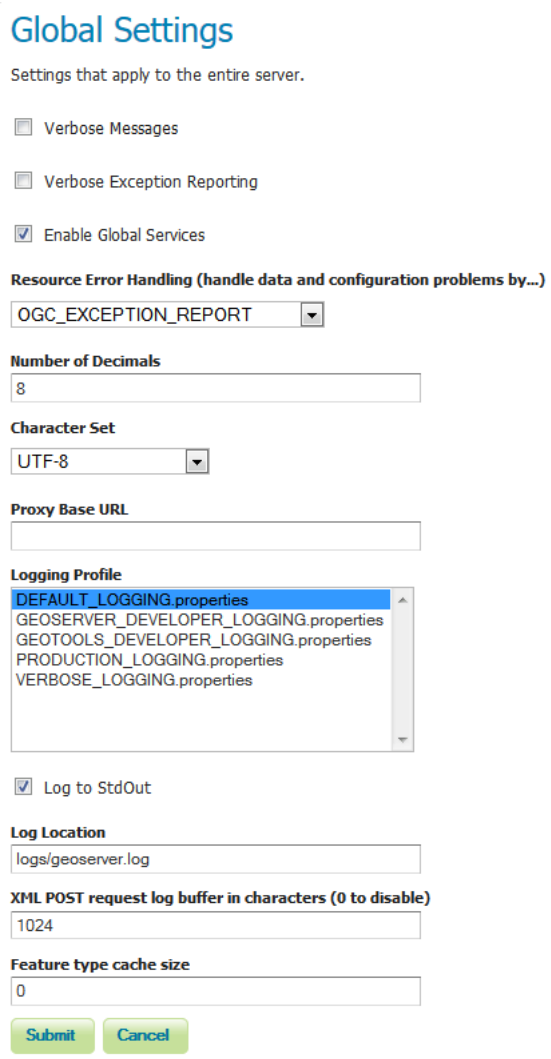


Figure 5.6: *Global Settings Page*

Verbose Exception Reporting

Verbose Exception Reporting returns service exceptions with full Java stack traces. It writes to the GeoServer log file and offers one of the most useful configuration options for debugging. When disabled, GeoServer returns single-line error messages.

Enable Global Services

When enabled, allows access to both global services and *virtual services*. When disabled, clients will only be able to access virtual services. Disabling is useful if GeoServer is hosting a large amount of layers and you want to ensure that client always request limited layer lists. Disabling is also useful for security reasons.

Resource Error Handling

This setting determines how GeoServer will respond when a layer becomes inaccessible for some reason. By default, when a layer has an error (for example, when the default style for the layer is deleted), a service exception is printed as part of the capabilities document, making the document invalid. For clients that rely on a valid capabilities document, this can effectively make a GeoServer appear to be "offline".

An administrator may prefer to configure GeoServer to simply omit the problem layer from the capabilities document, thus retaining the document integrity and allowing clients to connect to other published layers.

There are two options:

OGC_EXCEPTION_REPORT: This is the default behavior. Any layer errors will show up as Service Exceptions in the capabilities document, making it invalid.

SKIP_MISCONFIGURED_LAYERS: With this setting, GeoServer will elect simply to not describe the problem layer at all, removing it from the capabilities document, and preserving the integrity of the rest of the document. Note that having a layer "disappear" may cause other errors in client functionality.

Number of Decimals

Refers to the number of decimal places returned in a GetFeature response. Also useful in optimizing bandwidth. Default is **8**.

Character Set

Specifies the global character encoding that will be used in XML responses. Default is **UTF-8**, which is recommended for most users. A full list of supported character sets is available on the IANA Charset Registry.

Proxy Base URL

GeoServer can have the capabilities documents report a proxy properly. The Proxy Base URL field is the base URL seen beyond a reverse proxy.

Logging Profile

Logging Profile corresponds to a log4j configuration file in the GeoServer data directory. (Apache log4j is a Java-based logging utility.) By default, there are five logging profiles in GeoServer; additional customized profiles can be added by editing the log4j file.

There are six logging levels used in the log itself. They range from the least serious TRACE, through DEBUG, INFO, WARN, ERROR and finally the most serious, FATAL. The GeoServer logging profiles combine logging levels with specific server operations. The five pre-built logging profiles available on the global settings page are:

1. **Default Logging** (`DEFAULT_LOGGING`)—Provides a good mix of detail without being VERBOSE. Default logging enables INFO on all GeoTools and GeoServer levels, except certain (chatty) GeoTools packages which require WARN.
2. **GeoServer Developer Logging** (`GEOSERVER_DEVELOPER_LOGGING`)-A verbose logging profile that includes DEBUG information on GeoServer and VFNY. This developer profile is recommended for active debugging of GeoServer.
3. **GeoTools Developer Logging** (`GEOTOOLS_DEVELOPER_LOGGING`)—A verbose logging profile that includes DEBUG information only on GeoTools. This developer profile is recommended for active debugging of GeoTools.
4. **Production Logging** (`PRODUCTION_LOGGING`) is the most minimal logging profile, with only WARN enabled on all GeoTools and GeoServer levels. With such production level logging, only problems are written to the log files.
5. **Verbose Logging** (`VERBOSE_LOGGING`)—Provides more detail by enabling DEBUG level logging on GeoTools, GeoServer, and VFNY.

Log to StdOut

Standard output (StdOut) determines where a program writes its output data. In GeoServer, the Log to StdOut setting enables logging to the text terminal that initiated the program. If you are running GeoServer in a large J2EE container, you might not want your container-wide logs filled with GeoServer information. Clearing this option will suppress most GeoServer logging, with only FATAL exceptions still output to the console log.

Log Location

Sets the written output location for the logs. A log location may be a directory or a file, and can be specified as an absolute path (e.g., `C:\GeoServer\GeoServer.log`) or a relative one (for example, `GeoServer.log`). Relative paths are relative to the GeoServer data directory. Default is `logs/geoserver.log`.

XML POST request log buffer

In more verbose logging levels, GeoServer will log the body of XML (and other format) POST requests. It will only log the initial part of the request though, since it has to store (buffer) everything that gets logged for use in the parts of GeoServer that use it normally. This setting sets the size of this buffer, in characters. A setting of **0** will disable the log buffer.

Feature type cache size

GeoServer can cache datastore connections and schemas in memory for performance reasons. The cache size should generally be greater than the number of distinct featuretypes that are expected to be accessed simultaneously. If possible, make this value larger than the total number of featuretypes on the server, but a setting too high may produce out-of-memory errors.

5.2.4 Coverage Access settings

The Coverage Access Settings page in the Server menu in the *Web Administration Interface* provides configuration options to customize thread pool executors and ImageIO caching memory.

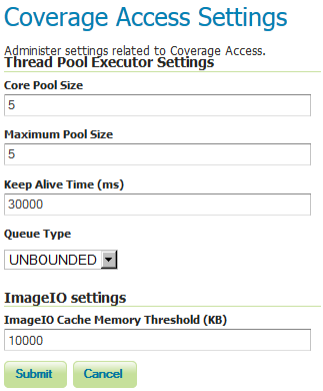


Figure 5.7: *Coverage Access Settings*

Thread Pool Executor Settings

The imageMosaic reader may load, in parallel, different files that make up the mosaic by means of a ThreadPoolExecutor . A global ThreadPoolExecutor instance is shared by all the readers supporting and using concurrent reads. This section of the Coverage Access Settings administration page allows to configure the parameters of the executor.

Core Pool Size—Sets the core pool size of the thread pool executor. A positive integer must be specified.

Maximum Pool Size—Sets the maximum pool size of the thread pool executor. A positive integer must be specified.

Keep Alive Time—Sets the time to be wait by the executor before terminating an idle thread in case there are more threads than *corePoolSize*.

Queue Type—The executor service uses a BlockingQueue to manage submitted tasks. Using an *unbounded* queue is recommended which allows to queue all the pending requests with no limits (Unbounded). With a *direct* type, incoming requests will be rejected when there are already *maximumPoolSize* busy threads.

Note: If a new task is submitted to the list of tasks to be executed, and less than *corePoolSize* threads are running, a new thread is created to handle the request. Incoming tasks are queued in case *corePoolSize* or more threads are running.

Note: If a request can't be queued or there are less than *corePoolSize* threads running, a new thread is created unless this would exceed *maximumPoolSize*.

Note: If the pool currently has more than *corePoolSize* threads, excess threads will be terminated if they have been idle for more than the *keepAliveTime*.

Note: If a new task is submitted to the list of tasks to be executed and there are more than *corePoolSize* but less than *maximumPoolSize* threads running, a new thread will be created only if the queue is full. This means that when using an *Unbounded* queue, no more threads than *corePoolSize* will be running and *keepAliveTime* has no influence.

Note: If *corePoolSize* and *maximumPoolSize* are the same, a fixed-size thread pool is used.

ImageIO Settings

WMS requests usually produce relatively small images whilst WCS requests may frequently deal with bigger datasets. Caching the image in memory before encoding it may be helpful when the size of the image isn't too big. For a huge image (as one produced by a big WCS request) it would be better instead caching through a temporary file with respect to caching in memory. This section allows to specify a threshold image size to let GeoServer decide whether to use a MemoryCacheImageOutputStream or FileCacheImageOutputStream when encoding the images.

ImageIO Cache Memory Threshold—Sets the threshold size (expressed in KiloBytes) which will made GeoServer choose between file cache vs memory based cache. If the estimated size of the image to be encoded is smaller than the threshold value, a *MemoryCacheImageOutputStream* will be used resulting into caching the image in memory. If the estimated size of the image to be encoded is greater than the threshold value, a *FileCacheImageOutputStream* will be used.

5.2.5 JAI

Java Advanced Imaging (JAI) is an image manipulation library built by Sun Microsystems and distributed with an open source license. JAI Image I/O Tools provides reader, writer, and stream plug-ins for the standard Java Image I/O Framework. Several JAI parameters, used by both WMS and WCS operations, can be configured in the JAI Settings page.

Memory & Tiling

When supporting large images it is efficient to work on image subsets without loading everything to memory. A widely used approach is tiling which basically builds a tessellation of the original image so that image data can be read in parts rather than whole. Since very often processing one tile involves surrounding tiles, tiling needs to be accompanied by a tile-caching mechanism. The following JAI parameters allow you to manage the JAI cache mechanism for optimized performance.

Memory Capacity—For memory allocation for tiles, JAI provides an interface called TileCache. Memory Capacity sets the global JAI TileCache as a percentage of the available heap. A number between 0 and 1 exclusive. If the Memory Capacity is smaller than the current capacity, the tiles in the cache are flushed to achieve the desired settings. If you set a large amount of memory for the tile cache, interactive operations are faster but the tile cache fills up very quickly. If you set a low amount of memory for the tile cache, the performance degrades.

Memory Threshold—Sets the global JAI TileCache Memory threshold. Refers to the fractional amount of cache memory to retain during tile removal. JAI Memory Threshold value must be between 0.0 and 1.0. The Memory Threshold visible on the *Status* page.

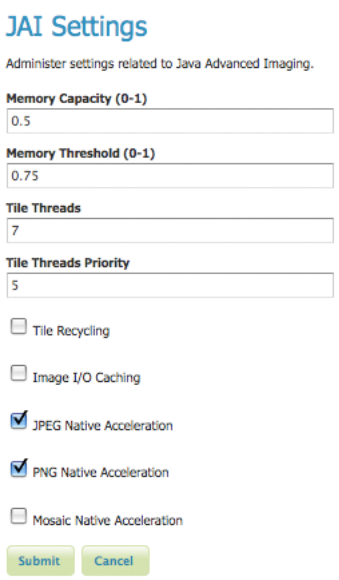


Figure 5.8: *JAI Settings*

Tile Threads—JAI utilizes a TileScheduler for tile calculation. Tile computation may make use of multi-threading for improved performance. The Tile Threads parameter sets the TileScheduler, indicating the number of threads to be used when loading tiles.

Tile Threads Priority—Sets the global JAI Tile Scheduler thread priorities. Values range from 1 (Min) to 10 (Max), with default priority set to 5 (Normal).

Tile Recycling—Enable/Disable JAI Cache Tile Recycling. If selected, Tile Recycling allows JAI to re-use already loaded tiles, with vital capability for performances.

Native Acceleration—To improve the computation speed of image processing applications, the JAI comes with both Java Code and native code for many platform. If the Java Virtual Machine (JVM) finds the native code, then that will be used. If the native code is not available, the Java code will be used. As such, the JAI package is able to provide optimized implementations for different platforms that can take advantage of each platform's capabilities.

JPEG Native Acceleration—Enables/disable JAI JPEG Native Acceleration. When selected, enables JPEG native code, which may speed performance, but compromise security and crash protection.

PNG Native Acceleration—Enables/disables JAI PNG Native Acceleration. When selected, enables PNG native code, which may speed performance, but compromise security and crash protection.

Mosaic Native Acceleration—To reduce the overhead of handling them, large data sets are often split into smaller chunks and then combined to create an image mosaic. An example of this is aerial imagery which is usually comprises thousands of small images at very high resolution. Both native and JAI implementations of mosaic are provided. When selected, Mosaic Native Acceleration use the native implementation for creating mosaics.

5.3 Layer Preview

This page provides layer views in various output formats. A layer must be enabled to be previewed.

Each layer row consists of a type, name, title, and available formats for viewing.

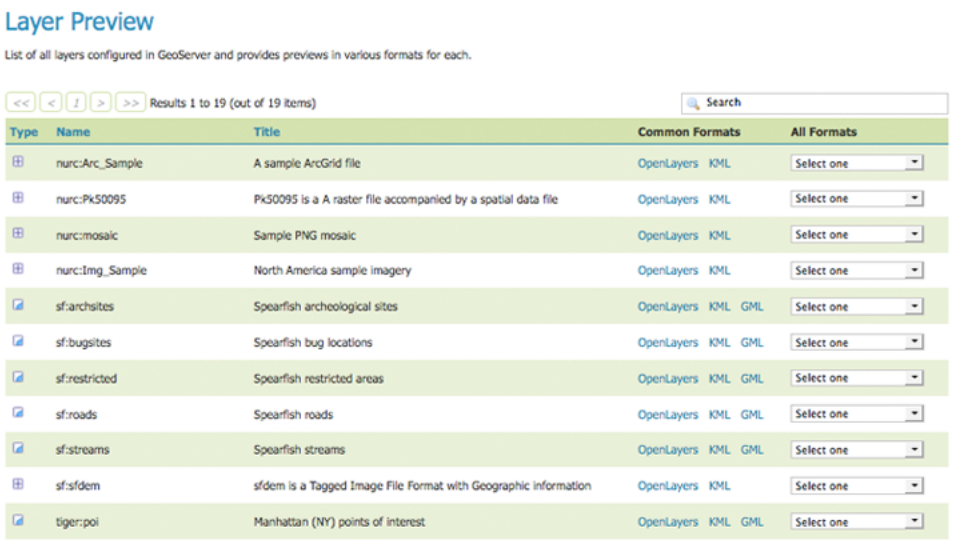


Figure 5.9: *Layer's Preview Page*

Field	Description
	Raster (grid) layer
	Vector (feature) layer
	Layer group

Name refers to the Workspace and Layer Name of a layer, while Title refers to the brief description configured in the *Edit Layer Data* panel. In the following example, nurc refers to the Workspace, Arc_Sample refers to the Layer Name and "A sample ArcGrid field" is specified on the Edit Later Data panel.

Figure 5.10: *Single Layer preview row*

5.3.1 Output Formats

The Layer Preview page supports a variety of output formats for further use or data sharing. You can preview all three layer types in the common OpenLayers and KML formats. Similarly, using the "All formats" menu you can preview all layer types in seven additional output formats—AtomPub, GIF, GeoRss, JPEG, KML (compressed), PDF, PNG, SVG, and TIFF. Only Vector layers provide the WFS output previews, including the common GML as well as the CSV, GML3, GeoJSON and shapefile formats. The table below provides a brief description of all supported output formats, organized by output type (image, text, or data).

Image Outputs

All image outputs can be initiated from a WMS getMap request on either a raster, vector or coverage data. WMS are methods that allows visual display of spatial data without necessarily providing access to the features that comprise those data.

Format	Description
KML	KML (Keyhole Markup Language) is an XML-based language schema for expressing geographic data in an Earth browser, such as Google Earth or Google Maps. KML uses a tag-based structure with nested elements and attributes. For GeoServer, KML files are distributed as a KMZ, which is a zipped KML file.
JPEG	WMS output in raster format. The JPEG is a compressed graphic file format, with some loss of quality due to compression. It is best used for photos and not recommended for exact reproduction of data.
GIF	WMS output in raster format. The GIF (Graphics Interchange Format) is a bitmap image format best suited for sharp-edged line art with a limited number of colors. This takes advantage of the format's lossless compression, which favors flat areas of uniform color with well defined edges (in contrast to JPEG, which favors smooth gradients and softer images). GIF is limited to an 8-bit palette, or 256 colors.
SVG	WMS output in vector format. SVG (Scalable Vector Graphics) is a language for modeling two-dimensional graphics in XML. It differs from the GIF and JPEG in that it uses graphic objects rather than individual points.
TIFF	WMS output in raster format. TIFF (Tagged Image File Format) is a flexible, adaptable format for handling multiple data in a single file. GeoTIFF containts geographic data embedded as tags within the TIFF file.
PNG	WMS output in raster format. The PNG (Portable Network Graphics) file format was created as the free, open-source successor to the GIF. The PNG file format supports truecolor (16 million colors) while the GIF supports only 256 colors. The PNG file excels when the image has large, uniformly coloured areas.
Open-Layers	WMS GetMap request outputs a simple OpenLayers preview window. OpenLayers is an open source JavaScript library for displaying map data in web browsers. The OpenLayers output has some advanced filters that are not available when using a standalone version of OpenLayers. Further, the generated preview contains a header with easy configuration options for display.
PDF	A PDF (Portable Document Format) encapsulates a complete description of a fixed-layout 2D document,including any text, fonts, raster images, and 2D vector graphics.

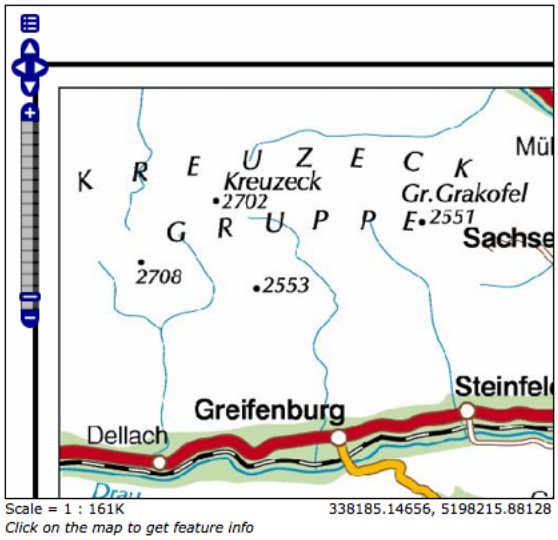


Figure 5.11: *Sample Image Output-an OpenLayers preview of nurc:Pk50095*

Text Outputs

Format	Description
AtomPub	WMS output of spatial data in XML format. The AtomPub (Atom Publishing Protocol) is an application-level protocol for publishing and editing Web Resources using HTTP and XML. Developed as a replacement for the RSS family of standards for content syndication, Atom allows subscription of geo data.
GeoRss	WMS GetMap request output of vector data in XML format. RSS (Rich Site Summary) is an XML format for delivering regularly changing web content. GeoRss is a standard for encoding location as part of a RSS feed.supports Layers Preview produces a RSS 2.0 documents, with GeoRSS Simple geometries using Atom.
GeoJSON	JavaScript Object Notation (JSON) is a lightweight data-interchange format based on the JavaScript programming language. This makes it an ideal interchange format for browser based applications since it can be parsed directly and easily in to javascript. GeoJSON is a plain text output format that add geographic types to JSON.
CSV	WFS GetFeature output in comma-delimited text. CSV (Comma Separated Values) files are text files containing rows of data. Data values in each row are separated by commas. CSV files also contain a comma-separated header row explaining each row's value ordering. GeoServer's CSVs are fully streaming, with no limitation on the amount of data that can be outputted.

A fragment of a simple GeoRSS for nurc:Pk50095 using Atom:

```
<?xml version="1.0" encoding="UTF-8"?>
     <rss xmlns:atom="http://www.w3.org/2005/Atom"
          xmlns:georss="http://www.georss.org/georss" version="2.0">
        <channel>
              <title>Pk50095</title>
              <description>Feed auto-generated by GeoServer</description>
              <link>></link>
              <item>
                <title>fid--f04ca6b_1226f8d829e_-7ff4</title>
                <georss:polygon>46.722110379286 13.00635746384126
                     46.72697223230676 13.308182612644663 46.91359611878293
                     13.302316867622581 46.90870264238999 12.999446822650462
                     46.722110379286 13.00635746384126
                </georss:polygon>
                </item>
        </channel>
</rss>
```

Data Outputs

All data outputs are initiated from a WFS GetFeature request on vector data.

Format	Description
GML2/3	GML (Geography Markup Language) is the XML grammar defined by the Open Geospatial Consortium (OGC) to express geographical features. GML serves as a modeling language for geographic systems as well as an open interchange format for geographic data sharing. GML2 is the default (Common) output format, while GML3 is available from the "All Formats" menu.
Shapefile	The ESRI Shapefile, or simply a shapefile, is the most commonly used format for exchanging GIS data. GeoServer outputs shapefiles in zip format, with a directory of .cst, .dbf, .prg, .shp, and .shx files.

5.4 Data

This section is the largest and perhaps the most important section of the Web Administration Interface. Each subsection links directly to a data type page with add, edit, and delete capabilities.

In the example below, the data view page displays a table of indexed data.

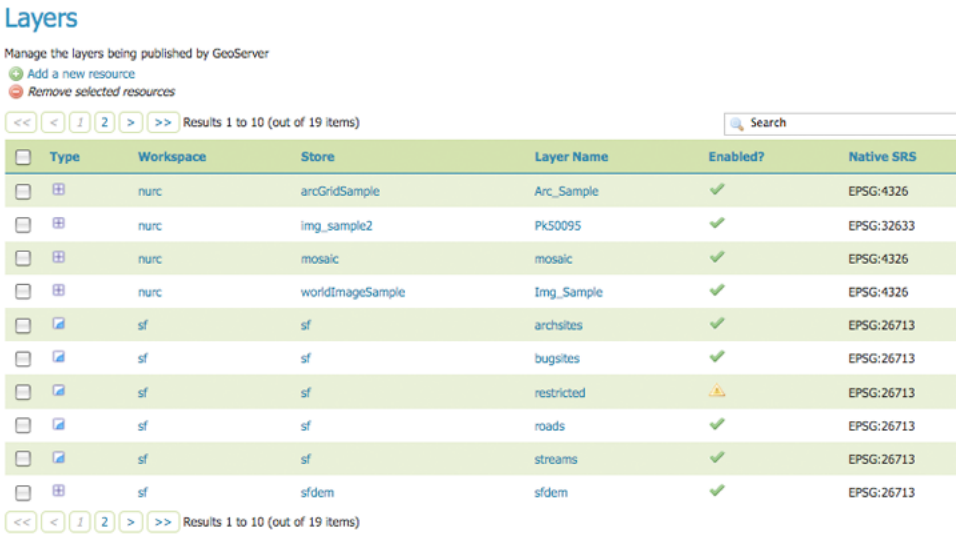


Figure 5.12: *Layers page*

To sort a data type alphabetically, click the column header.

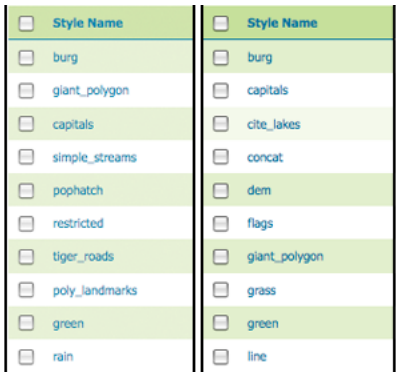


Figure 5.13: *Unsorted (left) and sorted (right) columns*

For simple searching of data type contents, enter the search criteria in the search box and click Enter. GeoServer will search the data types that are relevant to your query, and return a Search Results page.

Specific details for adding, editing and deleting various data types are discussed in the following sections.

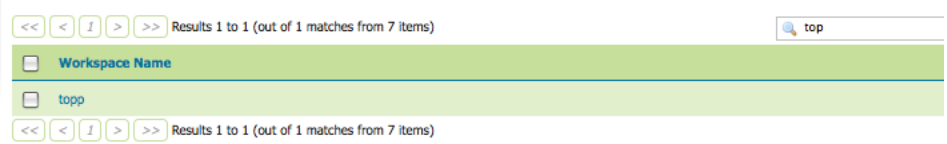


Figure 5.14: *Search results for the query "top"*

5.4.1 Workspaces

This section describes how to view and configure workspaces. Analogous to a namespace, a workspace is a container which organizes other items. In GeoServer, a workspace is often used to group similar layers together. Individual layers are often referred to by their workspace name, colon, then store. For example, Ex: topp:states. Two different layers with the same name can exist as long as they exist in different workspaces. For example, Ex: sf:states, topp:states.

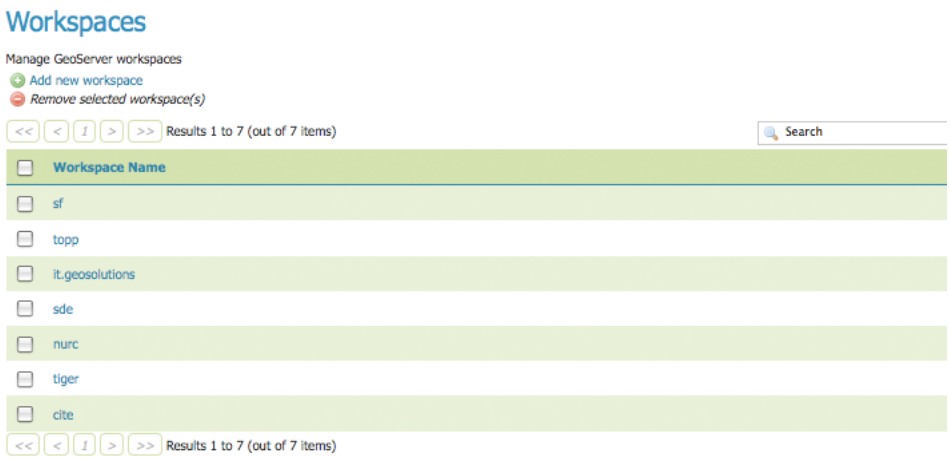


Figure 5.15: *Workspaces page*

Edit Workspace

To view details and edit a workspace, click a workspace name.

Figure 5.16: *Workspace named "topp"*

A workspace consists of a name and a Namespace URI (Uniform Resource Identifier). The workspace name is limited of ten characters and may not contain space. A URI is similar to a URL, except URIs don't need to point to a location on the web, and only need to be a unique identifier. For a Workspace URI, we recommend using a URL associated with your project, with perhaps a different trailing identifier, such as `http://www.openplans.org/topp` for the "topp" workspace.

Add or Remove a Workspace

The buttons for adding and removing a workspace can be found at the top of the Workspaces view page.

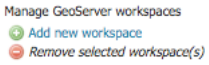


Figure 5.17: *Buttons to add and remove*

To add a workspace, select the *Add new workspace* button. You will be prompted to enter the the workspace name and URI.

Figure 5.18: *New Workspace page with example*

To remove a workspace, click the workspace's corresponding check box. As with the layer deletion process, multiple workspaces can be checked for removal on a single results page. Click the *Remove selected workspaces(s)* button. You will be asked to confirm or cancel the deletion. Clicking *OK* will remove the workspace.

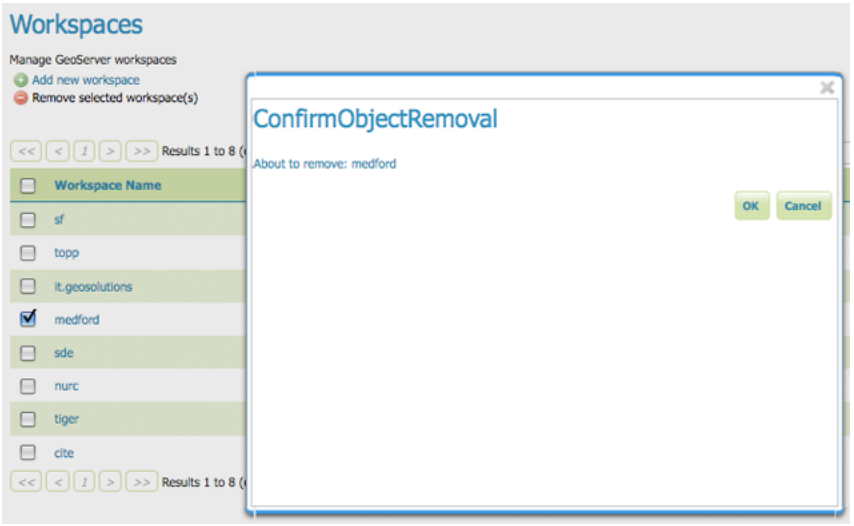


Figure 5.19: *Workspace removal confirmation*

5.4.2 Stores

A store connects to a data source that contains raster or vector data. A data source can be a file or group of files, a table in a database, a single raster file, or a directory, for example a Vector Product Format library. Using the store construct means that connection parameters are defined once, rather than for each piece of data in a source. As such, it is necessary to register a store before loading any data.

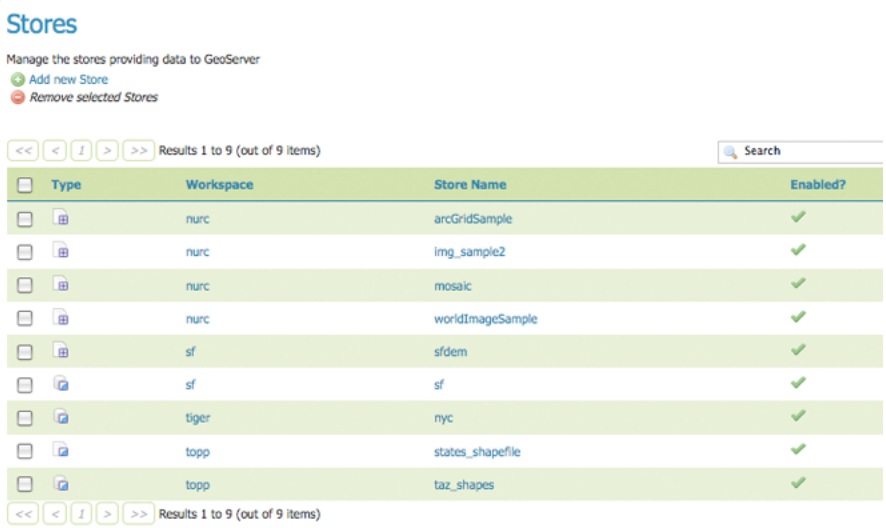


Figure 5.20: *Stores View*

While there are many potential formats for data source, there are only four types of stores. For raster data, a store can be a file. For vector data, a store can be a file, database, or server.

Type Icon	Description
	raster data in a file
	vector data in a file
	vector data in a database
	vector server (web feature server)

Editing a Store

To view and edit a store, click a store name. The exact contents of this page depend on the specific format chosen. (See the sections *Working with Vector Data*, *Working with Raster Data*, and *Working with Databases* for information about specific data formats.) In the example lists the contents of the `nurc:ArcGridSample` store.

While connection parameters will vary depending on data format, some of the basic information is common across formats. The Workspace menu lists all registered workspaces. The store is assigned to the selected workspace (`nurc`). *Data Source Name* is the store name as listed on the view page. The *Description* is optional and only displays in the administration interface. *Enabled* enables or disables access to the store, along with all data defined in it.

Adding a Store

The buttons for adding and removing a workspace can be found at the top of the Stores page.

To add a workspace, select the *Add new Store* button. You will be prompted to choose a data source. GeoServer natively supports many formats (with more available via extensions). Click the appropriate data source to continue.

The next page will configure the store. (The example below shows the ArcGrid raster configuration page.) However, since connection parameters differ across data sources, the exact contents of this page depend on the store's specific format. See the sections *Working with Vector Data*, *Working with Raster Data*, and *Working with Databases* for information on specific data formats.

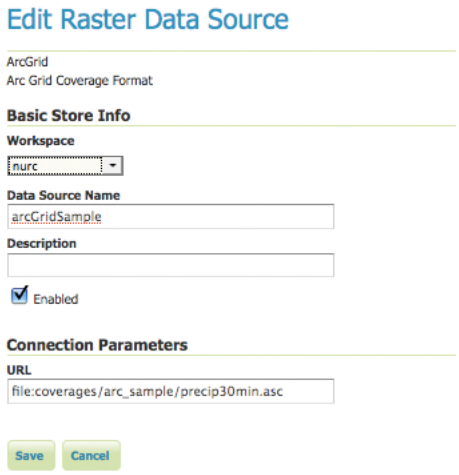


Figure 5.21: *Editing a raster data store*

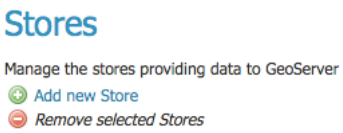


Figure 5.22: *Buttons to add and remove stores*

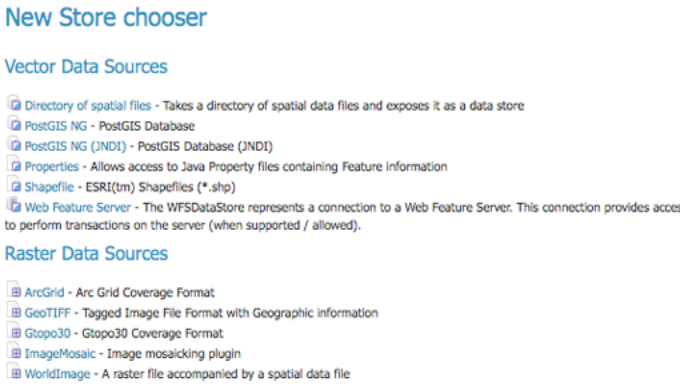


Figure 5.23: *Choosing the data source for a new store*

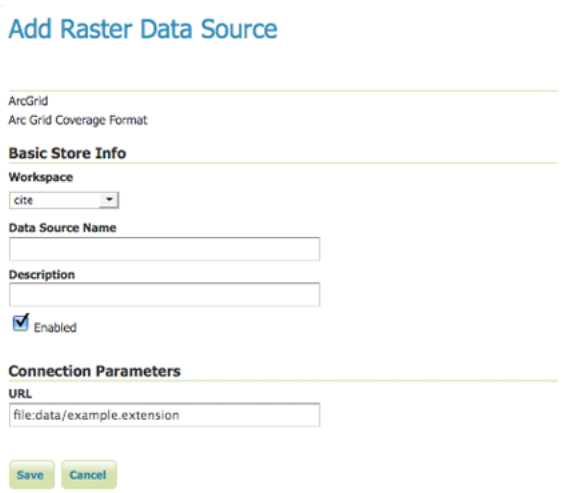


Figure 5.24: *Configuration page for an ArcGrid raster data source*

Removing a Store

To remove a store, click the store's corresponding check box. Multiple stores can be selected for batch removal.

Figure 5.25: *Stores selected for deletion*

Click the *Remove selected Stores* button. You will be asked to confirm the deletion of the the data within each store. Selecting *OK* removes the store(s), and will redirect to the main Stores page.

5.4.3 Layers

In Geoserver, the term layer refers to raster or vector data that contains geographic features. Vector layers are analogous to featureTypes and raster layers are analogous to coverages. Layers represent each feature that needs to be shown on a map. All layers have a source of data, known as a Store.

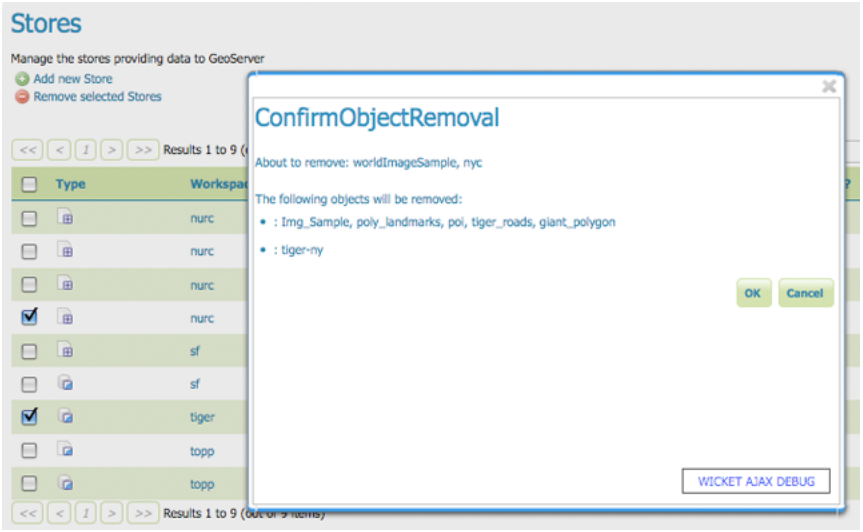


Figure 5.26: *Confirm deletion of stores*

In the layers section, you can view and edit an existing layers, add (register) a new layer, or delete (unregister) a layer. As in previous View tables, the Layers View page displays relevant dependencies, that is, the layer within the store within the workspace. The View page also displays the layer's status and native SRS.

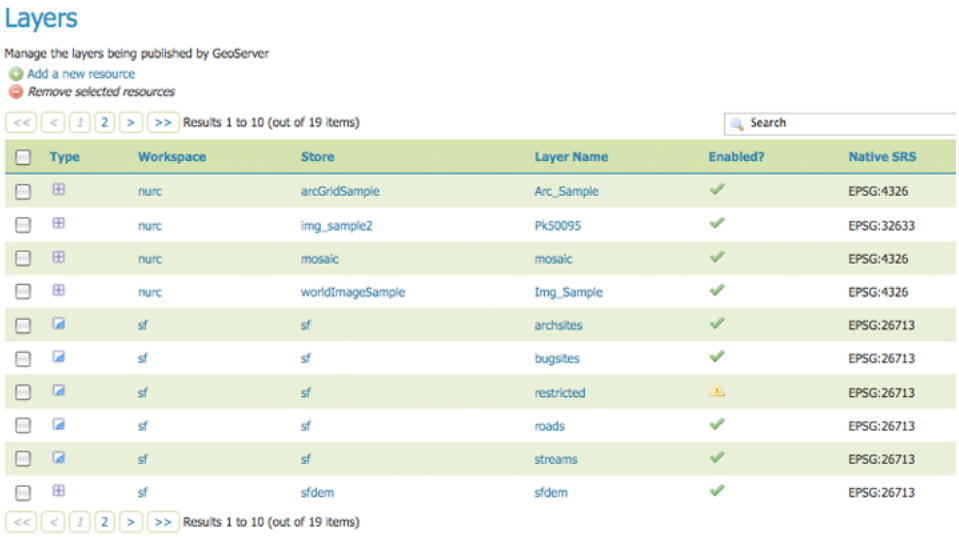


Figure 5.27: *Layers View*

Layer Types

Layers are organized into two types of data, raster and vector. These two formats differ in how they store spatial information. Vector types store information about feature types as mathematical paths—a point as a single x,y coordinate, lines as a series of x,y coordinates, and polygons as a series of x,y coordinates that start and end on the same place. Raster format data is a cell-based representation of features on the earth surface. Each cell has a distinct value, and all cells with the same value represent a specific feature.

Field	Description
	raster (grid)
	vector (feature)

Edit Layer Data

Clicking the layer name opens a layer configuration panel. The *Data* tab, activated by default, allows you to define and change data parameters for a layer.

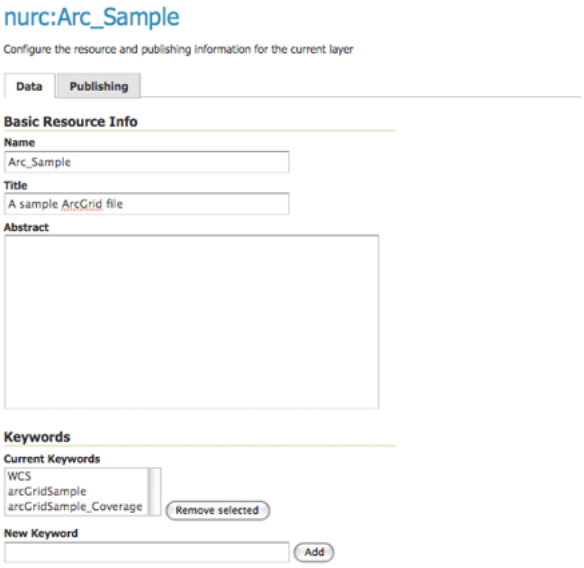


Figure 5.28: *Layers Data View*

Basic Info

The beginning sections–Basic Resource Info, Keywords and Metadata link are analogous to the *Service Metadata* section for WCS, WFS and WMS. These sections provide "data about the data," specifically textual information that make the layer data easier to work with it.

Name—Identifier used to reference the layer in WMS requests

Title—Human-readable description to briefly identify the layer to clients (required)

Abstract—Describes the layer

Keywords—List of short words associated with the layer to assist catalog searching

Metadata Link—Allows linking to external documents that describe the data layer. Currently only two standard format types are valid: TC211 and FGDC. TC211 refers to the metadata structure established by the ISO Technical Committee for Geographic Information/Geomatics (ISO/TC 211) while FGDC refers to those set out by the Federal Geographic Data Committee (FGDC) of the United States.

Coordinate Reference Systems

A coordinate reference system (CRS) defines how your georeferenced spatial data relates to real locations on the Earth's surface. CRSs are part of a more general model called Spatial Reference Systems (SRS), which

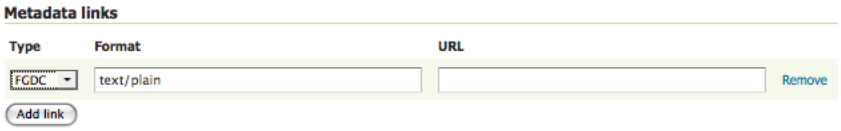


Figure 5.29: *Adding a metadata link n FGDC format*

includes referencing by coordinates and geographic identifiers. Geoserver needs to know what Coordinate Reference System of your data. This information is used for computing the latitude/longitude bounding box and reprojecting the data during both WMS and WFS requests

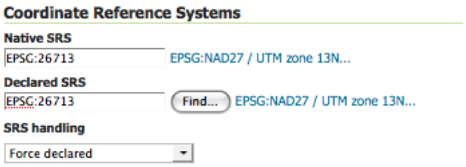


Figure 5.30: *Adding a metadata link n FGDC format*

Native SRS—Refers to the projection the layer is stored in. Clicking the projection link displays a description of the SRS.

Declared SRS—Refers to what GeoServer gives to clients

SRS Handling:—Determines how GeoServer should handle projection when the two SRS differ

Bounding Boxes

The bounding box determines the extent of a layer. The *Native Bounding Box* are the bounds of the data projected in the Native SRS. You can generate these bounds by clicking the *Compute from data* button. The *Lat/Long Bounding Box* computes the bounds based on the standard lat/long. These bounds can be generated by clicking the *Compute from native bounds* button.

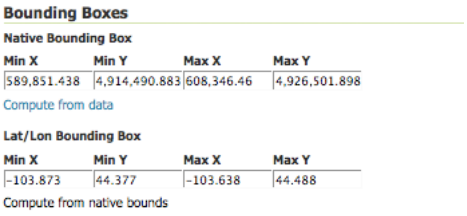


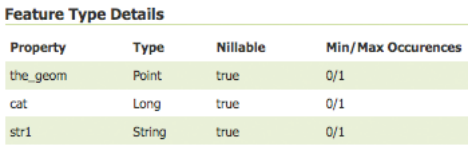
Figure 5.31: *Bounding Box for sf:archsites*

Coverage Parameters (Raster)

Optional coverage parameters are possible for certain types of raster data. WorldImage formats request a valid range of grid coordinates in two dimensions known as a *ReadGridGeometry2D*. For ImageMosaic, you can use *InputImageThresholdValue*, *InputTransparentColor*, and *OutputTransparentColor* to control the rendering of the mosaic in terms of thresholding and transparency.

Feature Type Details (Vector)

Instead of coverage parameters, vector layers have a list of the *Feature Type Details*. These include the *Property* and *Type* of a data source. For example, the `sf:archsites` layer show below includes a geometry, `the_geom` of type point.



Feature Type Details

Property	Type	Nillable	Min/Max Occurences
the_geom	Point	true	0/1
cat	Long	true	0/1
str1	String	true	0/1

Figure 5.32: *Feature Types Detaisl for sf:archsites*

The *Nillable* refers to whether the property requires a value or may be flagged as being null. Meanwhile *Min/Max Occurrences* refers to how many values a field is allowed to have. Currently both *Nillable* and *Min/Max Occurrences* are set to `true` and `0/1` but might be extended with future work on complex features.

Edit Publishing Information

The publishing tab is for configuring HTTP and WCS settings.

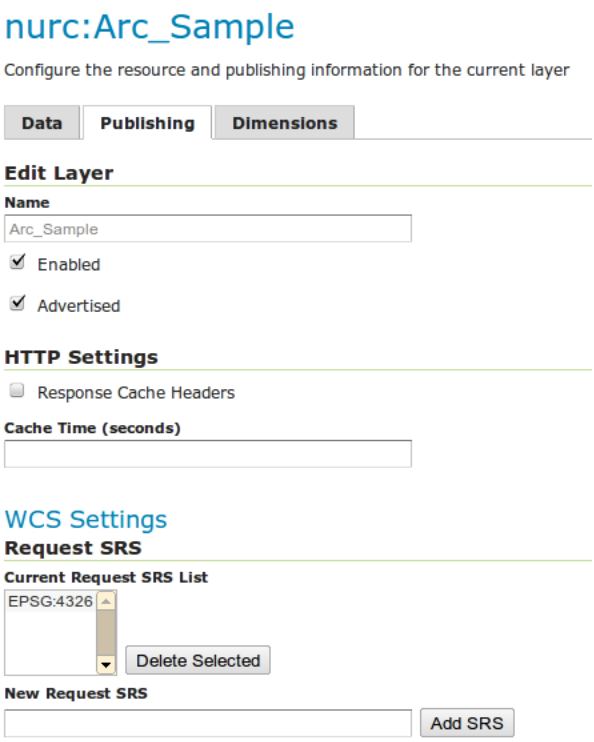


Figure 5.33: *Editing Publishing Data*

- *Enabled*—A layer that is not enabled won't be available to any kind of request, it will just show up in the configuration (and in REST-config)

- *Additional styles*—A layer is advertised by default. A non-advertised layer will be available in all data access requests (for example, WMS GetMap, WMS GetFeature) but won't appear in any capabilities document or in the layer preview.

HTTP Settings—Cache parameters that apply to the HTTP response from client requests. If *Response Cache Headers* is selected, GeoServer will not request the same tile twice within the time specified in *Cache Time*. One hour measured in seconds (3600), is the default value for *Cache Time*.

WMS Settings—Sets the WMS specific publishing parameters

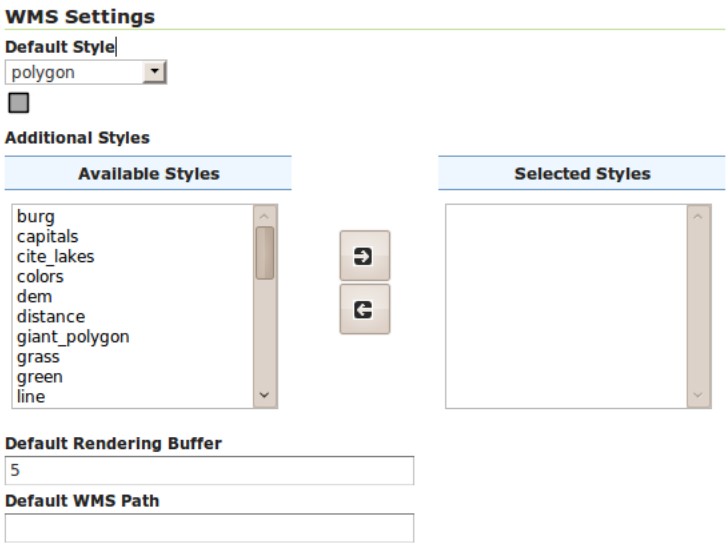


Figure 5.34: *WMS Settings*

- *Default style*:—Style that will be used when the client does not specify a named style in GetMap requests
- *Additional styles*—Other styles that can be associated to this layers. Some clients (and the GeoServer own preview) will present those as styling alternatives for that layer to the end user
- *Default rendering buffer* (available since version 2.0.3)—the default value of the `buffer` GetMap/GetFeatureInfo vendor parameter. See the *WMS vendor parameters* for more details
- *Default WMS path*—Location of the layer in the WMS capabilities layer tree. Useful to build non-opaque layer groups

WMS Attribution—Sets publishing information about data providers

- *Attribution Text*—Human-readable text describing the data provider. This might be used as the text for a hyperlink to the data provider's web site.
- *Attribution Link*—URL to the data provider's website.
- *Logo URL*—URL to an image that serves as a logo for the data provider.
- *Logo Content Type, Width, and Height* —These fields provide information about the logo image that clients may use to assist with layout. GeoServer will auto-detect these values if you click the *Auto-detect image size and type* link at the bottom of the section.

The text, link, and URL are each advertised in the WMS Capabilities document if they are provided. Some WMS clients will display this information to advise users which providers provide a particular dataset. If you omit some of the fields, those that are provided will be published and those that are not will be omitted from the Capabilities document.

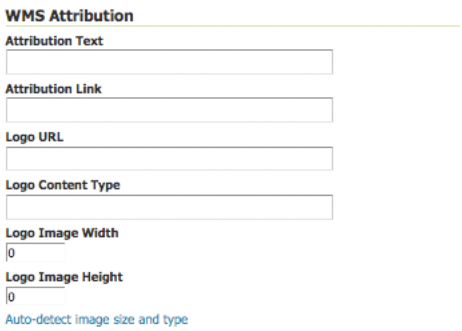


Figure 5.35: *WMS Attribution*

WFS Settings—Sets the maximum number of features for a layer a WFS GetFeature operation should generate (regardless of the actual number of query hits)

WCS Settings—Provides a list the SRS the layer can be converted to. *New Request SRS* allows you to add an SRS to that list.

Interpolation Methods—Sets the raster rendering process

Formats—Lists which output formats a layers supports

Default Title—Assigns a style to a layer. Additional styles are ones published with the layer in the capabilities document.

Geosearch—When enabled, allows the Google Geo search crawler to index from this particular layer. See What is a Geo Sitemap? for more information.

KML Format Settings—Limits features based on certain criteria, otherwise known as *regionation*. Choose which feature should show up more prominently than others with the guilabel:*Default Regionating Attribute*. There are four types of *Regionating Methods*:

- *external-sorting*—Creates a temporary auxiliary database within GeoServer. The first request to build an index takes longer than subsequent requests.
- *geometry*—Externally sorts by length (if lines) or area (if polygons)
- *native-sorting*—Uses the default sorting algorithm of the backend where the data is hosted. It is faster than external-sorting, but will only work with PostGIS datastores.
- *random*—Uses the existing order of the data and does not sort

Add or Delete a Layer

At the upper left-hand corner of the layers view page there are two buttons for the adding and deletion of layers. The green plus button allows you to add a new layer, referred to as resource. The red minus button allows you to remove selected layers.

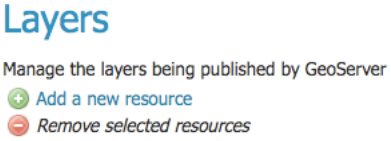


Figure 5.36: *Buttons to Add or Remove a Layer*

Clicking the *Add a new resource* button brings up a *New Layer Chooser* panel. The menu displays all currently enabled stores. From this menu, select the Store where the layer should be added.

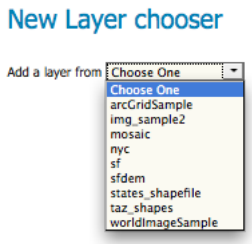


Figure 5.37: *List of all currently enabled stores*

Upon selection of a Store, a view table of existing layers within the selected store will be displayed. In this example, `giant_polygon`, `poi`, `poly_landmarks` and `tiger_roads` are all layers within the NYC store.

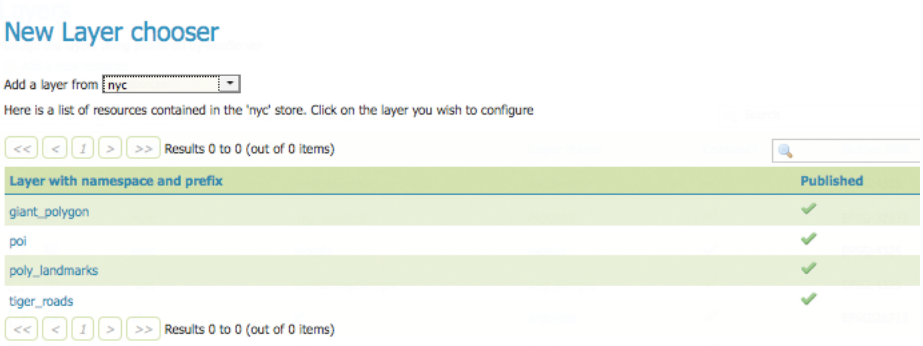


Figure 5.38: *View of all layers*

On selecting a layer name, you are redirected to a layer edit page. *Edit Layer Data*

To delete a layer, click the check box on the left side of each layer row. As shown below, multiple layers can be selected for removal on a single results page. It should be noted, however, that selections for removal will not persist from one results pages to the next.

All layers can be selected for removal by selecting the check box in the header row.

Once layer(s) are selected, the *Remove selected resources* link is activated. Once you've clicked the link, you will be asked to confirm or cancel the deletion. Selecting *OK* successfully deletes the layer.

5.4.4 Layer Groups

A layer group is a container in which layers and other groups can be organized in a hierarchical structure. A layer group can be referred to by one name, this allows for simpler WMS requests, as the request need only refer to one layer as opposed to multiple individual layers.

Layer group behaviour can be configured by setting its *mode*. There are 4 available values:

- **single**: the layer group is exposed as a single layer with a name.

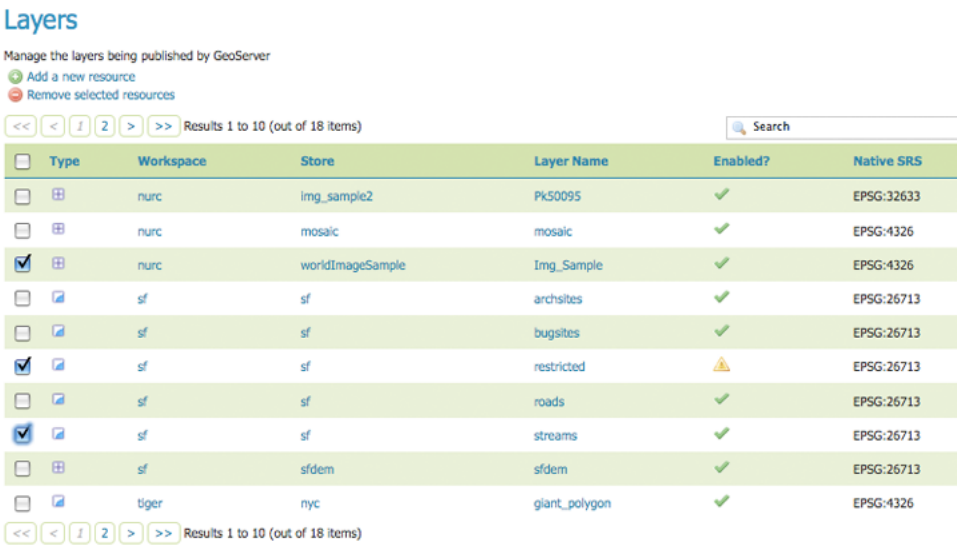


Figure 5.39: *Layers nurc:Img_Sample, sf:restricted, sf:streams selected for deletion*

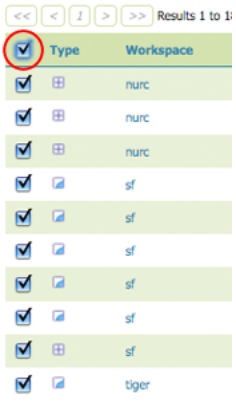


Figure 5.40: *All layers selected to be deleted*

- **named tree**: the layer group can be referred to by one name, but also exposes its nested layers and groups in the capabilities document.
- **container tree**: the layer group is exposed in the capabilities document, but does not have a name, making it impossible to render it on its own. This is called "containing category" in the WMS specification.
- **Earth Observation tree**: a special type of group created to manage the WMS Earth Observation requirements. This group does not render its nested layers and groups, but only a "preview layer" called Root Layer. When this mode is chosen, a new field "Root Layer" will be exposed in the configuration UI.

In case a layer is included in a any non *single* group it won't be listed anymore in the flat layer list, although it will still be possible to include the same layer in different layer groups.

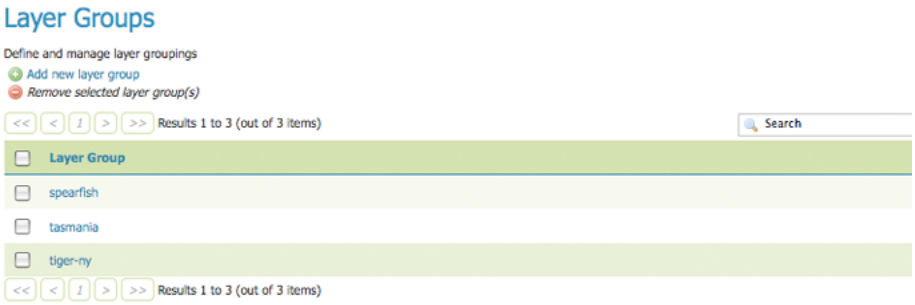


Figure 5.41: *Layer Groups page*

Edit Layer Group

To bring up the layer group edit page, click a layer group name. The initial fields allow you configure the name, title, abstract, workspace, bounds, projection and mode of the layer group. To automatically set bounding box, select the *Generate Bounds* button. You may also provide your own custom bounding box parameters. To select an appropriate projection click the *Find* button.

Note: A layer group can consist of layers with dissimilar bounds and projections. GeoServer will automatically reproject all layers to the projection of the layer group.

The table at the bottom of the page lists layers and groups contained within the current layer group. We refer to layers and layer groups as *publishable elements*. When a layer group is processed, the layers are rendered in the order provided, so the *publishable elements* at the bottom of list will be rendered last and will show on top of the other *publishable elements*.

A *publishable element* can be positioned higher or lower on this list by clicking the green up or down arrows, respectively.

The *Style* column shows the style associated with each layer. To change the style associated with a layer, click the appropriate style link. A list of enabled styles will be displayed. Clicking on a style name reassigns the layer's style.

To remove a *publishable element* from the layer group, select its button in the *Remove* column. You will now be prompted to confirm or cancel this deletion.

A layer can be added to the list by clicking the *Add Layer...* button at the top of the table. From the list of layers, select the layer to be added by clicking the layer name. The selected layer will be appended to the bottom of the *publishable* list.

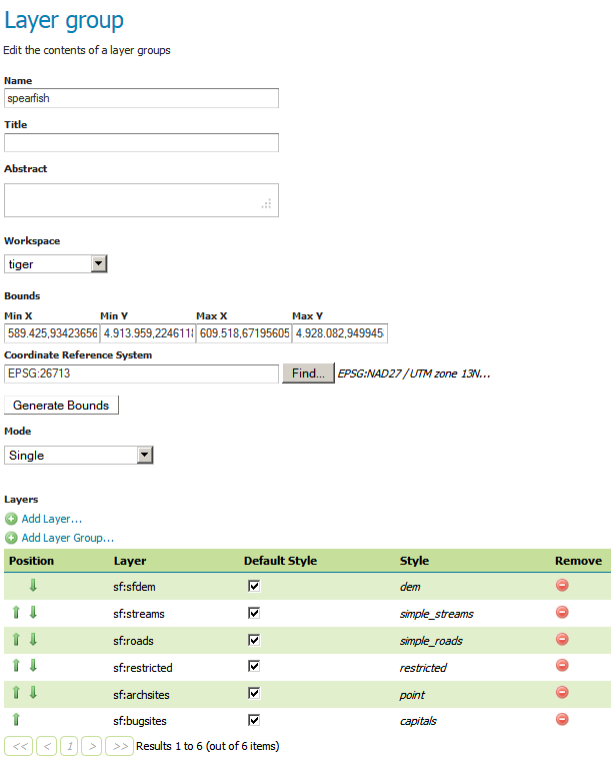


Figure 5.42: *Layer Groups Edit page*

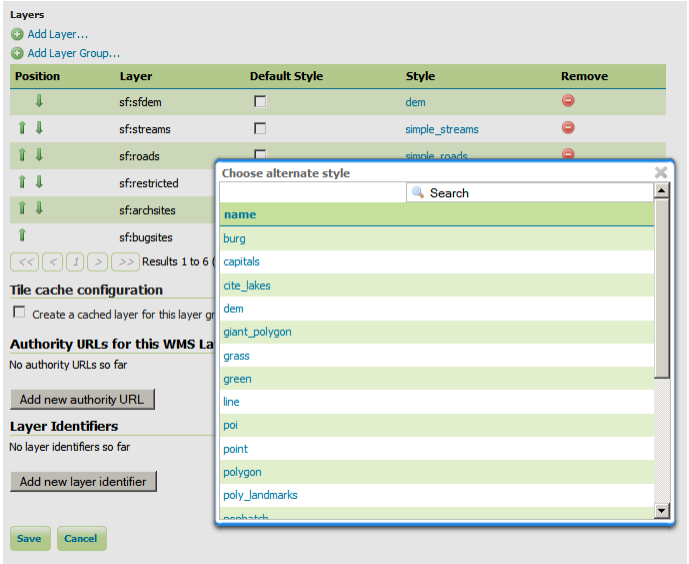


Figure 5.43: *Style editing for a layer within a layer group*

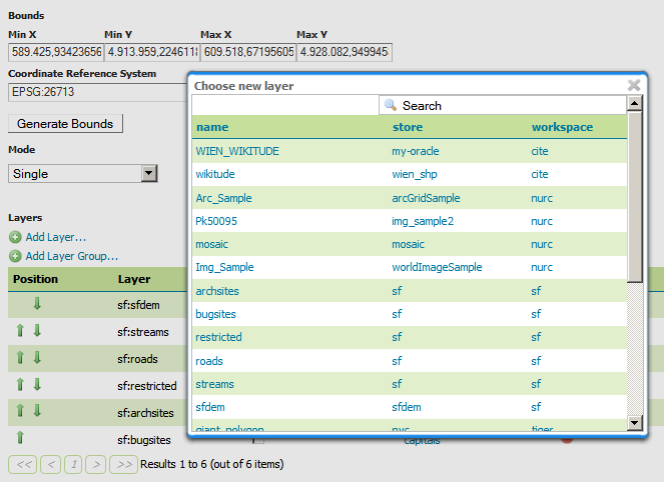


Figure 5.44: *Dialog for adding a layer to a layer group*

A layer group can be added by clicking the *Add Layer Group...* button at the top of the table. From the list of layer groups, select the layer group to be added by clicking its name. The selected group will be appended to the bottom of the *publishable* list.

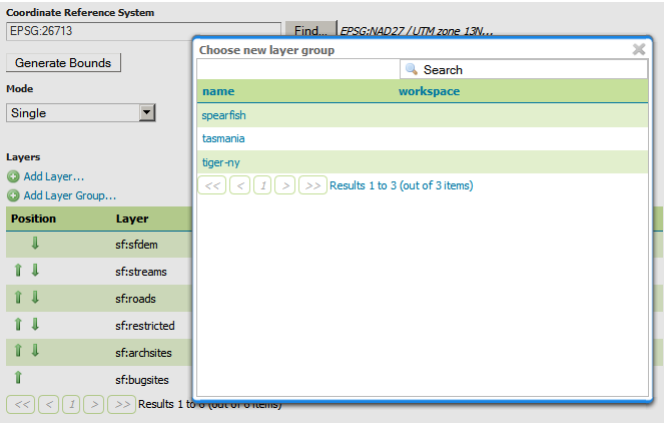


Figure 5.45: *Dialog for adding a layer group to a layer group*

You can view layer groups in the *Layer Preview* section of the web admin.

Add a Layer Group

The buttons for adding and removing a layer group can be found at the top of the *Layer Groups* page.

To add a new layer group, select the "Add a new layer group" button. You will be prompted to name the layer group.

When finished, click *Submit*. You will be redirected to an empty layer group configuration page. Begin by adding layers by clicking the *Add layer...* button (described in the previous section). Once the layers are positioned accordingly, press *Generate Bounds* to automatically generate the bounding box and projection. Press *Save* to save the new layer group.

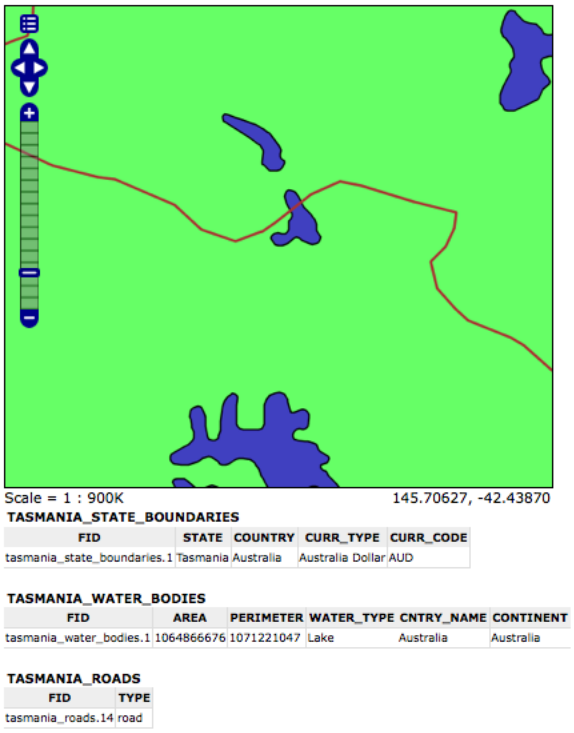


Figure 5.46: *Openlayers preview of the layer group "tasmania"*

Figure 5.47: *Buttons to add or remove a layer group*

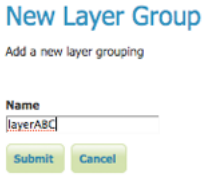


Figure 5.48: *New layer group dialog*

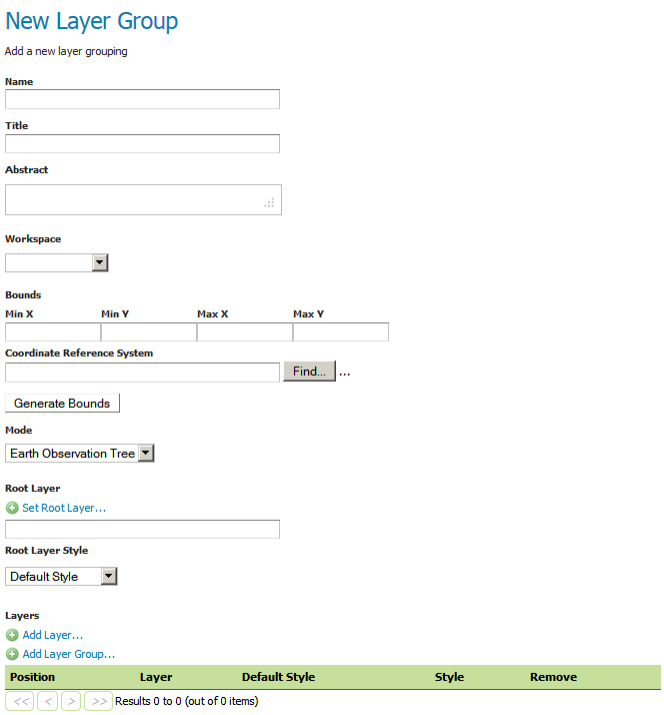


Figure 5.49: *New layer group configuration page*

Remove a layer group

To remove a layer group, click the check box next to the layer group. Multiple layer groups can be selected for batch removal. Click the *remove selected layer group(s)* link. You will be asked to confirm or cancel the deletion. Selecting *OK* successfully removes the layer group.

5.4.5 Styles

Styles render, or make available, geospatial data. Styles for GeoServer are written in Styled Layer Descriptor (SLD), a subset of XML. Please see the section on *Styling* for more information on working with styles.

On the Styles page, you can register or create a new style, edit an existing style, or delete remove a style.

Edit Styles

The *Style Editor* page presents options for configuring a style's name and code. SLD names are specified at the top in the name field. Typing or pasting of SLD code can be done in one of two modes. The first mode is an embedded EditArea a rich editor. The second mode is an unformatted text editor. Check the *Toggle Editor* to switch between modes.

The rich editor is designed for text formatting, search and replace, line numbering, and real-time syntax highlighting. You can also switch view to full-screen mode for a larger editing area.

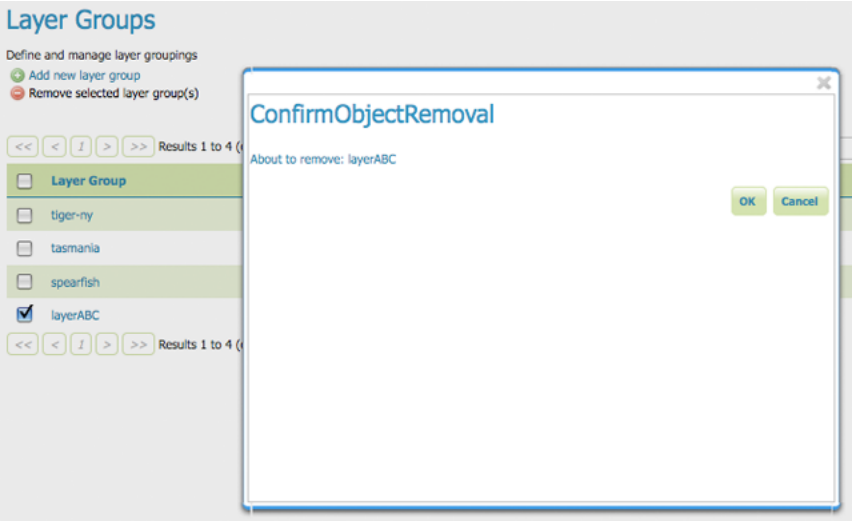


Figure 5.50: *Removing a layer group*

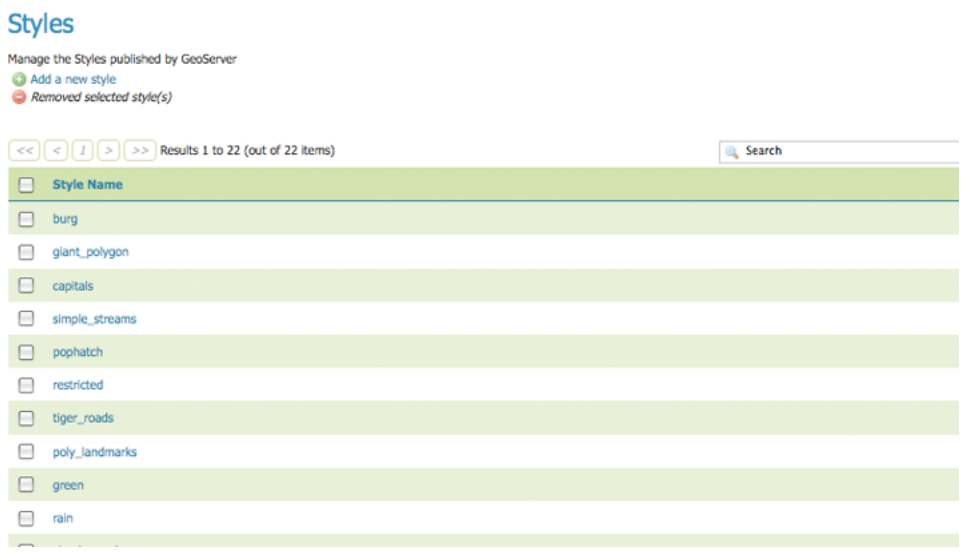


Figure 5.51: *Styles page*

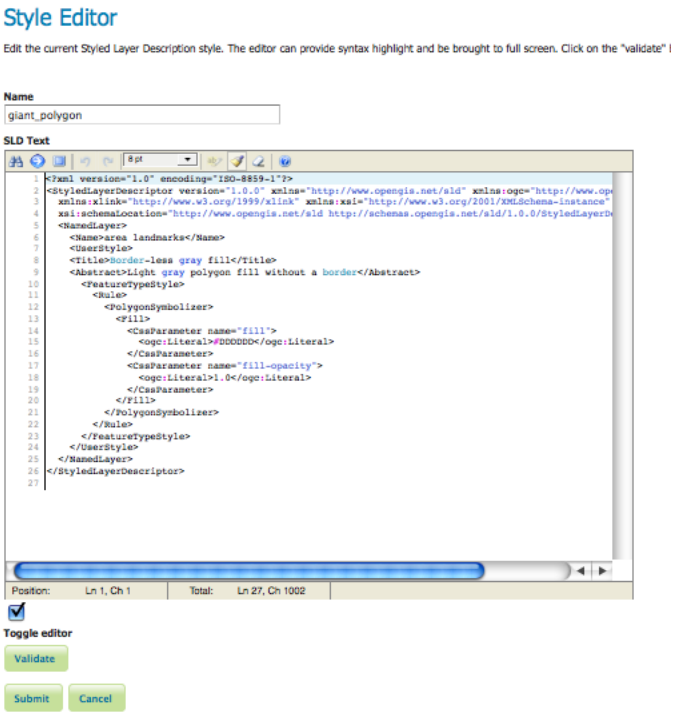


Figure 5.52: *Rich text editor*

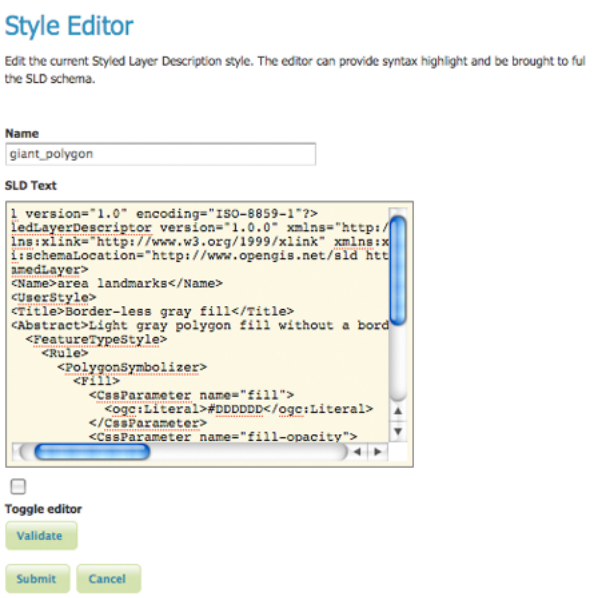


Figure 5.53: *Plain text editor*

Button	Description
	search
	go to line
	fullscreen mode
	undo
	redo
	toggle syntax highlight on/off
	reset highlight (if desynchronized from text)
	about

To confirm that the SLD code is fully compliant with the SLD schema, click the *Validate* button. A message box will confirm whether the style contains validation errors.

Note: GeoServer will sometimes render styles that fail validation, but this is not recommended.

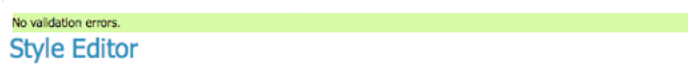


Figure 5.54: *No validation errors*

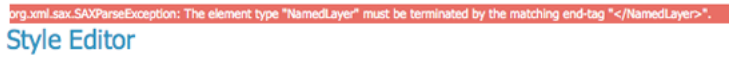


Figure 5.55: *Validation error message*

Add a Style

The buttons for adding and removing a style can be found at the top of the *Styles* page.

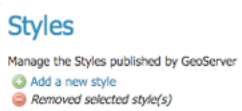


Figure 5.56: *Adding or removing a style*

To add a new layer group, select the *Add a new style* button. You will be redirected to an editor page. Enter a name for the style. The editor page provides two options for submitting an SLD. You can paste the SLD directly into the editor, or you can select and upload a local file that contains the SLD.

Once a style is successfully submitted, you will be redirected to the main *Styles* page where the new style will be listed.

Remove a Style

To remove a style, select the check box next to the style. Multiple layer groups can be selected for batch removal. Click the *Remove selected style(s)* link at the top of the page. You will be asked to confirm or cancel the deletion. Clicking *OK* removes the layer group.

Figure 5.57: *Uploading an SLD file from your local computer*

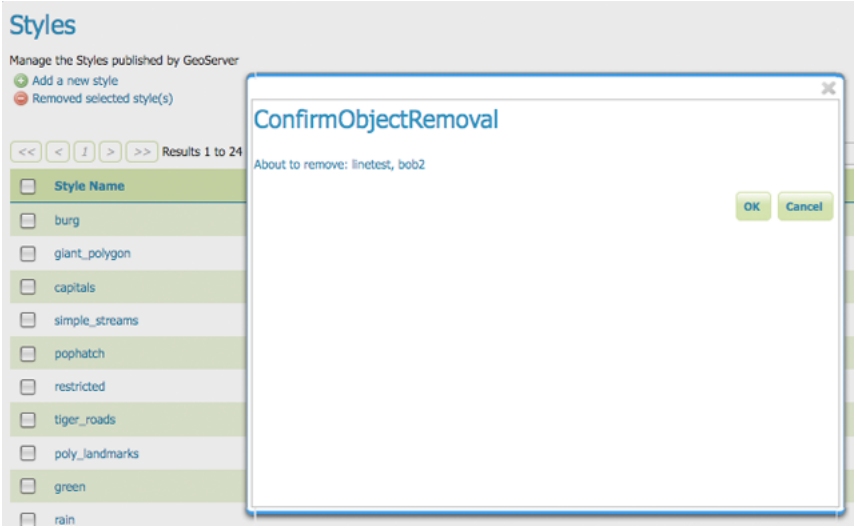


Figure 5.58: *Confirmation prompt for removing styles*

5.5 Services

GeoServer serves data using protocols established by the Open Geospatial Consortium (OGC). Web Coverage Service (WCS) supports requests for coverage data (rasters), Web Feature Service (WFS) supports requests of geographical feature data (vectors), and Web Map Service (WMS) allows for requests of images generated from geographical data.

This section of the Web Administration Interface describes how to configure these services for GeoServer.

5.5.1 WCS

The Web Coverage Service (WCS) provides few options for changing coverage functionality. While various elements can be configured for WFS and WMS requests, WCS allows only metadata information to be edited. This metadata information, entitled *Service Metadata*, is common to WCS, WFS and WMS requests.

Service Metadata

WCS, WFS, and WMS use common metadata definitions. These nine elements are described in the following table. Though these field types are the same regardless of service, their values are not shared. As such, parameter definitions below refer to the respective service. For example, "Enable" on the WFS Service page, enables WFS service requests and has no effect on WCS or WMS requests.

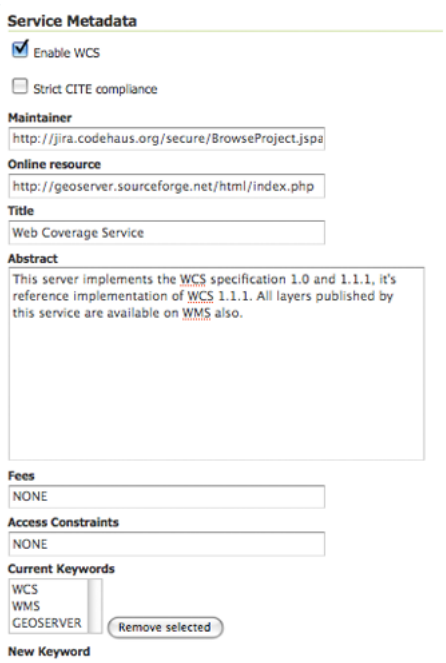


Figure 5.59: *WCS Configuration page*

Field	Description
Enabled	Specifies whether the respective services–WCS, WFS or WMS–should be enabled or disabled. When disabled, the respective service requests will not be processed.
Strict CITE compliance	When selected, enforces strict OGC Compliance and Interoperability Testing Initiative (CITE) conformance. Recommended for use when running conformance tests.
Maintainer	Name of the maintaining body
Online Resource	Defines the top-level HTTP URL of the service. Typically the Online Resource is the URL of the service "home page." (Required) |
Title	A human-readable title to briefly identify this service in menus to clients (required)
Abstract	Provides a descriptive narrative with more information about the service
Fees	Indicates any fees imposed by the service provider for usage of the service. The keyword NONE is reserved to mean no fees and fits most cases.
Access Constraints	Describes any constraints imposed by the service provider on the service. The keyword NONE is reserved to indicate no access constraints are imposed and fits most cases.
Keywords	List of short words associated with the service to aid in cataloging and searching

5.5.2 WFS

The Web Feature Service (WFS) page supports the configuration of features, service levels, and GML output.

Service Metadata

See the section on *Service Metadata*.

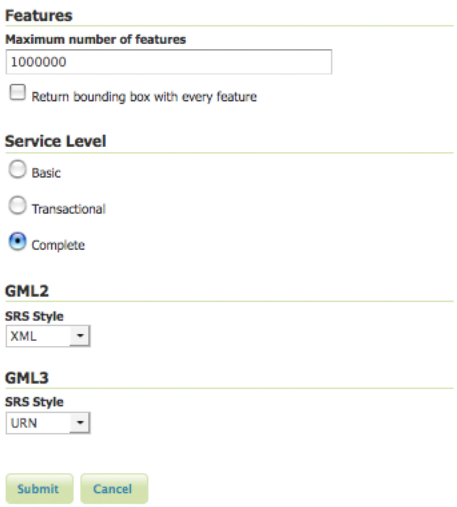


Figure 5.60: *WFS configuration options*

Features

The Open Geospatial Consortium (OGC) Web Feature Service (WFS) is a protocol for serving geographic features across the Web. Feature information that is encoded and transported using WFS includes both feature geometry and feature attribute values. Basic Web Feature Service (WFS) supports feature query and retrieval. Feature limits and bounding can be configured on the WFS page.

Maximum number of features—Maximum number of features sets the global feature limit that a WFS GetFeature operation should generate, regardless of the actual number of query hits. A WFS request can potentially contain a large dataset that is impractical to download to a client, and/or too large for a client's renderer. Maximum feature limits are also available for feature types. The default number is 1000000.

Return bounding box—Includes in the GetFeature GML output, an auto-calculated bounds element on each feature type. Not typically enabled, as including bounding box takes up extra bandwidth.

Service Levels

GeoServer is compliant with the full "Transactional Web Feature Server" (WFS-T) level of service as defined by the OGC. Specifying the WFS service level limits the capabilities of Geoserver while still remaining compliant. The WFS Service Level is an integer bitmask that indicates what WFS operations are "turned on." It defines the available operations and content at a service instance

Basic—Basic service levels provides facilities for searching and retrieving feature data with the GetCapabilities, DescribeFeatureType and GetFeature operations. It is compliant with the OGC basic Web Feature Service. This is considered a READ-ONLY web feature service.

Transactional—In addition to all basic WFS operations, transactional service level supports transaction requests. A transaction request facilities the creation, deletion, and updating of geographic features in conformance with the OGC Transactional Web Feature Service (WFS-T).

Complete—Includes the LockFeature support to the suite of transactional level operations. LockFeature operations help resolve links between related resources by processing lock requests on one or more instances of a feature type.

GML

Geography Markup Language (GML) is the XML-based specification defined by the Open Geospatial Consortium (OGC) to express geographical features. GML serves as a modeling language for geographic systems as well as an open interchange format for geographic transactions on the Internet.

The older GML standard, GML 2 encodes geographic information, including both spatial and non-spatial properties. GML3 extends GML2 support to 3D shapes (surfaces and solids) as well as other advanced facilities. GML3 is modular superset of GML2 that simplifies and minimizes the implementation size by allowing users to select out necessary parts. Additions in GML3 include support for complex geometries, spatial and temporal reference systems, topology, units of measure, metadata, gridded data, and default styles for feature and coverage visualization. GML3 is almost entirely backwards compatible with GML2.

WFS 1.1.0 requests return GML3 as the default GML and style a Spatial Reference System (SRS) is in the URN format. Meanwhile WFS 1.0.0 requests return GML2 as default and specify SRS in the XML or normal format. These formats effect the longitude/latitude (x/y) order of the returned data and are further described below.

Normal—Returns the typical EPSG number, `EPSG:XXXX`. This formats the geographic coordinates in longitude/latitude (x/y) order.

XML—Returns a URL that identifies each EPSG code: `http://www.opengis.net/gml/srs/epsg.xml#XXXX`. This formats the geographic coordinates in longitude/latitude (x/y) order.

URN—(WFS 1.1.1 only) Returns the colon delimited SRS formatting: `urn:x-ogc:def:crs:EPSG:XXXX`. This formats data in the traditional axis order for geographic and cartographic systems—latitude/longitude (y/x).

5.5.3 WMS

The Web Map Service (WMS) page supports the configuration of raster rendering and SVG options.

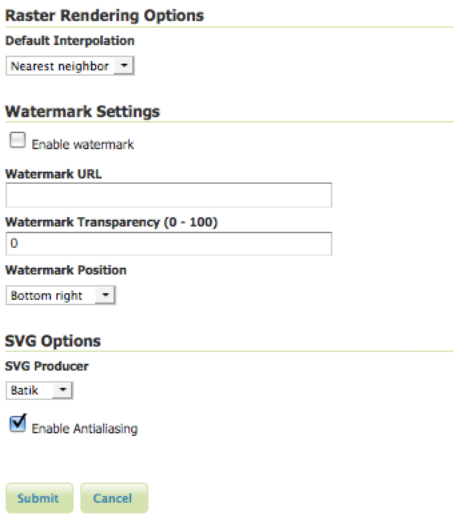


Figure 5.61: *WMS configuration options*

Service Metadata

See the section on *Service Metadata*.

Raster Rendering Options

The Web Map Service Interface Standard (WMS) provides a simple way to request and serve geo-registered map images. During pan and zoom operations, WMS requests generate map images through a variety of raster rendering processes. Such image manipulation is generally called resampling, interpolation, or down-sampling. GeoServer supports three resampling methods that determine how cell values of a raster are outputted. These sampling methods—Nearest Neighbor, Bilinear Interpolation and Bicubic—are available on the Default Interpolation menu.

Nearest Neighbor—Uses the center of nearest input cell to determine the value of the output cell. Original values are retained and no new averages are created. Because image values stay exactly the same, rendering is fast but possibly pixelated from sharp edge detail. Nearest neighbor interpolation is recommended for categorical data such as land use classification.

Bilinear—Determines the value of the output cell based by sampling the value of the four nearest cells by linear weighting. The closer an input cell, the higher its influence of on the output cell value. Since output values may differ from nearest input, bilinear interpolation is recommended for continuous data like elevation and raw slope values. Bilinear interpolation takes about five times as long as nearest neighbor interpolation.

Bicubic—Looks at the sixteen nearest cells and fits a smooth curve through the points to find the output value. Bicubic interpolation may both change the input value as well as place the output value outside of the range of input values. Bicubic interpolation is recommended for smoothing continuous data, but this incurs a processing performance overhead.

Watermark Settings

Watermarking is the process of embedding an image into a map. Watermarks are usually used for branding, copyright, and security measures. Watermarks are configured in the WMS watermarks setting section.

Enable Watermark—Turns on watermarking. When selected, all maps will render with the same watermark. It is not currently possible to specify watermarking on a per-layer or per-feature basis.

Watermark URL—Location of the graphic for the watermark. The graphic can be referenced as an absolute path (e.g., `C:GeoServerwatermark.png`), a relative one inside GeoServer's data directory (e.g., `watermark.png`), or a URL (e.g., `http://www.example.com/images/watermark.png`).

Each of these methods have their own advantages and disadvantages. When using an absolute or relative link, GeoServer keeps a cached copy of the graphic in memory, and won't continually link to the original file. This means that if the original file is subsequently deleted, GeoServer won't register it missing until the watermark settings are edited. Using a URL might seem more convenient, but it is more I/O intensive. GeoServer will load the watermark image for every WMS request. Also, should the URL cease to be valid, the layer will not properly display.

Watermark Transparency–Determines the opacity level of the watermark. Numbers range between 0 (opaque) and 100 (fully invisible).

Watermark Position—Specifies the position of the watermark relative to the WMS request. The nine options indicate which side and corner to place the graphic (top-left, top-center, top-right, etc). The default watermark position is bottom-right. Note that the watermark will always be displayed flush with the boundary. If extra space is required, the graphic itself needs to change.

Because each WMS request renders the watermark, a single tiled map positions *one* watermark relative to the view window while a tiled map positions the watermark for each tile. The only layer specific aspect of watermarking occurs because a single tile map is one WMS request, whereas a tiled map contains many WMS requests. (The latter watermark display resembles Google Maps faint copyright notice in their Satellite imagery.) The following three examples demonstrate watermark position, transparency and tiling display, respectively.

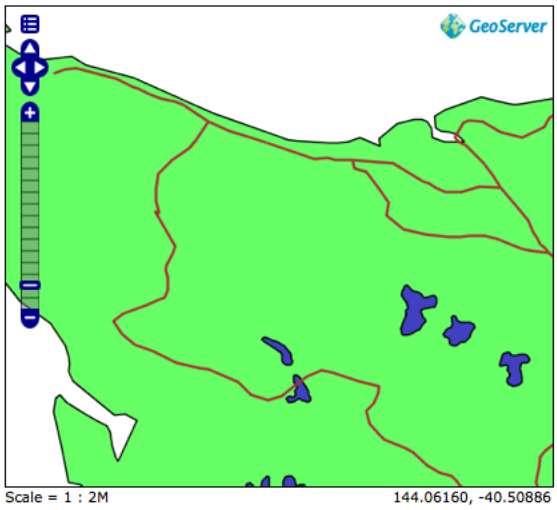


Figure 5.62: *Single tile watermark (aligned top-right, transparency=0)*

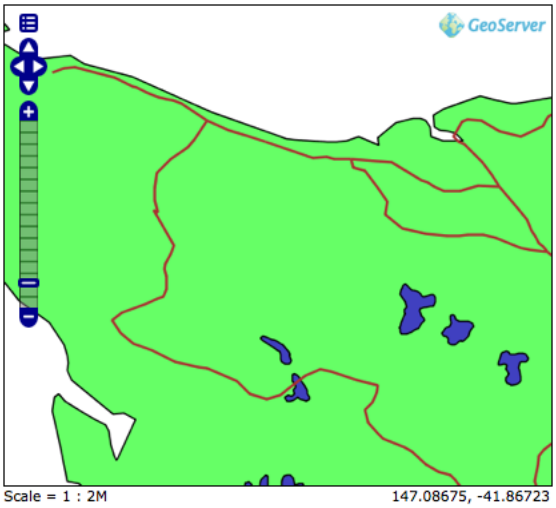


Figure 5.63: *Single tile watermark (aligned top-right, transparency=90)*

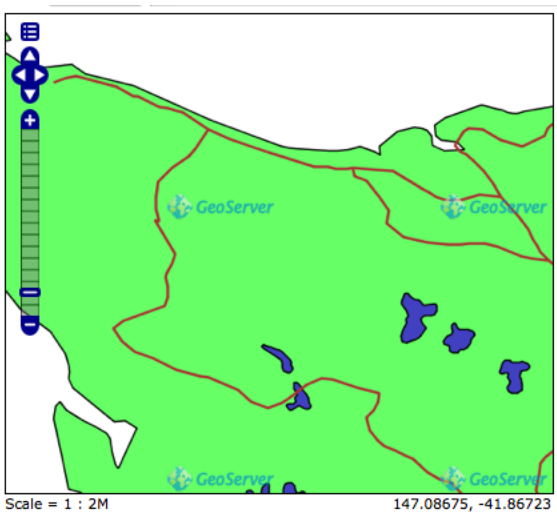


Figure 5.64: *Tiled watermark (aligned top-right, transparency=90)*

SVG Options

The GeoServer WMS supports SVG (Scalable Vector Graphics) as an output format. GeoServer currently supports two SVG renderers, available from the SVG producer menu.

1. *Simple*—Simple SVG renderer. It has limited support for SLD styling, but is very fast.
2. *Batik*—Batik renderer (as it uses the Batik SVG Framework). It has full support for SLD styling, but is slower.

Enable Anti-aliasing Anti-aliasing is a technique for making edges appear smoother by filling in the edges of an object with pixels that are between the object's color and the background color. Anti-aliasing creates the illusion of smoother lines and smoother selections. Turning on anti-aliasing will generally make maps look nicer, but will increase the size of the images, and will take longer to return. If you are overlaying the anti-aliased map on top of others, beware of using transparencies as the anti-aliasing process mixes with the colors behind and can create a "halo" effect.

5.6 Tile Caching

This section of the *Web Administration Interface* describes how to configure the tile caching options for GeoServer. GeoServer uses GeoWebCache to provide direct and integrated tile caching, and can dramatically increase your server's responsiveness and reliability.

For more information on GeoServer's integrated tile cache, please see the section on *Caching with GeoWebCache*.

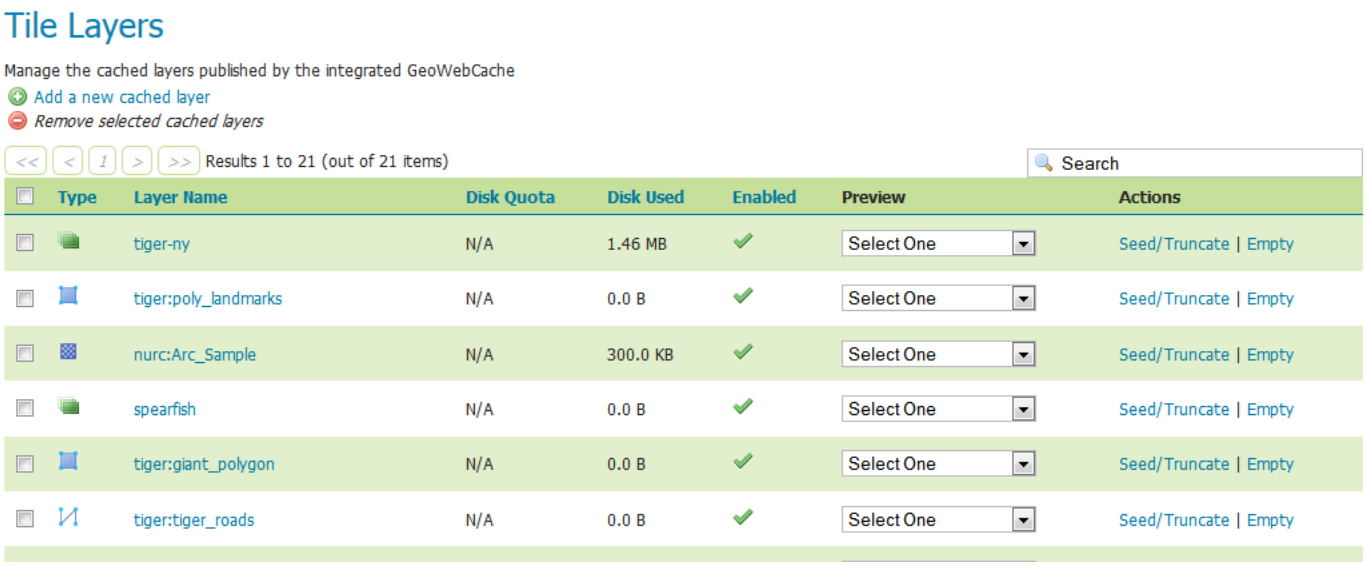
The pages in this menu can be accessed on the left side of the screen under the heading *Tile Caching*.

5.6.1 Tile Layers

This page shows a listing of all of the layers known to the integrated GeoWebCache. It is similar to the *Layer Preview* for GeoWebCache, with many of the same options.

Note: There is also a link to the *GeoWebCache standalone demo page <webadmin_tilecaching_demopage>*.

Figure 5.65: *Tile Caching menu*



Layer information

For each layer cached by GeoWebCache, the following information is available.

Disk Quota

The maximum amount of disk space that can be used for this layer. By default, this will be set to *N/A* (unbounded) unless *Disk Quotas* are enabled.

Disk Used

The current disk space being used by tiles for this particular layer.

Enabled

Indicates whether tile caching is enabled for this layer. It is possible to have a layer definition here but to not have tile caching enabled (set in the layer properties).

Preview

Similar to *Layer Preview*, this will generate a simple OpenLayers application populated with tiles from one of the available gridset/image format combinations. Select the desired option from the menu to view in OpenLayers.

Seed/Truncate

Opens the GeoWebCache page for automatically seeding and truncating the tile cache. Use this if you want to pre-populate some of your cache.

Empty

Will remove all saved tiles from the cache. This is identical to a full truncate operation for the layer.

Add or remove cached layers

The list of layers displayed on this page is typically the same as, or similar to, the full list of layers known to GeoServer. However, it may not be desirable to have every layer published in GeoServer have a cached layer component. In this case, simply select the box next to the layer to remove, and click *Remove selected cached layers*. The layer will be removed from GeoWebCache, and the disk cache for this layer will be entirely removed.

Warning: Deleting the tile cache cannot be undone.

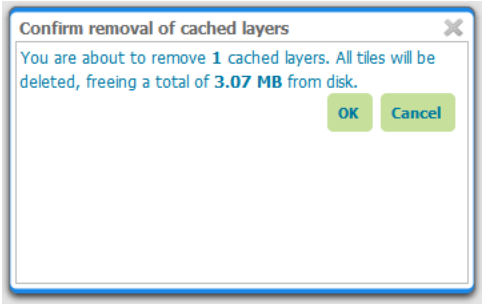


Figure 5.66: *Removing a cached layer*

To add in a layer from GeoServer (if it wasn't set up to be added automatically), click the *Add a new cached layer* link.

You have two options for layer configuration. The first option is to load the layer using the default (global) settings. To do this, select the layer you wish to start caching, and click the *Configure selected layers with caching defaults* link. The second option is to configure the caching parameters manually, via the *layer configuration* pages. To do this, just click the layer name itself.

5.6.2 Demo page

In addition to the *Tile Layers* page, there is also a demo page where you can view configured layers, reload the configuration (when changing settings or adding new layers), and seed or refresh the existing cache on

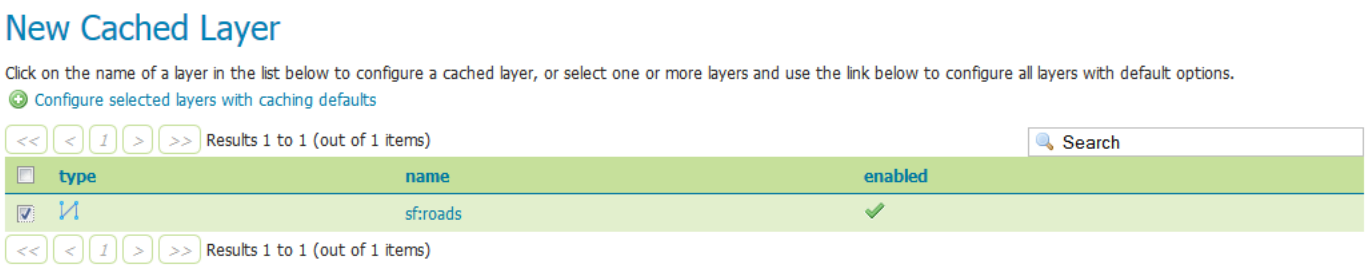


Figure 5.67: *Adding a new cached layer*

a per-layer basis.

As this interface is part of the standalone GeoWebCache, some of the functionality here is duplicated from the *Tile Layers* page.

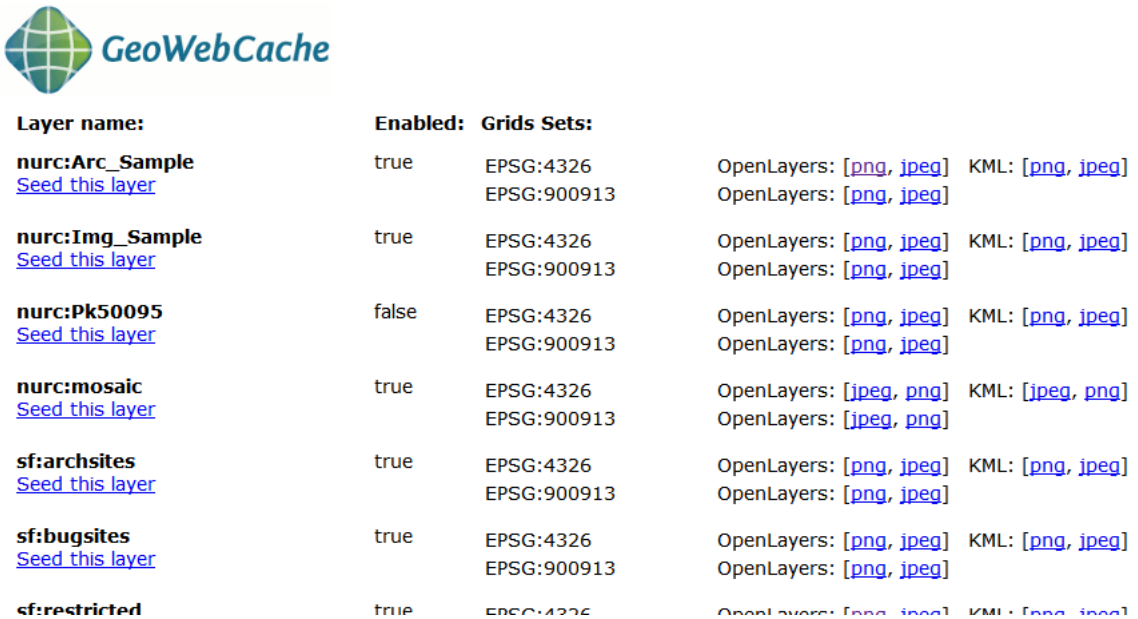


Figure 5.68: *Built-in demo page*

Viewing

To view the demo page, append `/gwc/demo` to the address of your GeoServer instance. For example, if your GeoServer is at the following address:

```
http://localhost:8080/geoserver
```

The GeoWebCache demo page is accessible here:

```
http://localhost:8080/geoserver/gwc/demo
```

If there is a problem loading this page, verify the steps on the *Using GeoWebCache* page have been carried out successfully.

Reload configuration

The demo page contains a list of every layer that GeoWebCache is aware of. This is typically (though not necessarily) identical to the list of layers as published in the GeoServer WMS capabilities document. If configuration changes are made to GeoServer, GeoWebCache will not automatically become aware of them. To ensure that GeoWebCache is using the latest configuration information, click the **Reload Configuration** button. Reloading the configuration will trigger authentication to GeoServer, and will require an administration username and password. Use the same username and password that you would use to log on to the *Web Administration Interface*. (See *Interface basics* for more information.) After a successful logon, the number of layers found and loaded will be displayed.

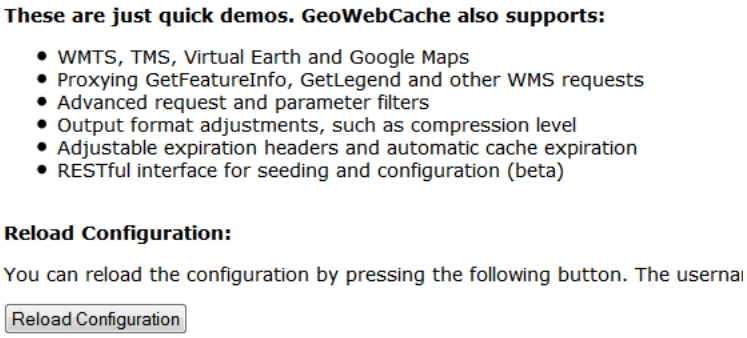


Figure 5.69: *Reloading the configuration*

Layers and output formats

For each layer that GeoWebCache serves, links are typically available for a number of different projections and output formats. By default, **OpenLayers** applications are available using image formats of PNG, PNG8, GIF, and JPEG in both **EPSG:4326** (standard lat/lon) and **EPSG:900913** (used in Google Maps) projections. In addition, **KML output** is available (EPSG:4326 only) using the same image formats, plus vector data ("kml").

Also on the list is an option to seed the layers (*Seed this layer*). More on this option below.

Seeding

You can configure seeding processes via the *Web Administration Interface*. See the *Tile Layers* page for more information.

It is also possible to configure seeding process via the *Demo page*. The page contains a link next to each layer entitled *Seed this layer*. This link will trigger authentication with the GeoServer configuration. Use the same username and password that you would use to log on to the *Web Administration Interface*. (See *Interface basics* for more information.) After a successful logon, a new page shows up with seeding options.

The seeding options page contains various parameters for configuring the way that the layer is seeded.

Option	Description
`Number of threads to use`	Possible values are between **1** and **16**.
`Type of operation`	Sets the operation. There are three possible values: **Seed** (creates tiles, but does not overwrite existing ones), **Reseed** (like Seed, but overwrites existing tiles) and **Truncate** (deletes all tiles within the given parameters)
`SRS`	Specifies the projection to use when creating tiles (default values are **EPSG:4326** and **EPSG:900913**)
`Format`	Sets the image format of the tiles. Can be **application/vnd.google-earth.kml+xml** (Google Earth KML), **image/gif** (GIF), **image/jpeg** (JPEG), **image/png** (24 bit PNG), and **image/png8** (8 bit PNG)
`Zoom start`	Sets the minimum zoom level. Lower values indicate map views that are more zoomed out. When seeding, GeoWebCache will only create tiles for those zoom levels inclusive of this value and `Zoom stop`.
`Zoom stop`	Sets the maximum zoom level. Higher values indicate map views that are more zoomed in. When seeding, GeoWebCache will only create tiles for those zoom levels inclusive of this value and `Zoom start`.
`Bounding box`	*(optional)* Allows seeding to occur over a specified extent, instead of the full extent of the layer. This is useful if your layer contains data over a large area, but the application will only request tiles from a subset of that area. The four boxes correspond to **Xmin**, **Ymin**, **Xmax**, and **Ymax**.

Warning: Currently there is no progress bar to inform you of the time required to perform the operation, nor is there any intelligent handling of disk space. In short, the process may take a *very* long time, and the cache may fill up your disk. You may wish to set a *Disk quota* before running a seed job.

5.6.3 Caching defaults

The Caching Defaults page shows the global configuration options for the tile caching functionality in GeoServer, an embedded GeoWebCache.

Note: For more information about this embedded version, please see the section on *Caching with GeoWebCache*.

GWC Provided Services

In addition to the GeoServer endpoints, GeoWebCache provides other endpoints for OGC services. For example, the GeoServer WMS endpoint is available at:

```
http://GEOSERVER_URL/wms?...
```

The GeoWebCache WMS endpoint is:

```
http://GEOSERVER_URL/gwc/service/wms?...
```

The following settings describe the different services that can be enabled with GeoWebCache.

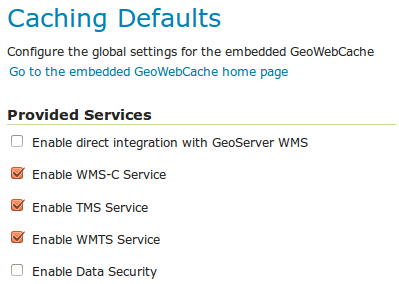


Figure 5.70: *Provided services*

Enable direct integration with GeoServer WMS

Direct integration allows WMS requests served through GeoServer to be cached as if they were received and processed by GeoWebCache. This provides all the advantages of using a tile server while still employing the more-flexible GeoServer WMS as a fallback. See the section on *Using GeoWebCache* for more details about this feature.

With direct integration, tile caching is enabled for all standard WMS requests that contain the `tiled=true` parameter and conform to all required parameters.

This setting is disabled by default. When enabling this option, it is a good idea to also turn on *Disk Quotas* as well, to prevent unbounded growth of the stored tiles.

Enable WMS-C Service

Enables the Cached Web Map Service (WMS-C) service. When this setting is enabled, GeoWebCache will respond to its own WMS-C endpoint:

```
http://GEOSERVER_URL/gwc/service/wms?SERVICE=WMS&VERSION=1.1.1&TILED=true&...
```

When the service is disabled, calls to the capabilities document will return a `Service is disabled` message.

Enable TMS Service

Enables the Tiled Map Service (TMS) endpoint in GeoWebCache. With the TMS service, GeoWebCache will respond to its own TMS endpoint:

```
http://GEOSERVER/URL/gwc/service/tms/1.0.0
```

When the service is disabled, calls to the capabilities document will return a `Service is disabled` message.

Enable WMTS Service

Enables the Web Map Tiled Service (WMTS) endpoint in GeoWebCache. When this setting is enabled, GeoWebCache will respond to its own WMTS endpoint:

```
http://GEOSERVER/URL/gwc/service/wmts?...
```

When the service is disabled, calls to the capabilities document will return a `Service is disabled` message.

Enable Data Security

Enables the *Geoserver Data Security* in the embedded GeoWebCache.

Default Caching Options for GeoServer Layers

This section describes the configuration of the various defaults and other global options for the tile cache in GeoServer.

Figure 5.71: *Default caching options*

Automatically configure a GeoWebCache layer for each new layer or layer group

This setting, enabled by default, determines how layers in GeoServer are handled via the embedded GeoWebCache. When this setting is enabled, an entry in the GeoWebCache layer listing will be created whenever a new layer or layer group is published in GeoServer. Use this setting to keep the GeoWebCache catalog in sync. (This is enabled by default.)

Automatically cache non-default styles

By default, only requests using the default style for a given layer will be cached. When this setting is enabled, all requests for a given layer, even those that use a non-standard style will be cached. Disabling this may be useful in situations where disk space is an issue, or when only one default style is important.

Default metatile size

A metatile is several tiles combined into a larger one. This larger metatile is generated and then subdivided before being served back (and cached) as standard tiles. The advantage of using metatiling is in situations where a label or geometry lies on a boundary of a tile, which may be truncated or altered. With metatiling, these tile edge issues are greatly reduced.

Moreover, with metatiling, the overall time it takes to seed the cache is reduced in most cases, when compared with rendering a full map with single tiles. In fact, using larger metatiling factors is a good way to reduce the time spent in seeding the cache.

The disadvantage of metatiling is that at large sizes, memory consumption can be an issue.

The size of the default metatile can be adjusted here. By default, GeoServer sets a metatile size of **4x4**, which strikes a balance between performance, memory usage, and rendering accuracy.

Default gutter size

The gutter size sets the amount of extra space (in pixels) used when generating a tile. Use this in conjunction with metatiles to reduce problems with labels and features not being rendered incorrectly due to being on a tile boundary.

Default Cache Formats

This setting determines the default image formats that can be cached when tiled requests are made. There are four image formats that can be used when saving tiles:

- PNG (24-bit PNG)
- PNG8 (8-bit PNG)
- JPEG
- GIF

The default settings are subdivided into vector layers, raster layers, and layer groups. You may select any of the above four formats for each of the three types of layers. Any requests that fall outside of these layer/format combinations will not be cached if sent through GeoServer, and will return an error if sent to the GeoWebCache endpoints.

These defaults can be overwritten on a per-layer basis when *editing the layer properties*.

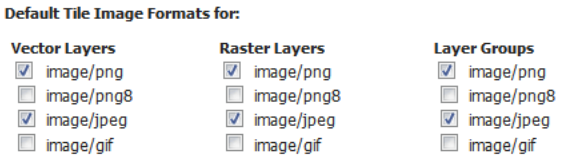


Figure 5.72: *Default image formats*

Default Cached Gridsets

This section shows the gridsets that will be automatically configured for cached layers. While there are some pre-configured gridsets available, only two are enabled by default. These correspond to the most common and universal cases:

- EPSG:4326 (geographic) with 22 maximum zoom levels and 256x256 pixel tiles
- EPSG:900913 (spherical Mercator) with 31 maximum zoom levels and 256x256 pixel tiles

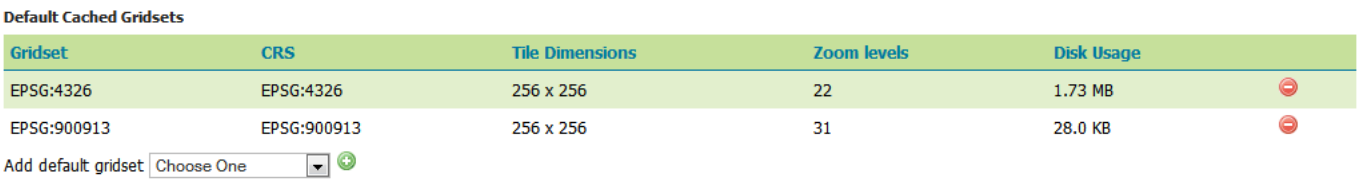


Figure 5.73: *Default gridsets*

To add a pre-existing grid set, select it from the *Add default grid set* menu, and click the Add icon (green circle with plus sign).

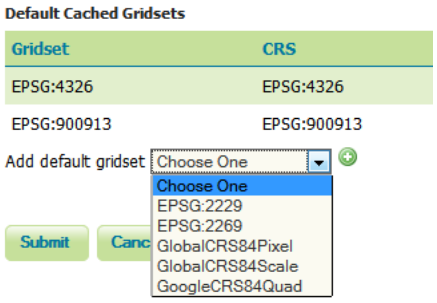


Figure 5.74: *Adding an existing gridset to the list of defaults*

These definitions are described in more detail on the *Gridsets* page.

5.6.4 Gridsets

A gridset defines a spatial reference system, bounding box (extent), a list of zoom levels (resolutions or scale denominators), and tile dimensions. Tile requests must conform to the gridset matrix, otherwise caching will not occur.

This page allows you to edit existing saved gridsets or create new ones. There are five preconfuigred gridsets, all in one of two coordinate reference systems: EPSG:4326 and EPSG:900913. For additional CRS support, new gridsets can be created. Another reason to create a new gridset would be to set a different tile size or different number of zoom levels.

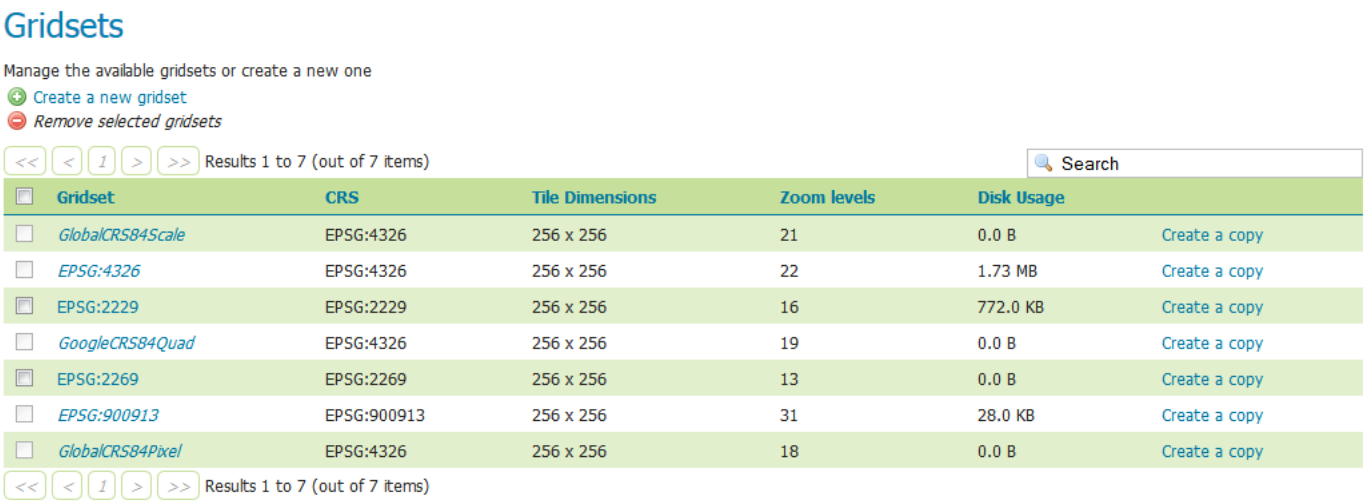


Figure 5.75: *Gridsets menu*

Creating a new gridset

To create a new gridset, click *Create new gridset*. You will then be asked to enter a range of parameters.

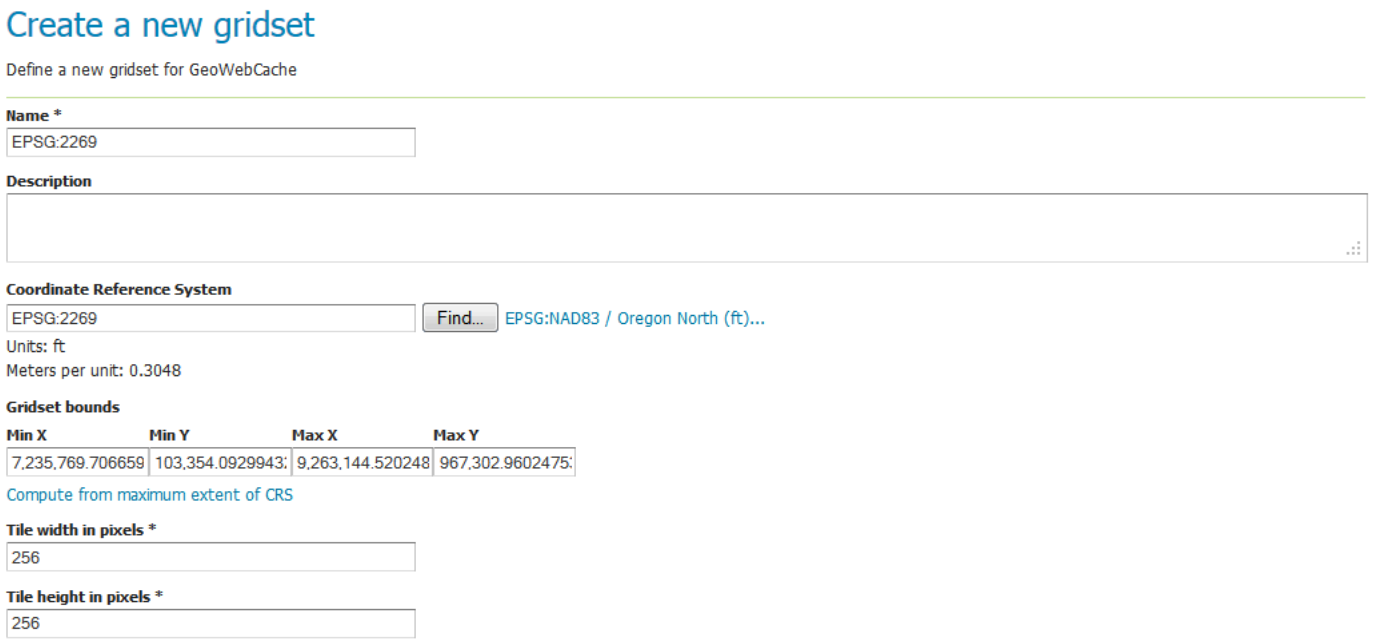


Figure 5.76: *Creating a new gridset*

Name

The short name of the new gridset.

Description

Metadata on the gridset.

Coordinate Reference System

The Coordinate Reference System (CRS) to use in the gridset. You can select from any CRS that GeoServer recognizes. After selection, both the units (meters, feet, degrees, etc.) and the number of meters per unit will be displayed.

Gridset bounds

Sets the maximum extent for the gridset. Typically this is set to be the maximum extent of the CRS used, but a smaller value can be substituted if desired. To populate the max extent in the fields, click *Compute from maximum extent of CRS*.

Tile width and height

Sets the tile dimensions. Default is **256x256 pixels**. The tile dimensions can be anything from 16 to 2048 pixels. In addition, the tiles need not be square.

Tile matrix set

The tile matrix set (or tile pyramid) is a list of zoom levels containing ever increasing amounts of tiles. This three dimensional collection of tile "slots" creates the framework where actual image tiles will be saved. You can define the tile matrix based on resolutions or scale denominators.

Click *Add zoom level* to generate the first zoom level. The parameters will be automatically configured such that the full extent of will be contained by a single pixel's height. The number of pixels in a given zoom level will be displayed, along with the Pixel Size, Scale, and an optional Name, where you can give a name to each zoom level if desired.

Typically each additional zoom level is twice as large in each dimension, and so contains four times as many tiles as the previous zoom level. The actual values will be populated automatically during subsequent clicking of the *Add zoom level* link. These defaults are usually sufficient, and you need only determine the maximum number of zoom levels desired for this gridset.

When finished, click *Save*. Before you will be able to use this new gridset with a layer, you will need to add this gridset to the layer's list of available gridsets. This is done on an individual layer's *properties* page. You can also add this gridset to the default list on the *Caching defaults* page.

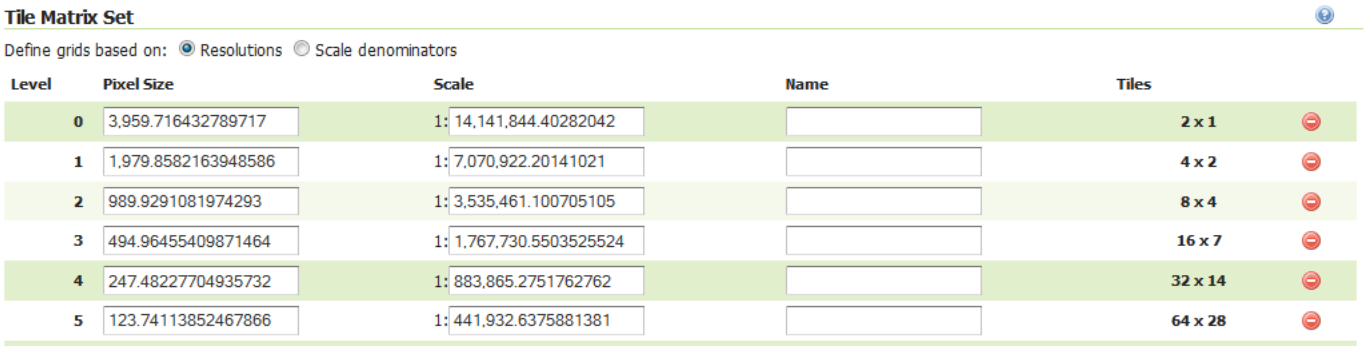


Figure 5.77: *Tile matrix set*

Editing a gridset

Click an existing gridset to open it for editing. Please note that the built-in gridsets cannot be edited. They can, however, be copied.

Copying a gridset

As there are many configuration options for a gridset, it is often more convenient to copy an existing gridset. For any of the existing gridsets, click the *Create a copy* link to copy the gridset information to a new gridset.

Removing a gridset

To remove a gridset, select the check box next to the gridset or gridsets, and click *Remove selected gridsets*.

> **Warning:** Removing a gridset definition will remove not only the gridset definition, but also any tiles on any layers generated with this gridset.

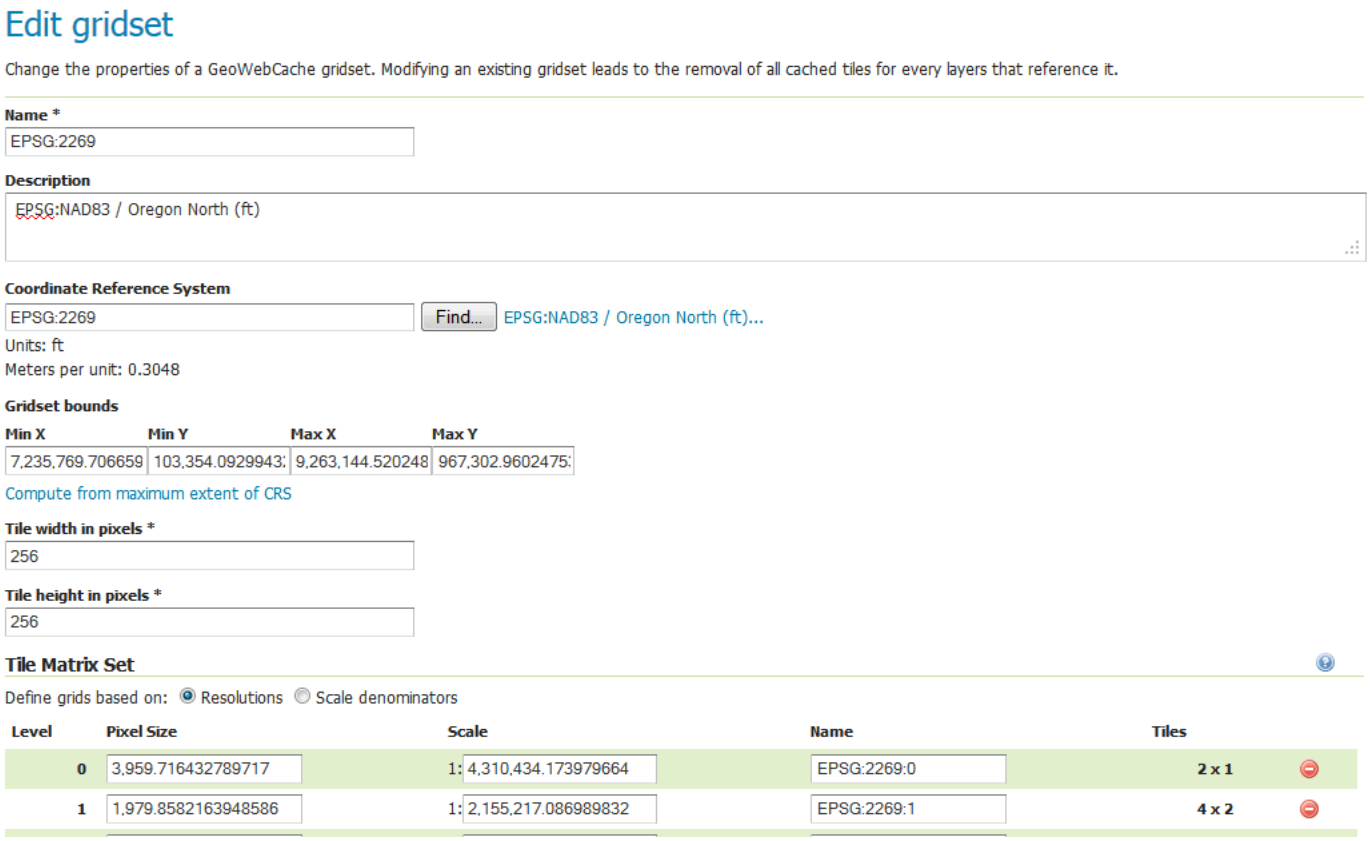


Figure 5.78: *Editing a gridset*

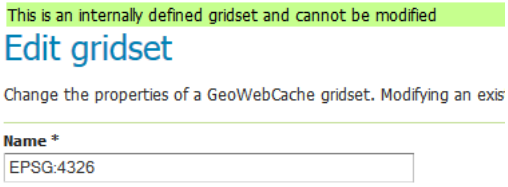


Figure 5.79: *This gridset is read-only*

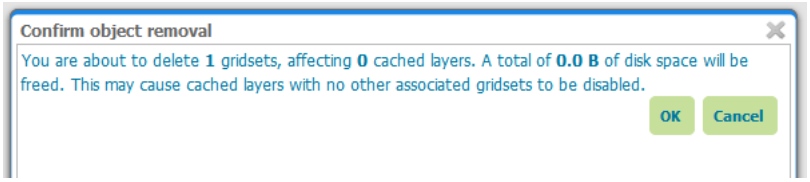


Figure 5.80: *Removing a gridset*

5.6.5 Disk Quotas

The Disk Quotas page manages the disk usage for cached tiles and allows you to set the global disk quota. Individual layer quotas can be set in the layer's *properties* page.

By default, disk usage for cached tiles is unbounded. However, this can cause disk capacity issues, especially when using Direct WMS integration (see *Disk Quotas* for more on this). Setting a disk quota establishes disk usage limits.

When finished making any changes, remember to click *Submit*.

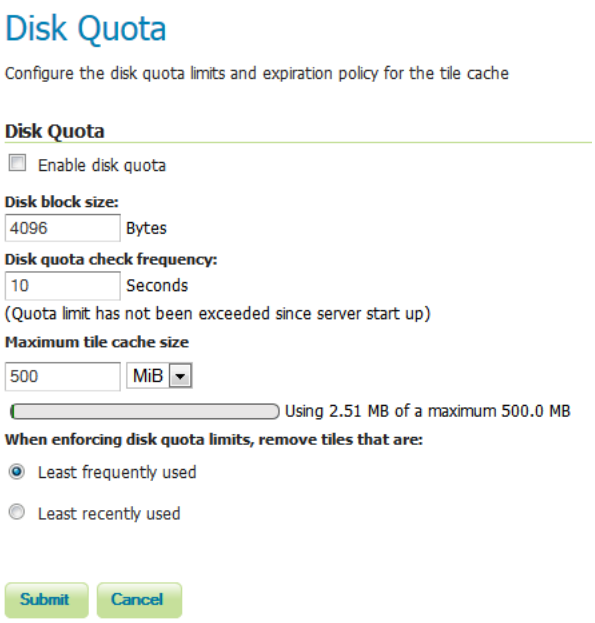


Figure 5.81: *Disk quota*

Enable disk quota

When enabled, the disk quota will be set according to the options listed below. The setting is disabled by default.

Disk block size

This setting determines how the tile cache calculates disk usage. The value for this setting should be equivalent to the disk block size of the storage medium where the cache is located. The default block size is **4096 bytes**.

Disk quota check frequency

This setting determines how often the cache is polled for any overage. Smaller values (more frequent polling) will slightly increase disk activity, but larger values (less frequent polling) may cause the disk quota to be temporarily exceeded. The default is **10 seconds**.

Maximum tile cache size

The maximum size for the cache. When this value is exceeded and the cache is polled, tiles will be removed according to the policy. Note that the unit options are **mebibytes (MiB)** (approx. 1.05MB), **gibibytes (GiB)** (approx. 1.07GB), and **tebibytes (TiB)** (approx. 1.10TB). Default is **500 MiB**.

The graphic below this setting illustrates the size of the cache relative to the disk quota.

Tile removal policy

When the disk quota is exceeded, this policy determines how the tiles to be deleted are identified. Options are **Least Frequently Used** (removes tiles based on how often the tile was accessed) or **Least Recently Used** (removes tiles based on date of last access). The optimum configuration is dependent on your data and server usage.

5.7 Security

GeoServer has a robust *security subsystem*, modeled on Spring Security. Most of the security features are available through the *Web Administration Interface*. This section describes how to configure GeoServer security.

5.7.1 Settings

The Settings page controls the global GeoServer security settings.

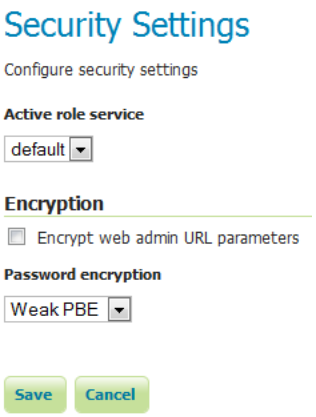


Figure 5.82: *Security Settings page*

Active role service

This option sets the active *role service* (provides information about roles). Role services are managed on the *Users, Groups, Roles* page. There can be only one active role service at one time.

Encryption

The GeoServer user interface (UI) can sometimes expose parameters in plain text inside the URLs. As a result, it may be desirable to encrypt the URL parameters. To enable encryption, select *Encrypt web admin URL parameters*. This will configure GeoServer to uses a PBE-based *Password encryption*.

For example, with this feature enabled, the page:

```
http://GEOSERVER/web/?wicket:bookmarkablePage=:org.geoserver.security.web.SecuritySettingsPage
```

would now be found at the following URL:

```
http://GEOSERVER/web/?x=hrTNYMcF3OY7u4NdyYnRanL6a1PxMdLxTZcY5xK5ZXyi617EFEFCagMwHBWhrlg*ujTOyd17DLSn(
```

Password encryption

This setting allows you to select the type of *Password encryption* used for passwords. The options are *Plain text*, *Weak PBE*, or *Strong PBE*.

If Strong PBE is not available as part of the JVM, a warning will display and the option will be disabled. To enable Strong PBE, you must install external policy JARs that support this form of encryption. See the section on *Password encryption* for more details about these settings.

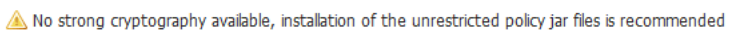


Figure 5.83: *Warning if Strong PBE is not available*

5.7.2 Authentication

This page manages the authentication options, including authentication providers and the authentication chain.

Anonymous authentication

By default, GeoServer will allow anonymous access to the *Web Administration Interface*. Without authentication, users will still be able to view the *Layer Preview*, capabilities documents, and basic GeoServer details. Anonymous access can be disabled by clearing the *Allow anonymous authentication* check box. Anonymous users navigating to the GeoServer page will get an HTTP 401 status code, which typically results in a browser-based request for credentials.

Note: Read more about *Authenticating to the Web Admin Interface*.

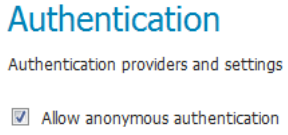


Figure 5.84: *Anonymous authentication checkbox*

Authentication providers

This section manages the *Authentication providers* (adding, removing, and editing). The default authentication provider uses basic *username/password authentication*. *JDBC* and *LDAP* authentication can also be used.

Click *Add new* to create a new provider. Click an existing provider to edit its parameters.

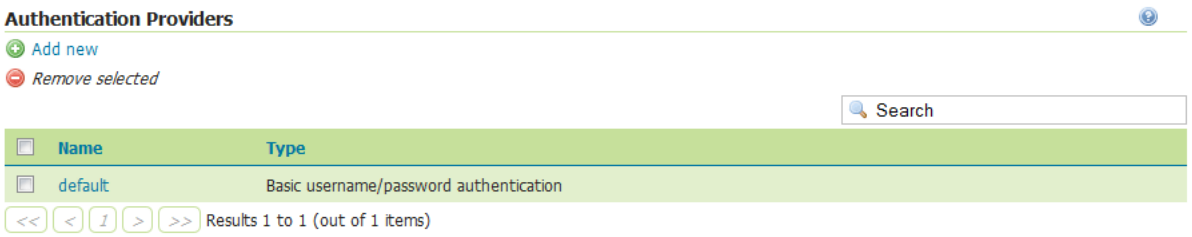


Figure 5.85: *List of authentication providers*

Username/password provider

The default new authentication provider uses a user/group service for authentication.

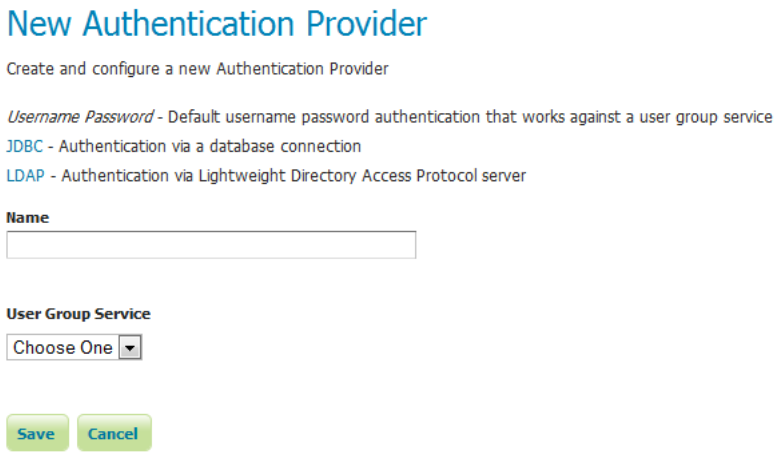


Figure 5.86: *Creating a new authentication provider with a username and password*

Option	Description
Name	Name of the provider
User Group Service	Name of the user/group service associated with this provider. Can be any one of the active user/group services.

JDBC provider

The configuration options for the JDBC authentication provider are illustrated below.

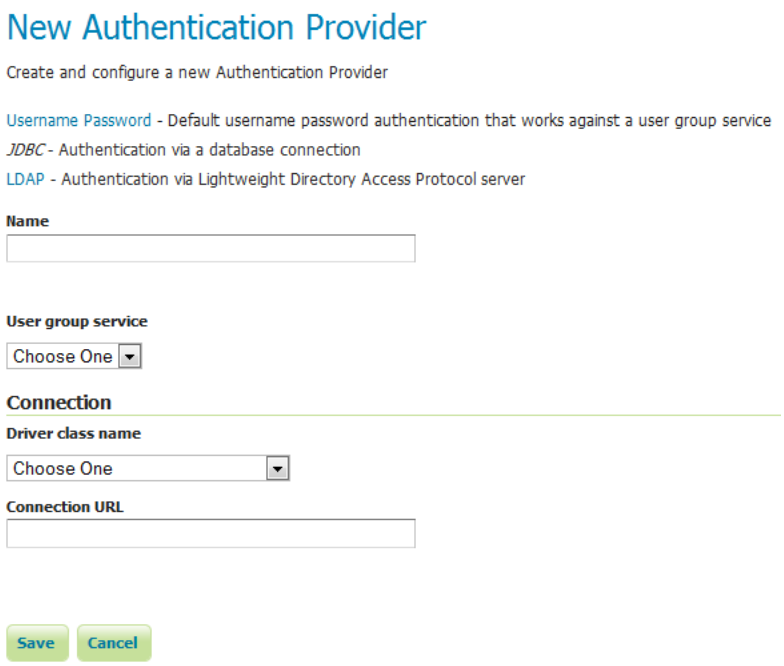


Figure 5.87: *Configuring the JDBC authentication provider*

Option	Description
Name	Name of the JDBC connection in GeoServer
User Group Service	Name of the user/group service to use to load user information after the user is authenticated
Driver class name	JDBC driver to use for the database connection
Connection URL	JDBC URL to use when creating the database connection

LDAP provider

The following illustration shows the configuration options for the LDAP authentication provider. The default option is to use LDAP groups for role assignment, but there is also an option to use a user/group service for role assignment. Depending on whether this option is selected, the page itself will have different options.

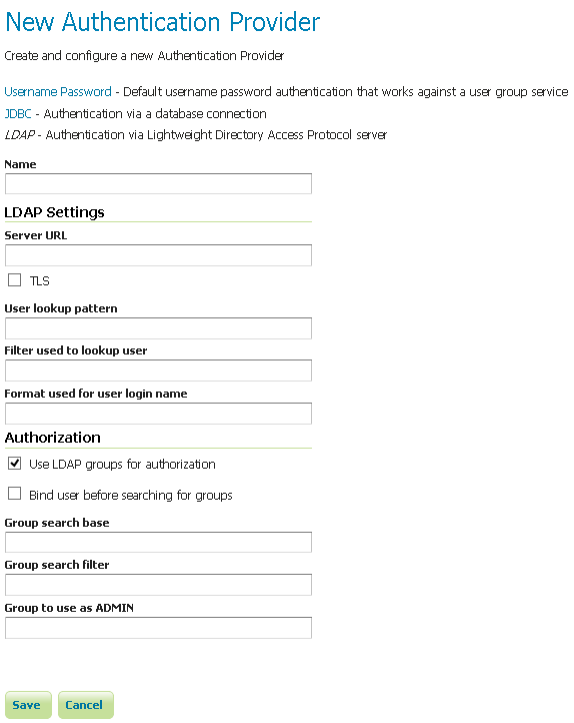


Figure 5.88: *Configuring the LDAP authentication provider using LDAP groups for role assignment*

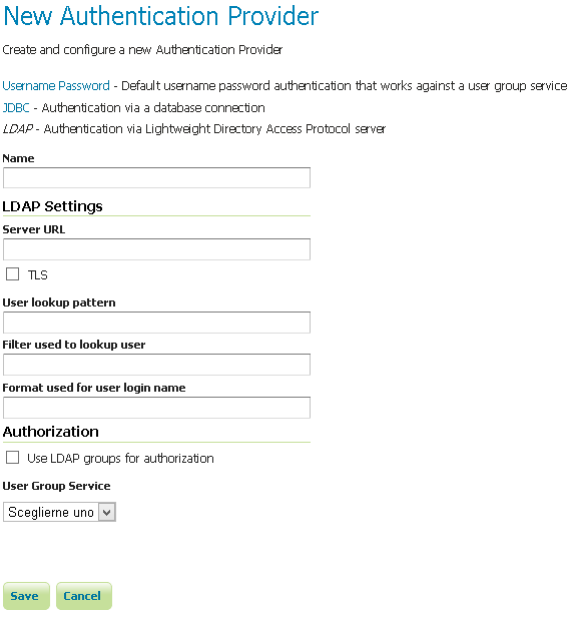


Figure 5.89: *Configuring the LDAP authentication provider using user/group service for authentication*

Option	Description
Name	Name of the LDAP connection in GeoServer
Server URL	URL for the LDAP server connection. It must include the protocol, host, and port, as well as the "distinguished name" (DN) for the root of the LDAP tree.
TLS	Enables a STARTTLS connection. (See the section on *Secure LDAP connections*.)
User DN pattern	Search pattern to use to match the DN of the user in the LDAP database. The pattern should contain the placeholder `{0}` which is injected with the `uid` of the user. Example: `uid={0},ou=people`. The root DN specified as port of the *Server URL* is automatically appended.
User Filter	LDAP Filter used to extract User data from LDAP database. Used alternatively to User DN pattern and combined with User Format to separate bind and user data extraction handling. Example: `(userPrincipalName={0})`. Gets user data searching for a single record matching the filter. This may contain two placeholder values: `{0}`, the full DN of the user, for example `uid=bob,ou=people,dc=acme,dc=com` `{1}`, the `uid` portion of the full DN, for example `bob`.
User Format	String formatter used to build username used for binding. Used alternatively to User DN pattern and combined with User Filter to separate bind and user data extraction handling. Example: `{0}@domain`. Binds user with the username built applying the format. This may contain one placeholder: `{0}`, the username, for example `bob`
Use LDAP groups for authorization	Specifies whether to use LDAP groups for role assignment
Bind before group search	Specifies whether to bind to LDAP server with the user credentials before doing group search
Group search base	Relative name of the node in the tree to use as the base for LDAP groups. Example: `ou=groups`. The root DN specified as port of the *Server URL* is automatically appended. Only applicable when the *Use LDAP groups for authorization(parameter is **checked**.*
Group search filter	Search pattern for locating the LDAP groups a user belongs to. This may contain two placeholder values: `{0}`, the full DN of the user, for example `uid=bob,ou=people,dc=acme,dc=com` `{1}`, the `uid` portion of the full DN, for example `bob`. Only applicable when the *Use LDAP groups for authorization(parameter is **checked**.*
Admin Group	Name of the group to be mapped to Administrator role (defaults to ADMINISTRATOR). Example: `ADMIN`. Adds the role ROLE_ADMINISTRATOR if the user belongs to a group named ADMIN (case insensitive)
Group Admin Group	Name of the group to be mapped to Group Administrator role (defaults to GROUP_ADMIN). Example: `GROUPADMIN`. Adds the role ROLE_GROUP_ADMIN if the user belongs to a group named GROUPADMIN (case insensitive)
User Group Service	The user/group service to use for role assignment. Only applicable when the *Use LDAP groups for authorization* parameter is **cleared**.

Authentication chain

This section selects the authentication chain. Currently, only one default authentication chain is available. For further information about the default chain, please refer to *Authentication chain*.

5.7.3 Passwords

This page configures the various options related to *Passwords*, the *Master password*, and *Password policies*.

Figure 5.90: *Selecting the authentication chain*

Note: User passwords may be changed in the Users dialog box accessed from the *Users, Groups, Roles* page.

Active master password provider

This option sets the active master password provider, via a list of all available master password providers.

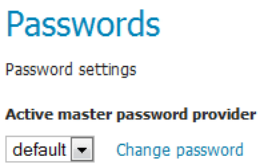


Figure 5.91: *Active master password provider*

To change the master password click the *Change password* link.

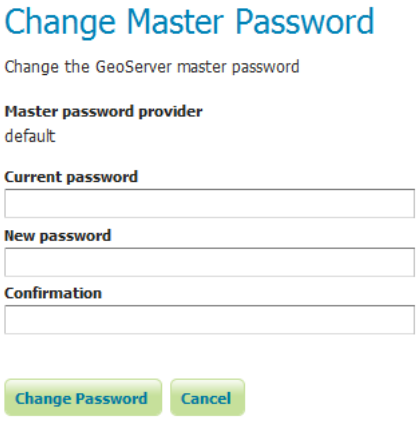


Figure 5.92: *Changing the master password*

Master Password Providers

This section provides the options for adding, removing, and editing master password providers.

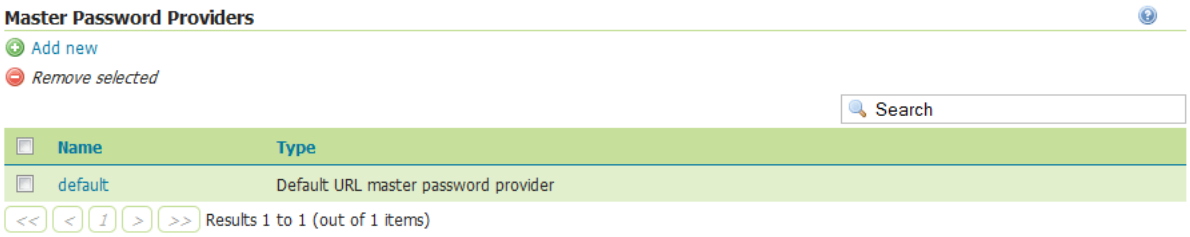


Figure 5.93: *Master password provider list*

Password policies

This section configures the various *Password policies* available to users in GeoServer. New password policies can be added or renamed, and existing policies edited or removed.

By default there are two password policies in effect, `default` and `master`. The `default` password policy, intended for most GeoServer users, does not have any active password constraints. The `master` password policy, intended for the *Root account*, specifies a **minimum password length of eight characters**. Password policies are applied to users via the user/group service.

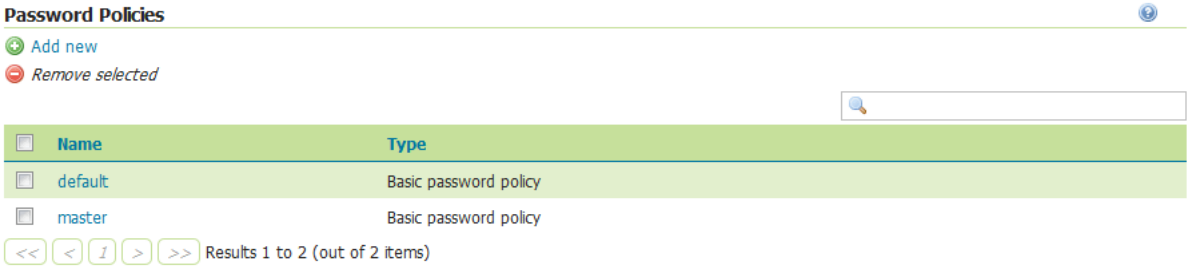


Figure 5.94: *List of password policies*

Clicking an existing policy enables editing, while clicking the *Add new* button will create a new password policy.

5.7.4 Users, Groups, Roles

This section provides the configuration options for *User/group services* and *Role services*. In addition, users, groups, and roles themselves and can be added, edited, or removed. A great deal of configuration can be accomplished in this section and related pages.

User Group Services

In this menu, user/group services can be added, removed, or edited. By default, there is one user/group service in GeoServer, which is *XML-based*. It is encrypted with *Weak PBE* and uses the default *password policy*. It is also possible to have a user/group service based on *JDBC*, with or without JNDI.

Clicking an existing user/group service will enable editing, while clicking the *Add new* link will configure a new user/group service.

There are three tabs for configuration: Settings, Users, and Groups.

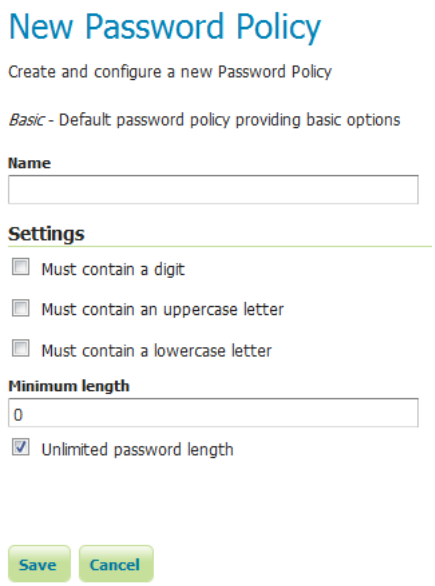


Figure 5.95: *Creating a new password policy*

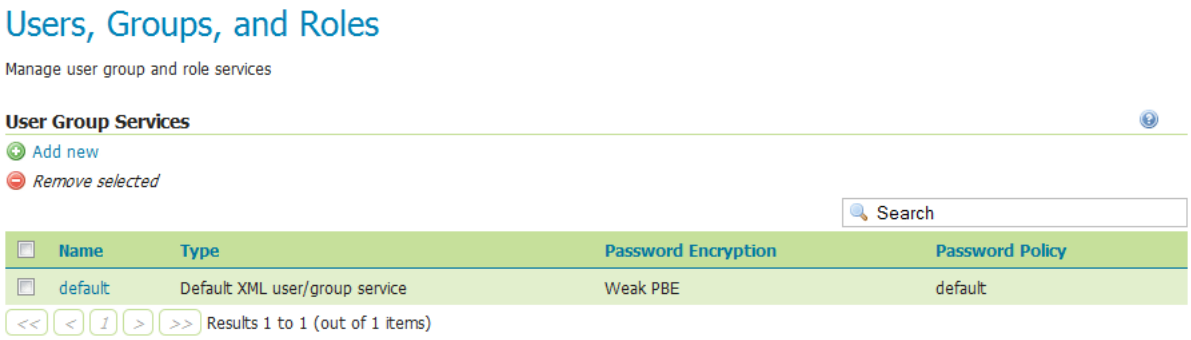


Figure 5.96: *User/group services*

Note: When creating a new user/group service, the form filled out initially can be found under the Settings tab.

Add new XML user/group service

To add a new XML user/group service, click the *Add new* link. XML is the default option. The following figure shows the configuration options for an XML user/group service.

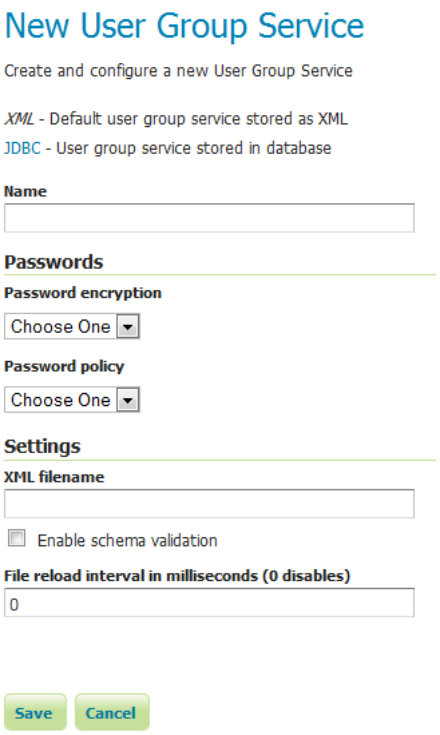


Figure 5.97: *Adding an XML user/group service*

Option	Description
Name	The name of the user/group service
Password encryption	Sets the type of *Password encryption*. Options are *Plain text*, *Weak PBE*, *Strong PBE*, and *Digest*.
Password policy	Sets the *password policy*. Options are any active password policies as set in the *Passwords* section.
XML filename	Name of the file that will contain the user and group information. Default is `users.xml` in the `security/usergroup/<name_of_usergroupservice>` directory.
Enable schema validation	If selected, forces schema validation to occur every time the XML file is read. This option is useful when editing the XML file by hand.
File reload interval	Defines the frequency (in milliseconds) in which GeoServer will check for changes to the XML file. If the file is found to have been modified, GeoServer will recreate the user/group database based on the current state of the file. This value is meant to be set in cases where the XML file contents might change "out of process" and not directly through the web admin interface. The value is specified in milliseconds. A value of 0 disables any checking of the file.

Add new JDBC user/group service

To add a new XML user/group service, click the *Add new* link, and then the *JDBC* option at the top of the following form. The following figure shows the configuration options for a JDBC user/group service.

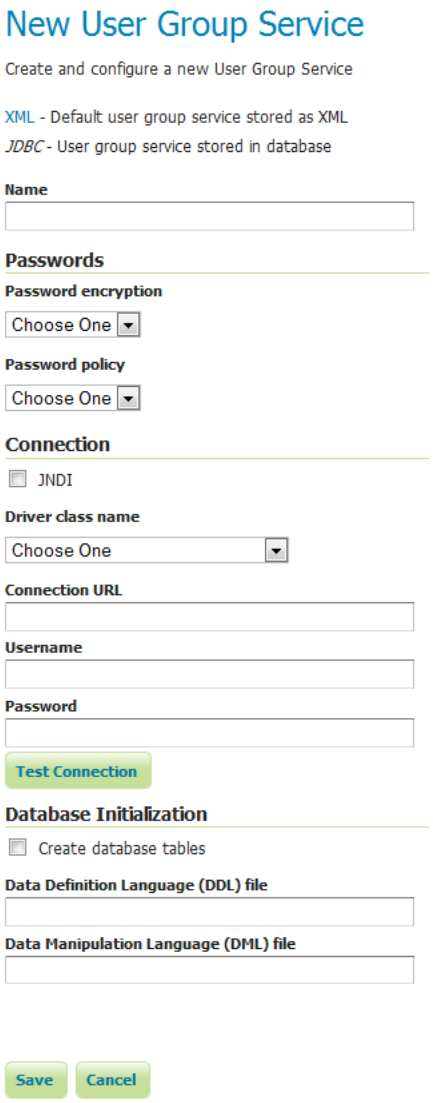


Figure 5.98: *Adding a user/group service via JDBC*

Option	Description
Name	Name of the JDBC user/group service in GeoServer
Password encryption	The method to used to *encrypt user passwords*
Password policy	The *policy* to use to enforce constraints on user passwords
JNDI	When unchecked, specifies a direct connection to the database. When checked, specifies an existing connection located through *JNDI*.
Driver class name	JDBC driver to use for the database connection
Connection URL	Specifies the JDBC URL to use when creating the database connection
Username	Username to use when connecting to the database
Password	Password to use when connecting to the database
Create database tables	Specifies whether to create all the necessary tables in the underlying database
Data Definition Language (DDL) file	Specifies a custom DDL file to use for creating tables in the underlying database, for cases where the default DDL statements fail on the given database. If left blank, internal defaults are used.
Data Manipulation Language (DML) file	Specifies a custom DML file to use for accessing tables in the underlying database, for cases where the default DML statements fail on the given database. If left blank, internal defaults are used.

In addition to the parameters listed above, the following additional parameter will apply when the *JNDI* flag is set.

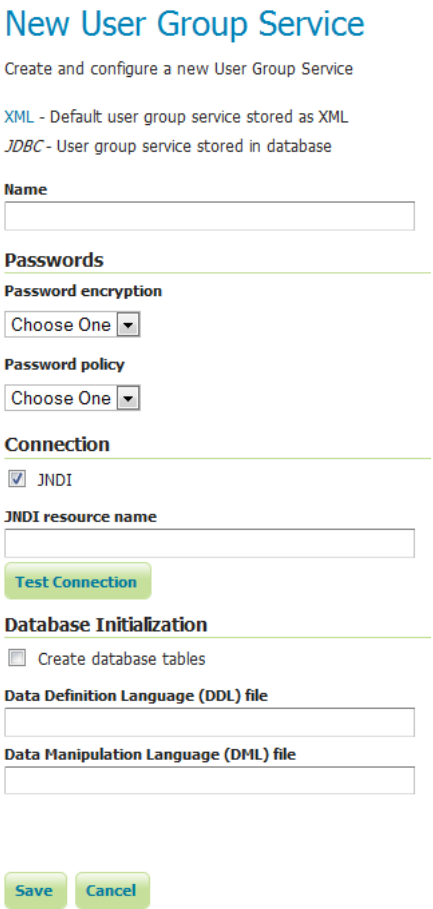


Figure 5.99: *Adding a user/group service via JDBC with JNDI*

Option	Description
JNDI resource name	JNDI name used to locate the database connection.

Edit user/group service

Once the new user/group service is added (either XML or JDBC), clicking on it in the list of user/group services will allow additional options to be specified, such as the users and groups associated with the service.

There are three tabs in the resulting menu: *Settings*, *Users*, and *Groups*. The Settings tab is identical to that found when creating the user/group service, while the others are described below.

The Users tab provides options to configure users in the user/group service.

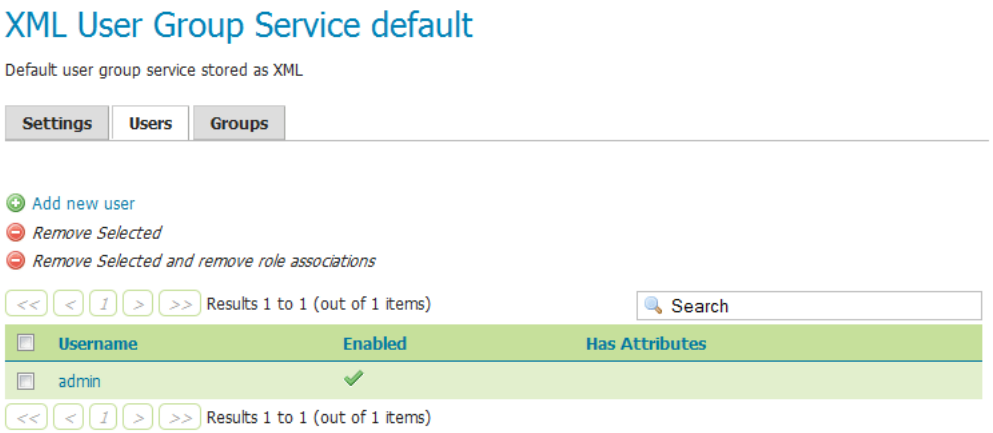


Figure 5.100: *Users tab*

Clicking a username will allow its parameters to be changed, while clicking the *Add new* link will create a new user.

Add user

Option	Description
User name	The name of the user
Enabled	When selected, will enable the user to authenticate
Password	The password for this user. Existing passwords will be obscured when viewed.
Confirm password	To set or change the password enter the password twice.
User properties	Key/value pairs associated with the user. Used for associating additional information with the user.
Group list	Full list of groups, including list of groups to which the user is a member. Membership can be toggled here via the arrow buttons.
Add a new group	Shortcut to adding a new group. Also available in the Groups tab.
Role list	Full list of roles, including a list of roles to which the user is associated. Association can be toggled here via the arrow buttons.
Add a new role	Shortcut to adding a new role
List of current roles for the user	List of current roles associated with the user. Click a role to enable editing.

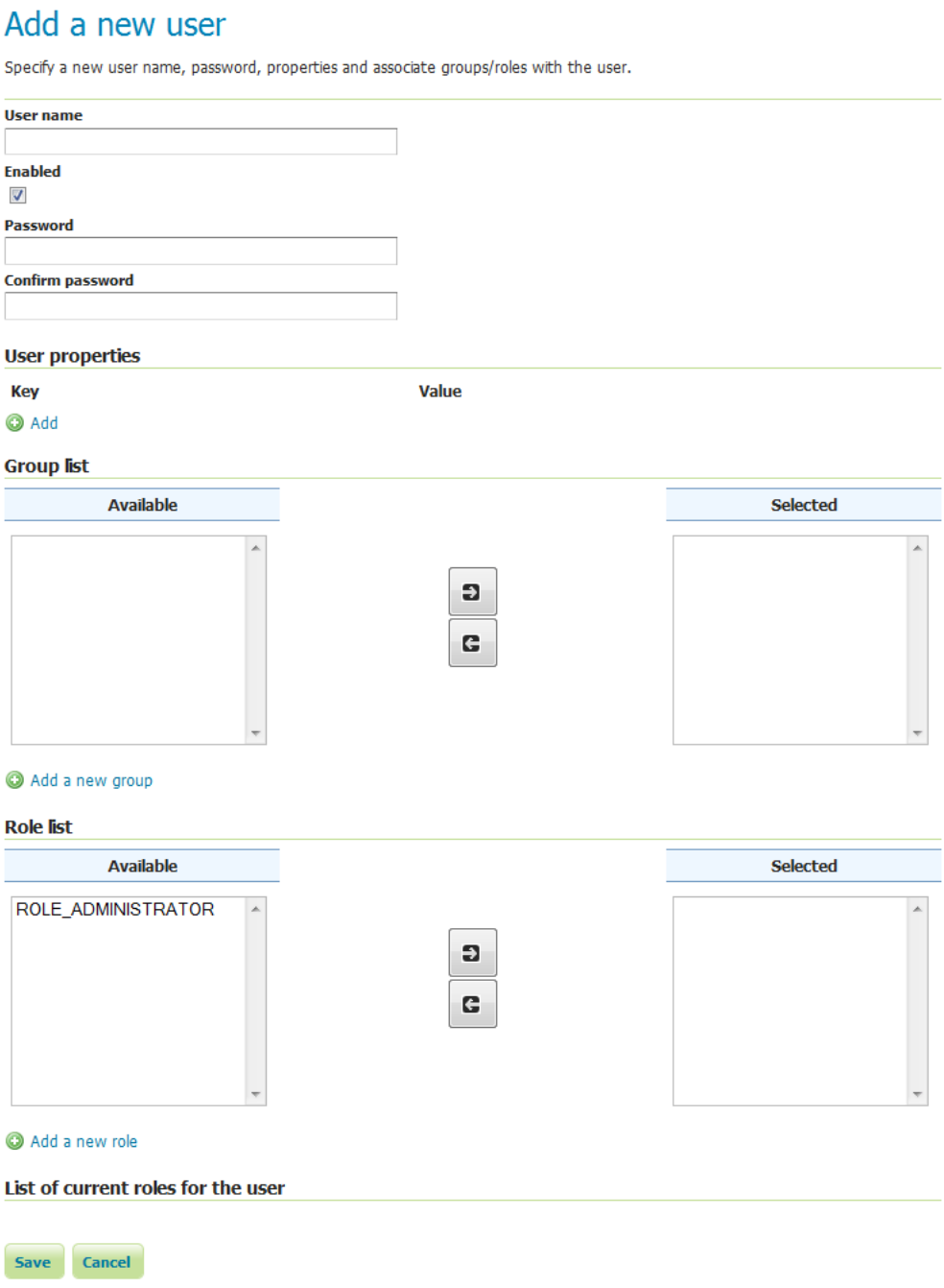


Figure 5.101: *Creating or editing a user*

The Groups tab provides configuration options for groups in this user/group service. There are options to add and remove a group, with an additional option to remove a group and the roles associated with that group.

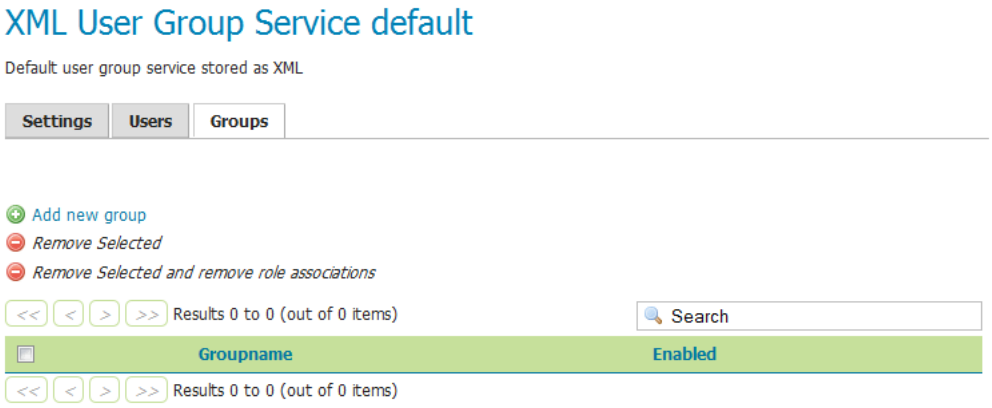


Figure 5.102: *Groups tab*

Add group

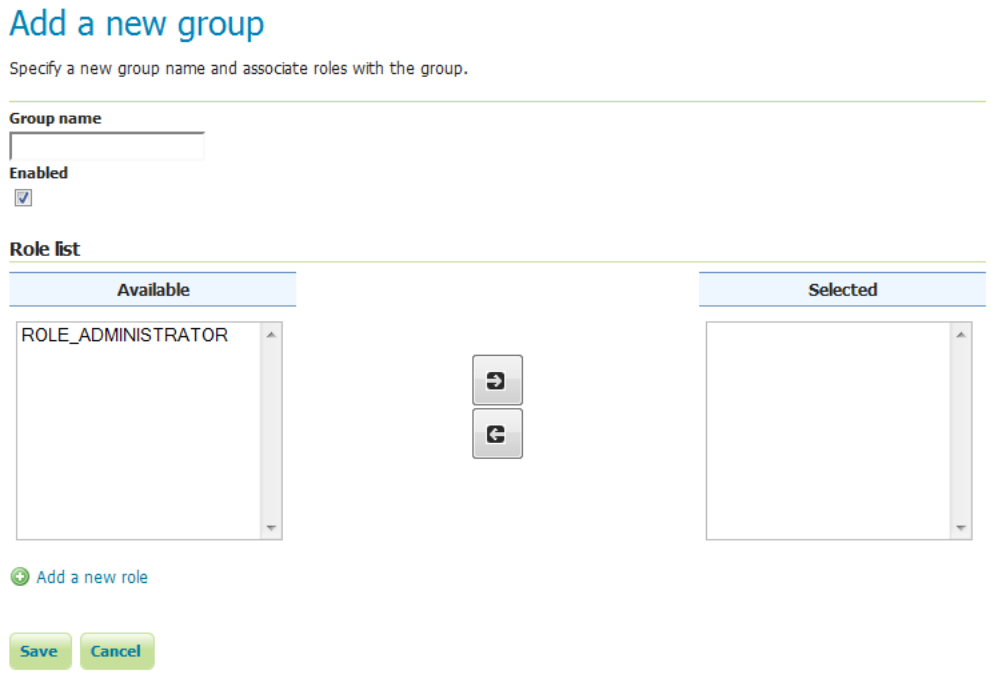


Figure 5.103: *Creating or editing a group*

Option	Description
Group name	The name of the group
Enabled	When selected the group will be active
Role list	Full list of roles, including a list of roles to which the group is associated. Association can be toggled here via the arrow buttons.
Add a new role	Shortcut to adding a new role

In this menu, user/group services can be added, removed, or edited. By default, there is one user/group service in GeoServer, which is *XML-based*. It is encrypted with *Weak PBE* and uses the default *password policy*. It is also possible to have a user/group service based on *JDBC* with or without JNDI.

Role services

In this menu, role services can be added, removed, or edited. By default, the active role service in GeoServer is *XML-based*, but it is also possible to have a role service based on *JDBC*, with or without JNDI.

The Administrator role is called `ROLE_ADMINISTRATOR`.

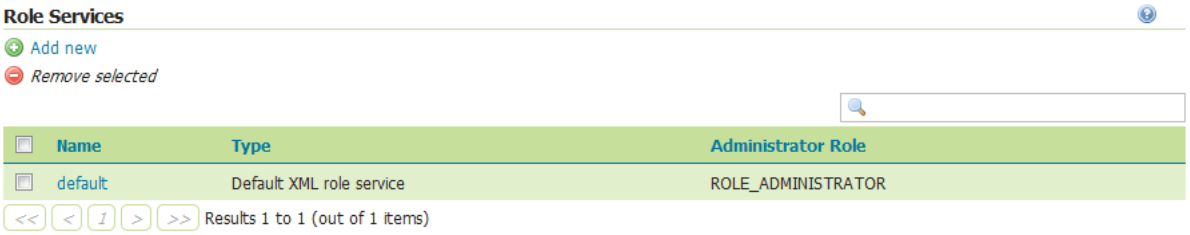


Figure 5.104: *Role services*

Clicking an existing role service will open it for editing, while clicking the *Add new* link will configure a new role service.

There are two pages for configuration: Settings and Roles.

Note: When creating a new role service, the form filled out initially can be found under the Settings tab.

Add new XML role service

To add a new XML role service, click the *Add new* link. XML is the default option. The following figure shows the configuration options for an XML role service.

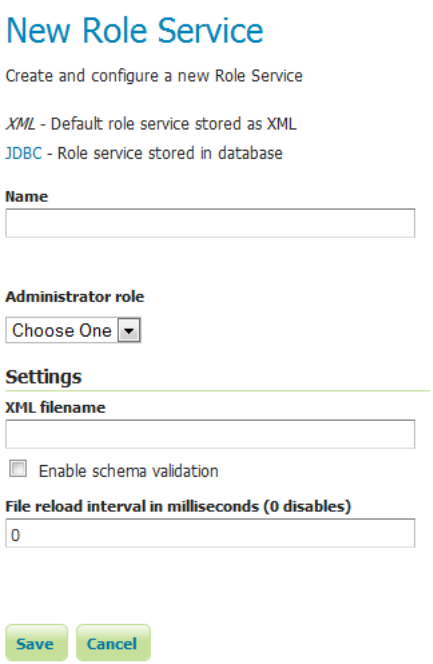


Figure 5.105: *Adding an XML role service*

Option	Description
Name	The name of the role service
Administrator role	The name of the role that performs the administrator functions
XML filename	Name of the file that will contain the role information. Default is `roles.xml` in the `security/role/<name_of_roleservice>` directory.
File reload interval	Defines the frequency (in milliseconds) in which GeoServer will check for changes to the XML file. If the file is found to have been modified, GeoServer will recreate the user/group database based on the current state of the file. This value is meant to be set in cases where the XML file contents might change "out of process" and not directly through the web admin interface. The value is specified in milliseconds. A value of 0 disables any checking of the file.

Add new JDBC role service

To add a new XML role service, click the *Add new* link, and then the *JDBC* option at the top of the following form. The following figure shows the configuration options for a JDBC role service.

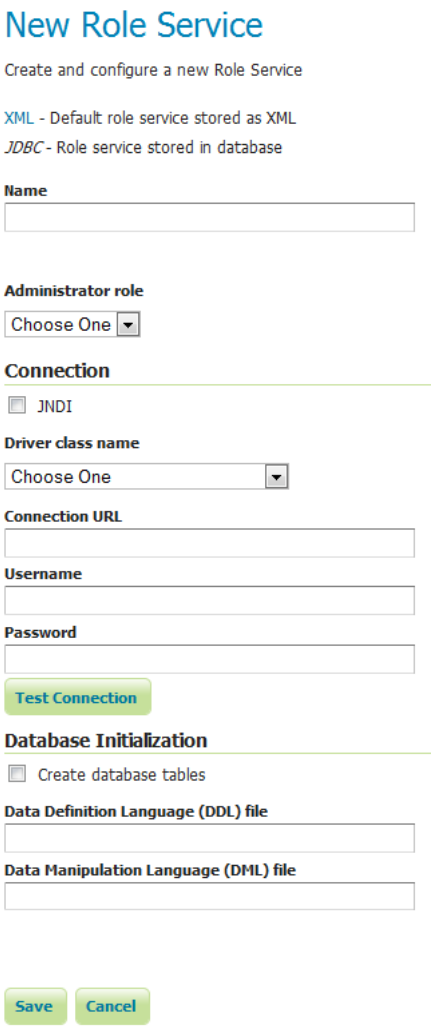


Figure 5.106: *Adding a role service via JDBC*

Option	Description
Name	Name of the JDBC role service in GeoServer
Administrator role	The name of the role that performs the administrator function
JNDI	When unchecked, specifies a direct connection to the database. When checked, specifies an existing connection located through *JNDI*.
Driver class name	JDBC driver to use for the database connection
Connection URL	Specifies the JDBC URL to use when creating the database connection
Username	Username to use when connecting to the database
Password	Password to use when connecting to the database
Create database tables	Specifies whether to create all the necessary tables in the underlying database
Data Definition Language (DDL) file	Specifies a custom DDL file to use for creating tables in the underlying database, for cases where the default DDL statements fail on the given database. If left blank, internal defaults are used.
Data Manipulation Language (DML) file	Specifies a custom DML file to use for accessing tables in the underlying database, for cases where the default DML statements fail on the given database. If left blank, internal defaults are used.

In addition to the parameters listed above, the following additional parameter will apply when the *JNDI* flag is set.

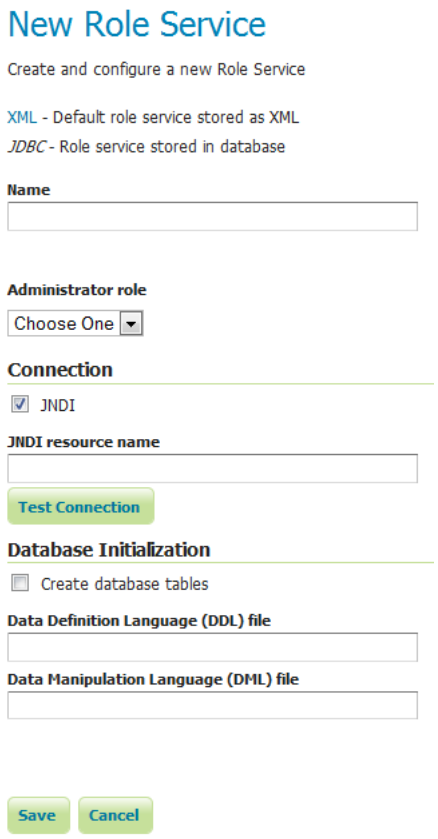


Figure 5.107: *Adding a role service via JDBC with JNDI*

Option	Description
JNDI resource name	JNDI name used to locate the database connection.

Add new LDAP role service

To add a new LDAP role service, click the *Add new* link, and then the *LDAP* option at the top of the following form. The following figure shows the configuration options for a LDAP role service.

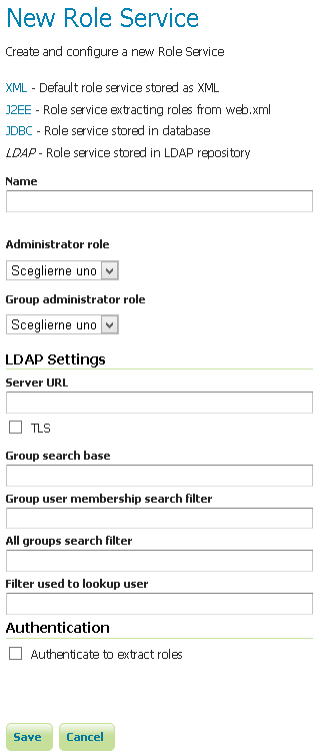


Figure 5.108: *Adding a role service via LDAP*

Option	Description
Name	Name of the LDAP role service in GeoServer
Administrator role	The name of the role that performs the administrator function
Group administrator role	The name of the role that performs the group administrator function
Server URL	URL for the LDAP server connection. It must include the protocol, host, and port, as well as the "distinguished name" (DN) for the root of the LDAP tree.
TLS	Enables a STARTTLS connection. (See the section on *Secure LDAP connections*.)
Group search base	Relative name of the node in the tree to use as the base for LDAP groups. Example: `ou=groups`. The root DN specified as port of the *Server URL* is automatically appended.
Group user membership search filter	Search pattern for extracting users of a LDAP group a user belongs to. This may contain some placeholder values: `{0}`, the `username` of the user, for example `bob`. `{1}`, the full DN of the user, for example `uid=bob,ou=users`. To use this placeholder, the *Filter used to lookup user* needs to be defined, so that the dn of a user can be extracted from its username.
All groups search filter	Search pattern for locating the LDAP groups to be mapped to GeoServer roles inside the *Group search base* root node
Filter used to lookup user.	optional filter used to extract a user dn, to be used together with *Group user membership search filter* when the {1} placeholder is specified. This may contain a placeholder value: `{0}`, the `username` of the user, for example `bob`.
Authenticate to extract roles	When checked all LDAP searches will be done in authenticated mode, using the credentials given with the *Username* and *Password* options
Username	Username to use when connecting to the LDAP server. Only applicable when the *Authenticate to extract roles* parameter is **checked**.
Password	Password to use when connecting to the LDAP server. Only applicable when the *Authenticate to extract roles* parameter is **checked**.

Edit role service

Once the new role service is added (either XML or JDBC), clicking it in the list of role services will allow the additional options to be specified, such as the roles associated with the service.

There are two tabs in the resulting menu: *Settings* and *Roles*. The Settings tab is identical to that found when creating the role service, while the Roles tab is described below.

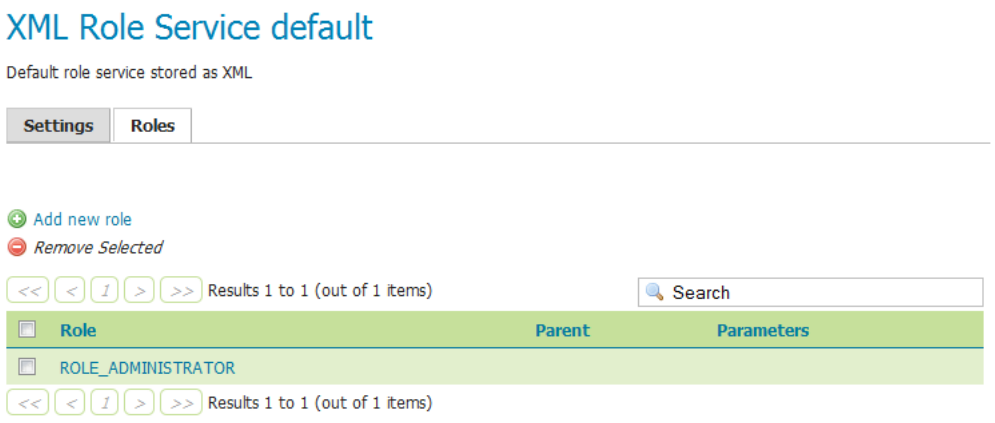


Figure 5.109: *Roles tab*

Clicking a role will allow its parameters to be changed, while clicking the *Add new* link will create a new

role.

Add role

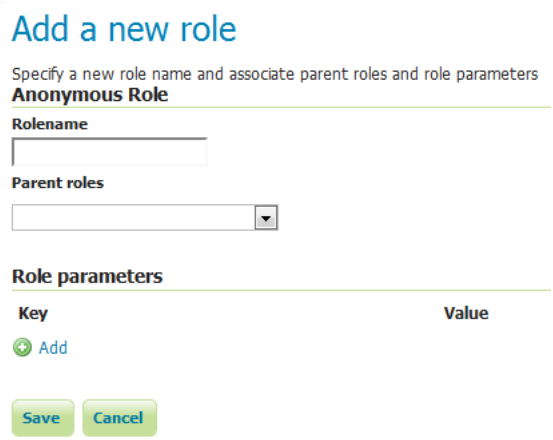


Figure 5.110: *Creating or editing a role*

Option	Description
Role name	The name of role. Convention is uppercase, but is not required.
Parent roles	The role that this role inherits. See the section on *Roles* for more information on inheritance.
Role parameters	Key/value pairs associated with the role. Used for associating additional information with the role.

5.7.5 Data

This section provides access to security settings related to data management and *Layer security*. Data access is granted to roles, and roles are granted to users and groups.

Rules

There are two rules available by default, but they don't provide any restrictions on access by default. The first rule `*.*.r`, applied to all roles, states that any operation in any resource in any workspace can be read. The second rule, `*.*.w`, also applied to all roles, says the same for write access.

Clicking an existing rule will open it for editing, while clicking the *Add a new rule* link will create a new rule.

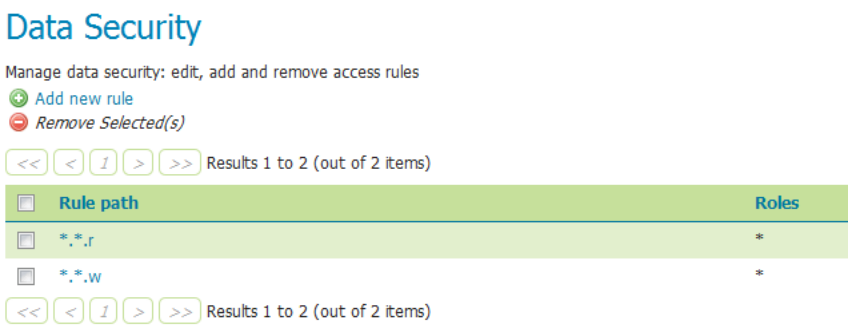


Figure 5.111: *Rules for data access*

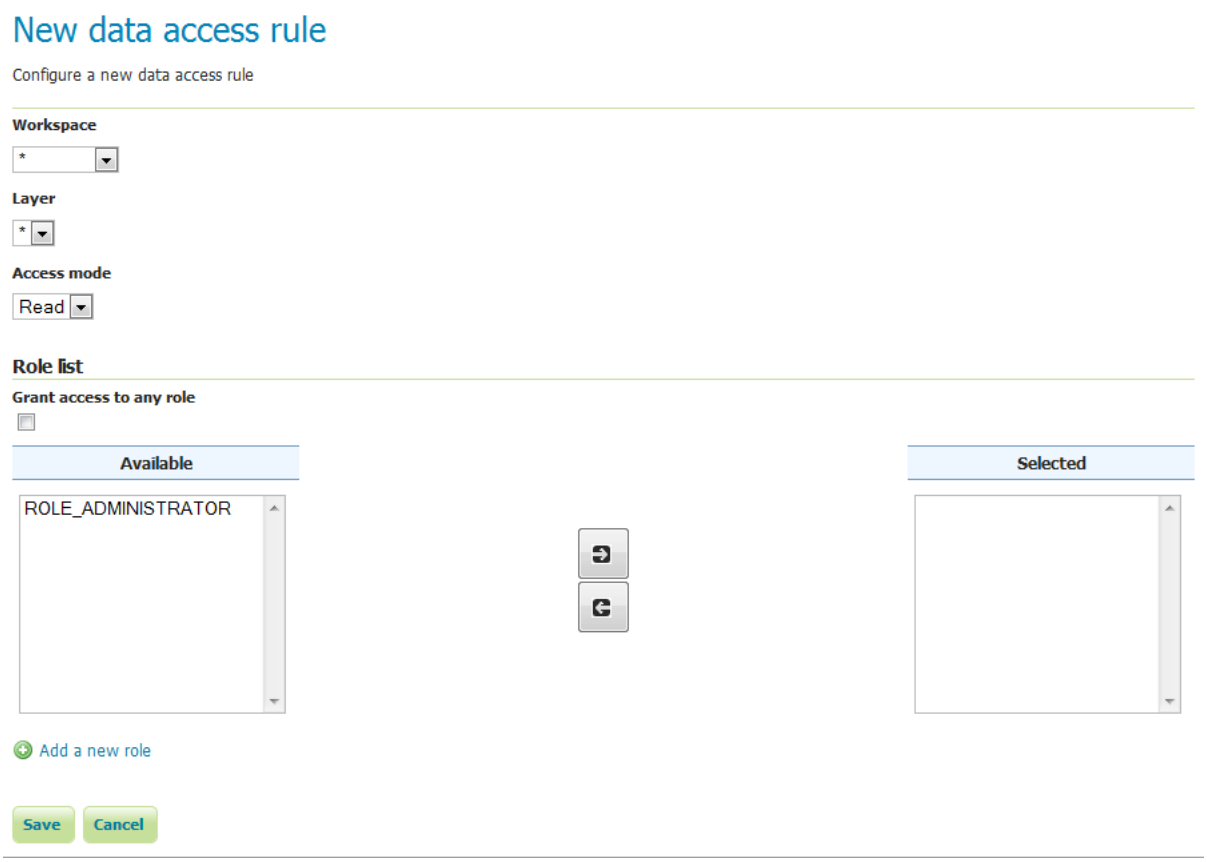


Figure 5.112: *Creating a new rule*

Option	Description
Workspace	Sets the allowed workspace for this rule. Options are * (all workspaces), or the name of each workspace.
Layer	Sets the allowed layer for this rule. Options are * (all layers), or the name of each layer in the above workspace. Will be disabled until the workspace is set.
Access mode	Specifies whether the rule refers to either `Read` or `Write` mode
Grant access to any role	If selected, the rule will apply to all roles, with no need to specify
Role list	Full list of roles, including a list of roles to which the rule is associated. Association can be toggled here via the arrow buttons. This option is not applied if *Grant access to any role* is checked.
Add a new role	Shortcut to adding a new role

Catalog Mode

This mode configures how GeoServer will advertise secured layers and behave when a secured layer is accessed without the necessary privileges. There are three options: *HIDE*, *MIXED*, and *CHALLENGE*. For further information on these options, please see the section on *Layer security*.

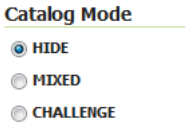


Figure 5.113: *Catalog mode*

5.7.6 Services

This section provides access to the settings for *Service Security*. GeoServer can limit access based on OWS services (WFS, WMS, etc.) and their specific operations (GetCapabilities, GetMap, and so on).

By default, no service-based security is in effect in GeoServer. However rules can be added, removed, or edited here.

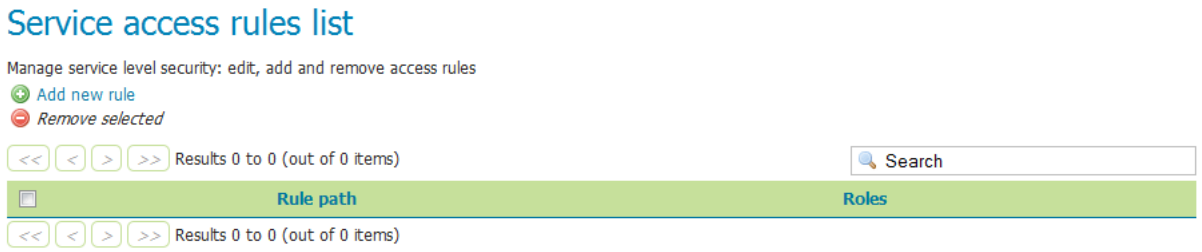


Figure 5.114: *Service access rules list*

Clicking the *Add a new rule* link will create a new rule.

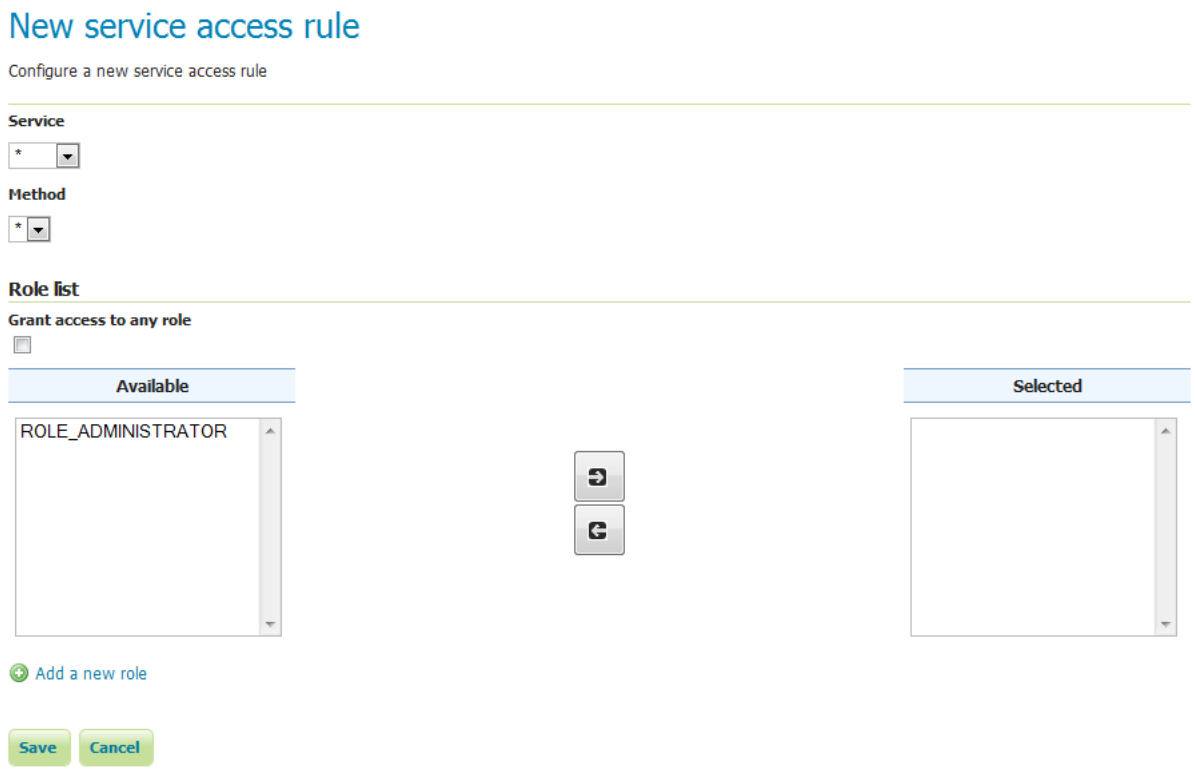


Figure 5.115: *New service rule*

Option	Description
Service	Sets the OWS service for this rule. Options are `*`, meaning all services, `wcs`, `wfs`, or `wms`.
Method	Sets the specific operation for this rule. Options depend on the *Service*, but include `*`, meaning all operations, as well as every service operation known to GeoServer, such as Capabilities, Transaction, GetMap, and more.
Grant access to any role	If selected, the rule will apply to all roles (no need to specify which ones)
Role list	Full list of roles, including a list of roles to which the rule is associated. Association can be switched here via the arrow buttons. This option is not applied if *Grant access to any role* is checked.
Add a new role	Shortcut to adding a new role

5.7.7 File Browsing

The GeoServer web admin employs a file browser dialog that will expose locations of the file system other than the GeoServer directory. These locations include the root of the file system and the users home directory. In highly secure and multi-tenant environments disabling this feature may be desired.

The property `GEOSERVER_FILEBROWSER_HIDEFS` can be used to disable this functionality. When set to `true` only the GeoServer data directory will be exposed through the file browser.

The property is set through one of the standard means:

- `web.xml`

```
<context-param>
  <param-name>GEOSERVER_FILEBROWSER_HIDEFS</param-name>
  <param-value>true</param-value>
</context-param>
```

- System property

  ```
  -DGEOSERVER_FILEBROWSER_HIDEFS=true
  ```

- Environment variable

  ```
  export GEOSERVER_FILEBROWSER_HIDEFS=true
  ```

5.8 Demos

This page contains helpful links to various information pages regarding GeoServer and its features. You do not need to be logged into GeoServer to access this page.

Figure 5.116: *Demos page*

5.8.1 Demo Requests

This page has example WMS, WCS and WFS requests for GeoServer that you can use, examine, and change. Select a request from the drop down list.

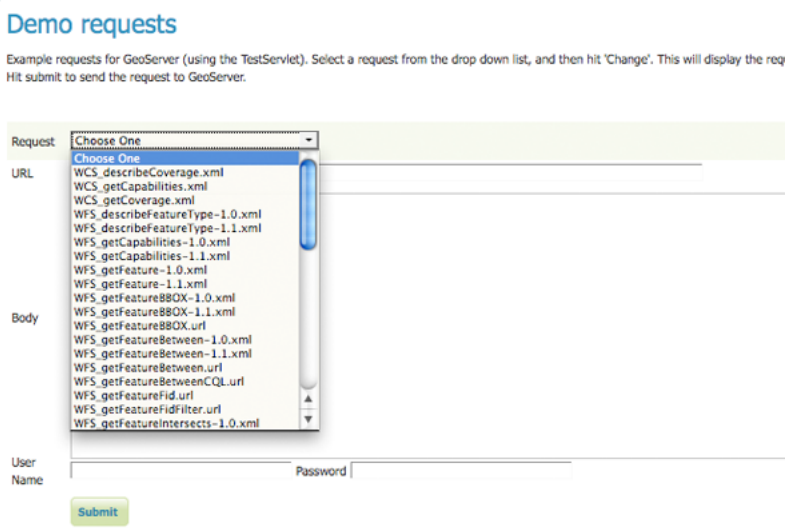


Figure 5.117: *Selecting demo requests*

Both Web Feature Service (*Web Feature Service*) as well as Web Coverage Service (*Web Coverage Service*) requests will display the request URL and the XML body. Web Map Service (*Web Map Service*) requests will only display the request URL.

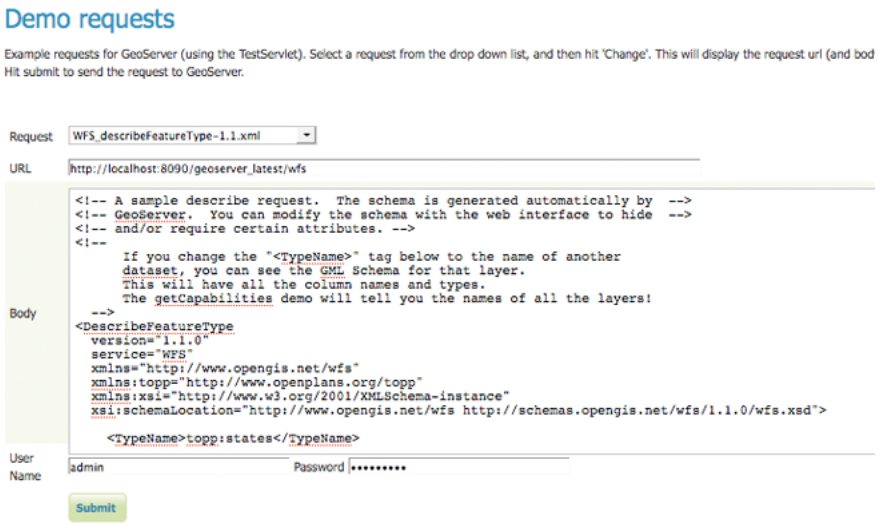


Figure 5.118: *WFS 1.1 DescribeFeatureType sample request*

Click *Submit* to send the request to GeoServer. For WFS and WCS requests, GeoServer will automatically generate an XML reponse.

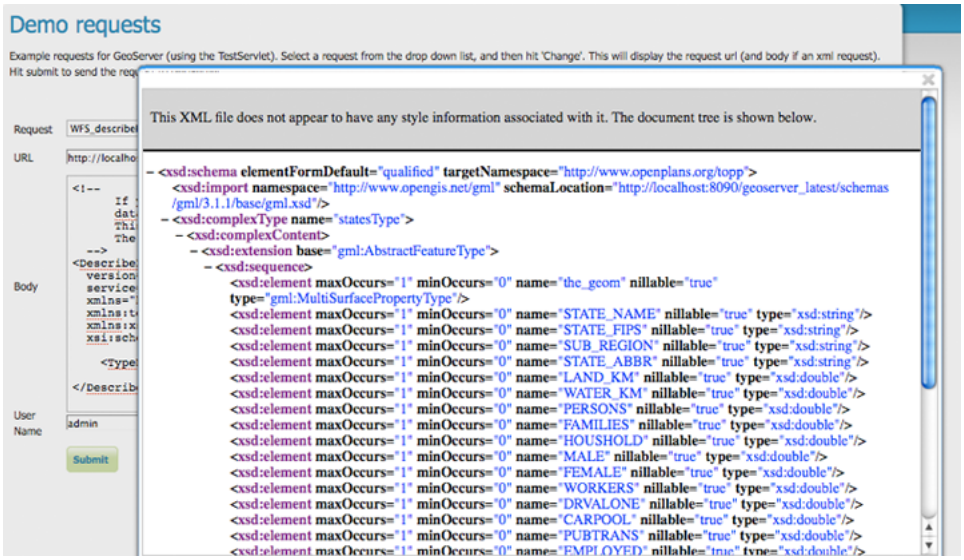


Figure 5.119: *XML reponse from a WFS 1.1 DescribeFeatureType sample request*

Submitting a WMS GetMap request displays an image based on the provided geographic data.

WMS GetFeatureInfo requests retrieve information regarding a particular feature on the map image.

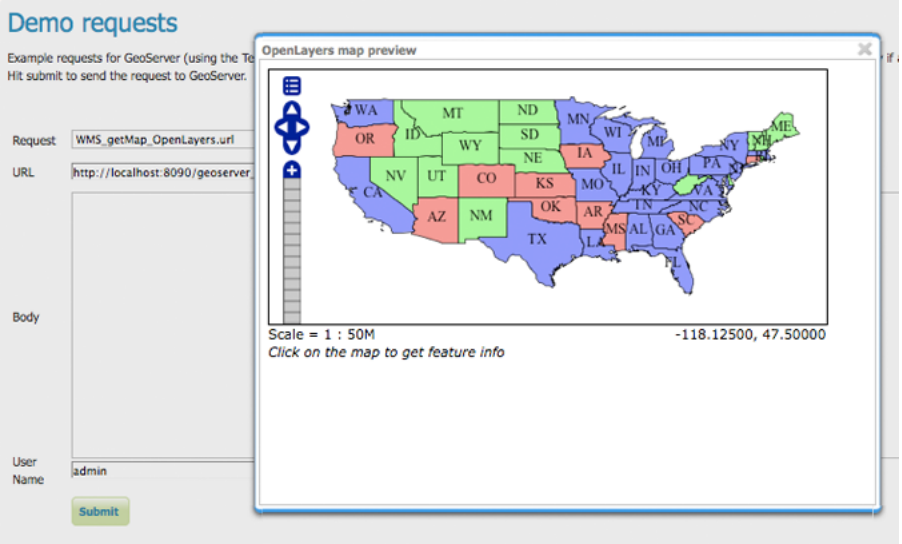


Figure 5.120: *OpenLayers WMS GetMap request*

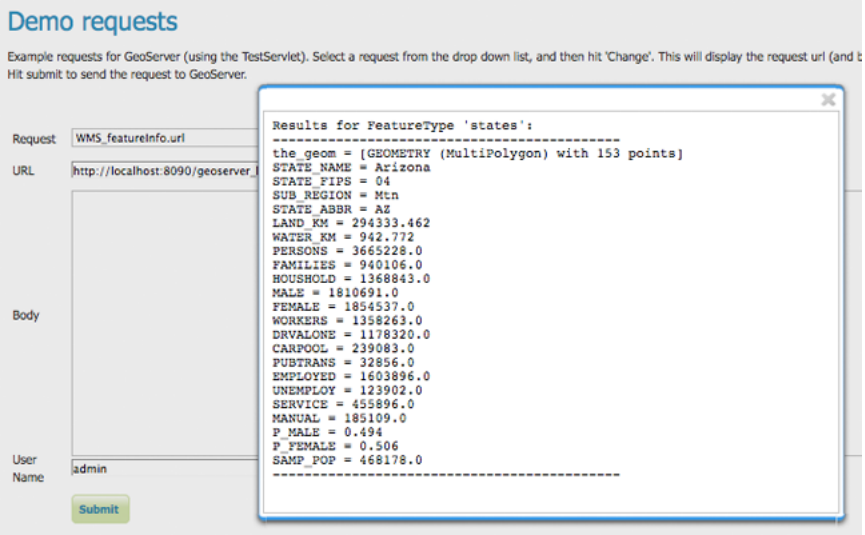


Figure 5.121: *WMS GetFeatureInfo request*

5.8.2 SRS

GeoServer natively supports almost 4,000 Spatial Referencing Systems (SRS), also known as **projections**, and more can be added. A spatial reference system defines an ellipsoid, a datum using that ellipsoid, and either a geocentric, geographic or projection coordinate system. This page lists all SRS info known to GeoServer.

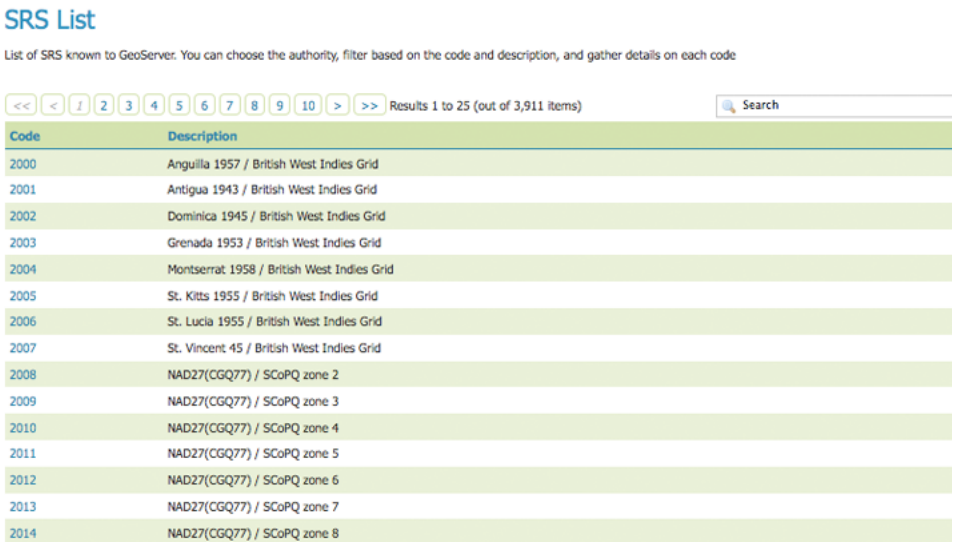


Figure 5.122: *Listing of all Spatial Referencing Systems (SRS) known to GeoServer*

The *Code* column refers to the unique integer identifier defined by the author of that spatial reference system. Each code is linked to a more detailed description page, accessed by clicking on that code.

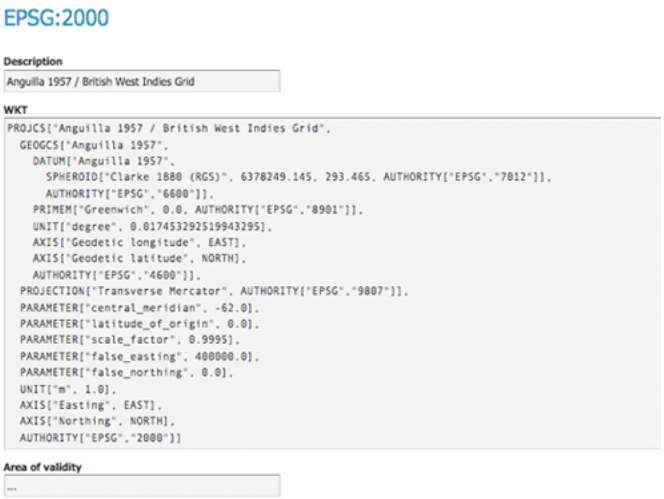


Figure 5.123: *Details for SRS EPSG:2000*

The title of each SRS is composed of the author name and the unique integer identifier (code) defined by the Author. In the above example, the author is the European Petroleum Survey Group (EPSG) and the Code is 2000. The fields are as follows:

Description—A short text description of the SRS

WKT—A string describing the SRS. WKT stands for "Well Known Text"

Area of Validity—The bounding box for the SRS

CHAPTER 6

Working with Vector Data

This section discusses the vector data sources that GeoServer can access.

The standard GeoServer installation supports the loading and serving of the following data formats:

6.1 Shapefile

A shapefile is a popular geospatial vector data format.

Note: While GeoServer has robust support for the shapefile format, it is not the recommended format of choice in a production environment. Databases such as PostGIS are more suitable in production and offer better performance and scalability. See the section on *Running in a Production Environment* for more information.

6.1.1 Adding a shapefile

A shapefile is actually a collection of files (with the extensions: `.shp`, `.dbf`, `.shx`, `.prj`, and sometimes others). All of these files need to be present in the same directory in order for GeoServer to accurately read them. As with all formats, adding a shapefile to GeoServer involves adding a new store to the existing *Stores* through the *Web Administration Interface*.

Warning: The `.prj` file, while not mandatory, is strongly recommended when working with GeoServer as it contains valuable projection info. GeoServer may not be able to load your shapefile without it!

To begin, navigate to *Stores* → *Add a new store* → *Shapefile*.

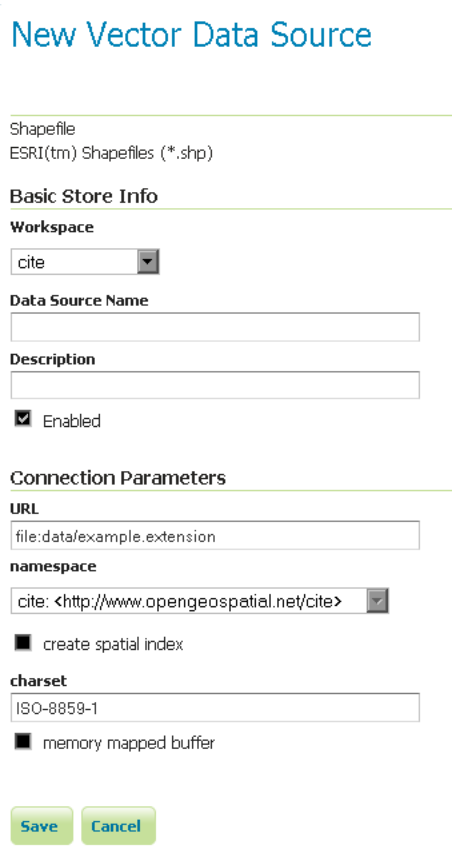


Figure 6.1: *Adding a shapefile as a store*

Option	Description
Workspace	Name of the workspace to contain the store. This will also be the prefix of the layer created from the store.
Data Source Name	Name of the shapefile as known to GeoServer. Can be different from the filename. The combination of the workspace name and this name will be the full layer name (ex: topp:states).
Description	Description of the shapefile/store.
Enabled	Enables the store. If unchecked, no data in the shapefile will be served.
URL	Location of the shapefile. Can be an absolute path (such as `file:C:\Data\shapefile.shp`) or a path relative to the data directory (such as `file:data/shapefile.shp`.
namespace	Namespace to be associated with the shapefile. This field is altered by changing the workspace name.
create spatial index	Enables the automatic creation of a spatial index.
charset	Character set used to decode strings from the `.dbf` file.
memory mapped buffer	Enables the use of memory mapped I/O, improving caching of the file in memory. **Turn off on Windows servers**.
Cache and reuse memory maps	

When finished, click *Save*.

6.1.2 Configuring a shapefile layer

Shapefiles contain exactly one layer, which needs to be added as a new layer before it will be able to be served by GeoServer. See the section on *Layers* for how to add and edit a new layer.

6.2 Directory of spatial files

The directory store automates the process of loading multiple shapefiles into GeoServer. Loading a directory that contains multiple shapefiles will automatically add each shapefile to GeoServer.

Note: While GeoServer has robust support for the shapefile format, it is not the recommended format of choice in a production environment. Databases such as PostGIS are more suitable in production and offer better performance and scalability. See the section on *Running in a Production Environment* for more information.

6.2.1 Adding a directory

To begin, navigate to *Stores* → *Add a new store* → *Directory of spatial files*.

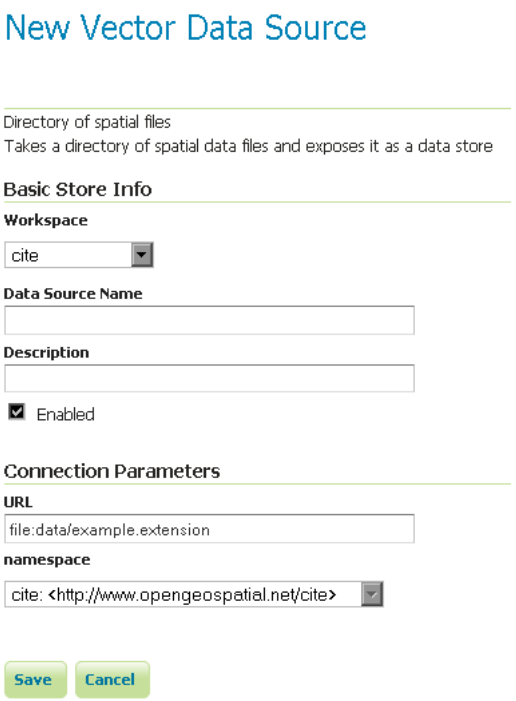


Figure 6.2: *Adding a directory of spatial files as a store*

Option	Description
Workspace	Name of the workspace to contain the store. This will also be the prefix of all of the layer names created from shapefiles in the store.
Data Source Name	Name of the store as known to GeoServer.
Description	Description of the directory store.
Enabled	Enables the store. If disabled, no data in any of the shapefiles will be served.
URL	Location of the directory. Can be an absolute path (such as `file:C:\Data\shapefile_directory`) or a path relative to the data directory (such as `file:data/shapefile_directory`.
namespace	Namespace to be associated with the store. This field is altered by changing the workspace name.

When finished, click *Save*.

6.2.2 Configuring shapefiles

All of the shapefiles contained in the directory store will be loaded as part of the directory store, but they will need to be individually configured as new layers they can be served by GeoServer. See the section on *Layers* for how to add and edit new layers.

6.3 Java Properties

The Properties data store provides access to one or more feature types (layers) stored in Java property files; these are plain text files stored on the local filesystem. The Properties data store was never intended to be shipped with GeoServer. It originated in a GeoTools tutorial, and later found widespread use by developers in automated tests that required a convenient store for small snippets of data. It slipped into GeoServer through the completeness of the packaging process, and was automatically detected and offered to users via the web interface. The Property data store has proved useful in tutorials and examples.

- We do not recommend the use the Properties data store for large amounts of data, with either many features or large geometries. Its performance will be terrible.
- For small data sets, such as collections of a few dozen points, you may find it to be satisfactory. For example, if you have a few points you wish to add as an extra layer, and no convenient database in which store them, the Properties data store provides a straightforward means of delivering them.
- Changes to a property file are immediately reflected in GeoServer responses. There is no need to recreate the data store unless the first line of a property file is changed, or property files are added or removed.

6.3.1 Adding a Properties data store

By default, *Properties* will be an option in the *Vector Data Sources* list when creating a new data store.

Figure 6.3: *Properties in the list of vector data stores*

6.3.2 Configuring a Properties data store

Option	Description
`Workspace`	Sets the namespace prefix of the feature types (layers) and their properties
`Data Source Name`	Unique identifier to distinguish this data store
`Description`	Optional text giving a verbose description of the data store
`Enabled`	Features will be delivered only if this option is checked
`directory`	Filesystem path to a directory containing one or more property files, for example `/usr/local/geoserver/data/ex`

Every property file `TYPENAME.properties` in the designated directory is served as a feature type `TYPENAME` (the name of the file without the `.properties`), in the namespace of the data store.

Before a feature type (layer) can be used, you must edit it to ensure that its bounding box and other metadata is configured.

6.3.3 Property file format

The property file format is a subset of the Java properties format: a list of lines of the form `KEY=VALUE`.

This example `stations.properties` defines four features of the feature type (layer) `stations`:

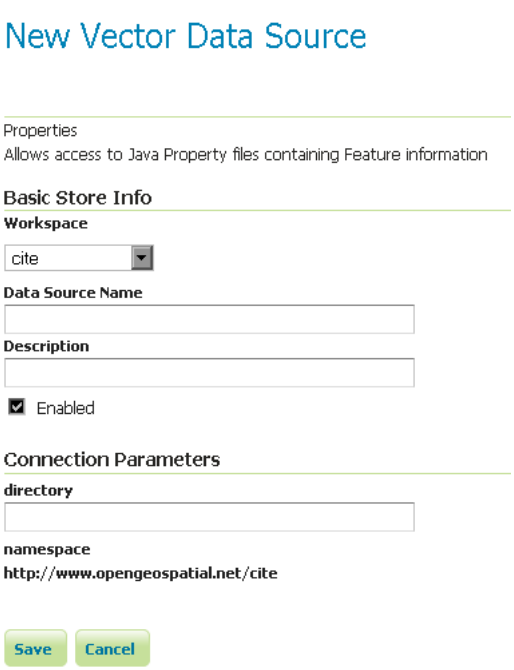


Figure 6.4: *Configuring a Properties data store*

```
_=id:Integer,code:String,name:String,location:Geometry:srid=4326
stations.27=27|ALIC|Alice Springs|POINT(133.8855 -23.6701)
stations.4=4|NORF|Norfolk Island|POINT(167.9388 -29.0434)
stations.12=12|COCO|Cocos|POINT(96.8339 -12.1883)
stations.31=31|ALBY|Albany|POINT(117.8102 -34.9502)
```

- Blank lines are not permitted anywhere in the file.
- The first line of the property file begins with _= and defines the type information required to interpret the following lines.
  - Comma separated values are of the form `NAME:TYPE`
  - Names are the property name that are used to encode the property in WFS responses.
  - Types include `Integer`, `String`, `Float`, and `Geometry`
  - `Geometry` can have an extra suffix `:srid=XXXX` that defines the Spatial Reference System by its numeric EPSG code. Note that geometries defined in this way are in longitude/latitude order.
- Subsequent lines define features, one per line.
  - The key before the = is the feature ID (`fid` or `gml:id` in WFS responses). Each must be an NCName.
  - Feature data follows the = separated by vertical bars (`|`). The types of the data must match the declaration on the first line.
  - Leave a field empty if you want it to be null; in this case the property will be ignored.

Note that in this example `srid=4326` sets the spatial reference system (SRS) to `EPSG:4326`, which is by convention in longitude/latitude order when referred to in the short form. If you request these features in GML 3 you will see that GeoServer correctly translates the geometry to the URN form SRS

`urn:x-ogc:def:crs:EPSG:4326` in latitude/longitude form. See the *WFS* page for more on SRS axis order options.

Other data sources are supplied as GeoServer extensions. Extensions are downloadable modules that add functionality to GeoServer. Extensions are available at the GeoServer download page.

Warning: The extension version must match the version of the GeoServer instance.

6.4 GML

Note: GeoServer does not come built-in with support for GML; it must be installed through an extension. Proceed to *Installing the GML extension* for installation details.

Warning: Currently the GML extension is unmaintained and carries unsupported status. While still usable, do not expect the same reliability as with other extension.

Geographic Markup Language (GML) is a XML based format for representing vector based spatial data.

6.4.1 Supported versions

Currently GML version 2 is supported.

6.4.2 Installing the GML extension

1. Download the GML extension from the GeoServer download page.

   Warning: Make sure to match the version of the extension to the version of the GeoServer instance!

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

6.4.3 Adding a GML data store

Once the extension is properly installed *GML* will be an option in the *Vector Data Sources* list when creating a new data store.

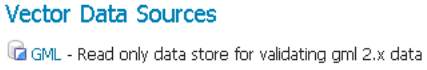


Figure 6.5: *GML in the list of vector data stores*

6.4.4 Configuring a GML data store

6.5 VPF

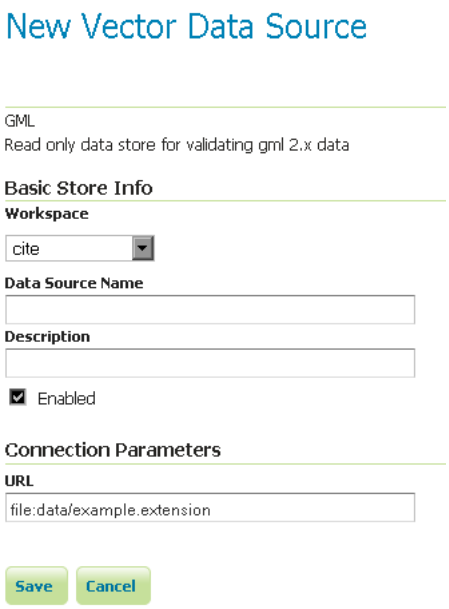


Figure 6.6: *Configuring a GML data store*

Note: GeoServer does not come built-in with support for VPF; it must be installed through an extension. Proceed to *Installing the VPF extension* for installation details.

Vector Product Format (VPF) is a military standard for vector-based digital map products produced by the U.S. Department of Defense. For more information visit The National Geospatial-Intelligence Agency.

6.5.1 Installing the VPF extension

1. Download the VPF extension from the GeoServer download page.

   Warning: Make sure to match the version of the extension to the version of the GeoServer instance!

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

6.5.2 Adding a VPF file

Once the extension is properly installed *Vector Product Format Library* will be an option in the *Vector Data Sources* list when creating a new data store.

Figure 6.7: *VPF in the list of new data sources*

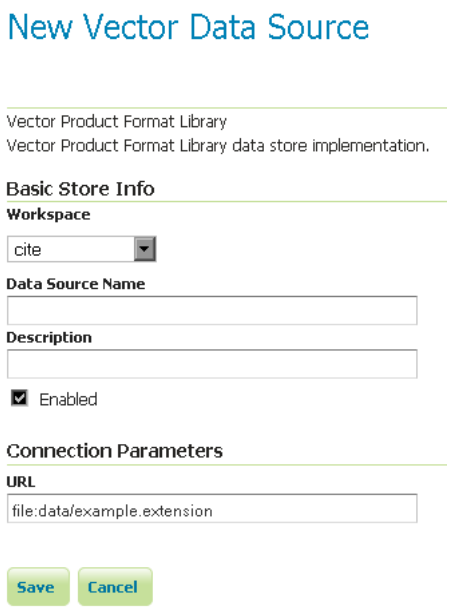


Figure 6.8: *Configuring a VPF data store*

6.5.3 Configuring a VPF data store

6.6 Pregeneralized Features

Note: GeoServer does not come built-in with support for Pregeneralized Features; it must be installed through an extension.

6.6.1 Installing the Pregeneralized Features extension

1. Download the Pregeneralized Features extension from the GeoServer download page.

   Warning: Make sure to match the version of the extension to the version of the GeoServer instance!

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

6.6.2 Adding a Pregeneralized Features data store

If the extension is properly installed, *Generalized Data Store* will be listed as an option when creating a new data store.

Figure 6.9: *Generalized Data Store in the list of vector data stores*

6.6.3 Configuring a Pregeneralized Features data store

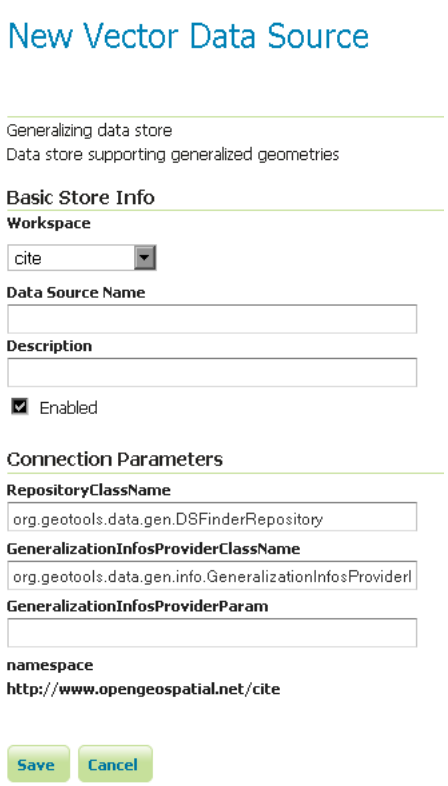


Figure 6.10: *Configuring a Pregeneralized Features data store*

For a detailed description, look at the *Tutorial*

CHAPTER 7

Working with Raster Data

This section discusses the raster (coverage) data sources that GeoServer can access.

The standard GeoServer installation supports the loading and serving of the following data formats:

7.1 GeoTIFF

A GeoTIFF is a georeferenced TIFF (Tagged Image File Format) file.

7.1.1 Adding a GeoTIFF data store

By default, *GeoTIFF* will be an option in the *Raster Data Sources* list when creating a new data store.

Figure 7.1: *GeoTIFF in the list of raster data stores*

7.1.2 Configuring a GeoTIFF data store

Option	Description
`Workspace`	Name of the workspace to contain the GeoTIFF store. This will also be the prefix of the raster layer created from the store.
`Data Source Name`	Name of the GeoTIFF as it will be known to GeoServer. This can be different from the filename. The combination of the workspace name and this name will be the full layer name (ex: world:landbase)
`Description`	A full free-form description of the GeoTIFF store.
`Enabled`	If checked, it enables the store. If unchecked (disabled), no data in the GeoTIFF will be served from GeoServer.
`URL`	Location of the GeoTIFF file. This can be an absolute path (such as `file:C:\Data\landbase.tif`) or a path relative to GeoServer's data directory (such as `file:data/landbase.tif`).

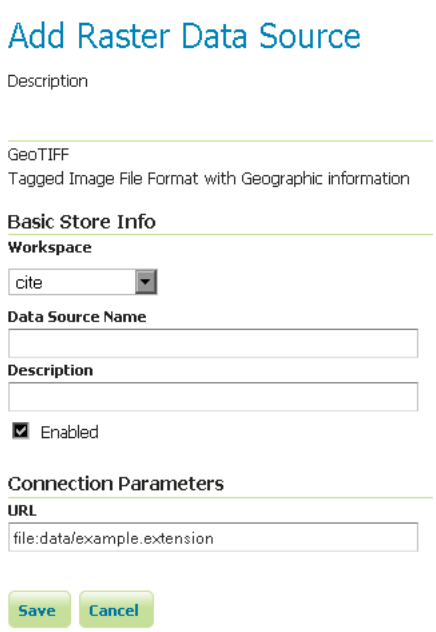


Figure 7.2: *Configuring a GeoTIFF data store*

7.2 GTOPO30

GTOPO30 is a Digital Elevation Model (DEM) dataset with a horizontal grid spacing of 30 arc seconds.

Note: An example of a GTOPO30 can be found at http://edc.usgs.gov/products/elevation/gtopo30/gtopo30.html

7.2.1 Adding a GTOPO30 data store

By default, *GTOPO30* will be an option in the *Raster Data Sources* list when creating a new data store.

Figure 7.3: *GTOPO30 in the list of raster data stores*

7.2.2 Configuring a GTOPO30 data store

Option	Description
Workspace	
Data Source Name	
Description	
Enabled	
URL	

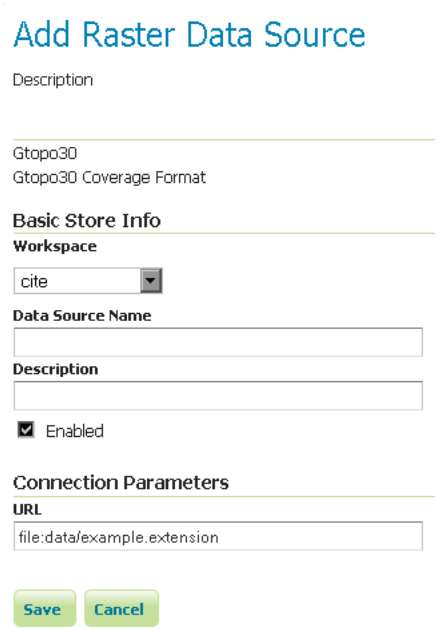


Figure 7.4: *Configuring a GTOPO30 data store*

7.3 WorldImage

A world file is a plain text file used to georeference raster map images. This file (often with an extension of `.jgw` or `.tfw`) accompanies an associated image file (`.jpg` or `.tif`). Together, the world file and the corresponding image file is known as a WorldImage in GeoServer.

7.3.1 Adding a WorldImage data store

By default, *WorldImage* will be an option in the *Raster Data Sources* list when creating a new data store.

Figure 7.5: *WorldImage in the list of raster data stores*

7.3.2 Configuring a WorldImage data store

Option	Description
`Workspace`	
`Data Source Name`	
`Description`	
`Enabled`	
`URL`	

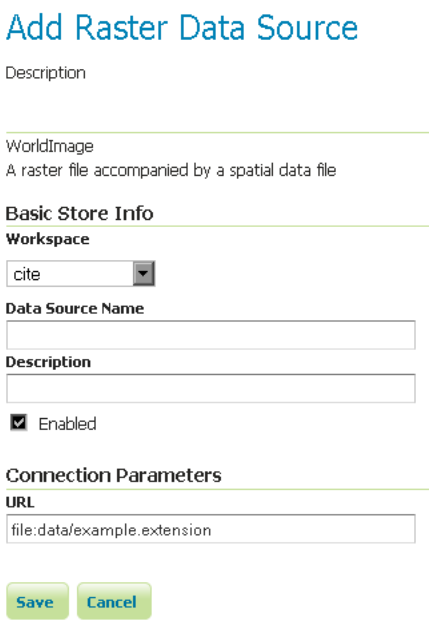


Figure 7.6: *Configuring a WorldImage data store*

7.4 ImageMosaic

The ImageMosaic data store allows the creation of a mosaic from a number of georeferenced rasters. The plugin can be used with GeoTIFFs, as well as rasters accompanied by a world file (`.pgw` for PNG files, `.jgw` for JPG files, etc.).

The "Mosaic" operation creates a mosaic of two or more source images. This operation could be used for example to assemble a set of overlapping geospatially rectified images into a contiguous image. It could also be used to create a montage of photographs such as a panorama.

The best current source of information on configuring an ImageMosiac is the tutorial: *Using the ImageMosaic plugin*.

7.4.1 Adding an ImageMosaic data store

By default, *ImageMosaic* will be an option in the *Raster Data Sources* list when creating a new data store.

Figure 7.7: *ImageMosaic in the list of raster data stores*

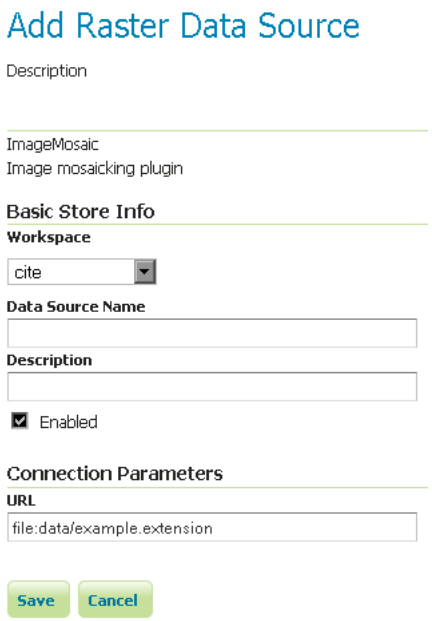


Figure 7.8: *Configuring an ImageMosaic data store*

7.4.2 Configuring an ImageMosaic data store

Option	Description
Workspace	
Data Source Name	
Description	
Enabled	
URL	

Other data sources are supplied as GeoServer extensions. Extensions are downloadable modules that add functionality to GeoServer. Extensions are available at the GeoServer download page.

Warning: The extension version must match the version of the GeoServer instance.

7.5 ArcGrid

ArcGrid is a coverage file format created by ESRI.

7.5.1 Adding an ArcGrid data store

By default, *ArcGrid* will be an option in the *Raster Data Sources* list when creating a new data store.

Figure 7.9: *ArcGrid in the list of raster data stores*

7.5.2 Configuring a ArcGrid data store

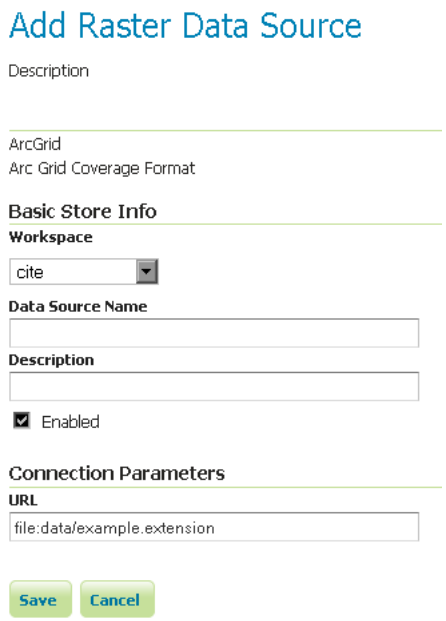


Figure 7.10: *Configuring an ArcGrid data store*

Option	Description
Workspace	
Data Source Name	
Description	
Enabled	
URL	

7.6 GDAL Image Formats

GeoServer can leverage the ImageI/O-Ext GDAL libraries to read selected coverage formats. GDAL is able to read many formats, but for the moment GeoServer supports only a few general interest formats and those that can be legally redistributed and operated in an open source server.

The following image formats can be read by GeoServer using GDAL:

- DTED, Military Elevation Data (.dt0, .dt1, .dt2): http://www.gdal.org/frmt_dted.html
- EHdr, ESRI .hdr Labelled: <http://www.gdal.org/frmt_various.html#EHdr>
- ENVI, ENVI .hdr Labelled Raster: <http://www.gdal.org/frmt_various.html#ENVI>
- HFA, Erdas Imagine (.img): <http://www.gdal.org/frmt_hfa.html>
- JP2MrSID, JPEG2000 (.jp2, .j2k): <http://www.gdal.org/frmt_jp2mrsid.html>
- MrSID, Multi-resolution Seamless Image Database: <http://www.gdal.org/frmt_mrsid.html>
- NITF: <http://www.gdal.org/frmt_nitf.html>
- ECW, ERDAS Compressed Wavelets (.ecw): <http://www.gdal.org/frmt_ecw.html>

- JP2ECW, JPEG2000 (.jp2, .j2k): http://www.gdal.org/frmt_jp2ecw.html
- AIG, Arc/Info Binary Grid: <http://www.gdal.org/frmt_various.html#AIG>
- JP2KAK, JPEG2000 (.jp2, .j2k): <http://www.gdal.org/frmt_jp2kak.html>

7.6.1 Installing GDAL extension

From GeoServer version 2.2.x, GDAL must be installed as an extension. To install it:

1. Navigate to the GeoServer download page
2. Find the page that matches the version of the running GeoServer.

   Warning: Be sure to match the version of the extension with that of GeoServer, otherwise errors will occur.

3. Download the GDAL extension. The download link for *GDAL* will be in the *Extensions* section under *Coverage Store*.
4. Extract the files in this archive to the `WEB-INF/lib` directory of your GeoServer installation.

Moreover, in order for GeoServer to leverage these libraries, the GDAL (binary) libraries must be installed through your host system's OS. Once they are installed, GeoServer will be able to recognize GDAL data types. See bloe for more information.

Installing GDAL native libraries

The ImageIO-Ext GDAL plugin for geoserver master uses ImageIO-Ext 1.1.7 whose artifacts can be downloaded from here.

Browse to the native and then gdal directory for the link. Now you should see a list of artifacts that can be downloaded. We need to download two things now:

1. The CRS definitions
2. The native libraries matching the target operating system

Let's now install the CRS definitions.

- Click on the "gdal_data.zip" to download the CRS definitions archive.
- Extract this archive on disk and place it in a proper directory on your system.
- Create a GDAL_DATA environment variable to the folder where you have extracted this file. Make also sure that this directory is reachable and readable by the application server process's user.

We now have to install the native libraries.

- Assuming you are on a 64 bits Ubuntu 11 Linux Operating System (as an instance), click on the linux folder and then on "gdal192-Ubuntu11-gcc4.5.2-x86_64.tar.gz" to download the native libraries archive (Before doing this, make sure to read and agree with the ECWEULA if you intend to use ECW).
- Extract the archive on disk and place it in a proper directory on your system.

  Warning: If you are on Windows, make sure that the GDAL DLL files are on your PATH. If you are on Linux, be sure to set the LD_LIBRARY_PATH environment variable to refer to the folder where the SOs are extracted.

Note: The native libraries contains the GDAL gdalinfo utility which can be used to test whether or not the libs are corrupted. This can be done by browsing to the directory where the libs have been extracted and performing a *gdalinfo* command with the *formats* options that shows all the formats supported. Moreover the package contains also a Java versions of the gdalinfo utility to check also the Java bindings correct functioning (you can see a .bat script for Windows and .sh for Linux).

Once these steps have been completed, restart GeoServer. If all the steps have been performed correctly, new data formats will be in the *Raster Data Sources* list when creating a new data store as shown here below.

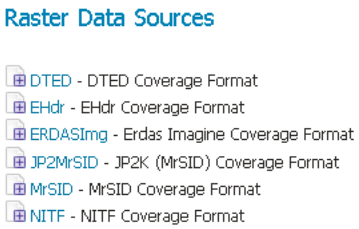


Figure 7.11: *GDAL image formats in the list of raster data stores*

If instead now new formats appear in the GUI and in the logs the following messages is shown:

it.geosolutions.imageio.gdalframework.GDALUtilities loadGDAL WARNING: Native library load failed.java.lang.UnsatisfiedLinkError: no gdaljni in java.library.path

that means that the installations failed for some reason.

7.6.2 Extra Steps for Windows Platforms

There are a few things to be careful with as well as some extra steps if you are deploying on Windows.

First of all, you'll notice that we have multiple versions like MSVC2005, MSVC2008 and so on macthing the Microsoft Visual C++ Redistributables. Depending on the version of the underlying operating system you'll have to pick up the right one. You can google around for the one you need.

That said, we have DLLs for both 32 bits as well as 64 bits Operating Systems. Again, pick the one that matches your infrastructure.

Note on running GeoServer as a Service on Windows

Simply deploying the GDAL ImageI/O-Ext native libraries in a location referred by the PATH environment variable (like, as an instance, the JDK/bin folder) doesn't allow GeoServer to leverage on GDAL, when run as a service. As a result, during the service startup, GeoServer log reports this worrysome message:

it.geosolutions.imageio.gdalframework.GDALUtilities loadGDAL WARNING: Native library load failed.java.lang.UnsatisfiedLinkError: no gdaljni in java.library.path

Taking a look at the wrapper.conf configuration file available inside the GeoServer installation (at bin/wrapper/wrapper.conf), there is this useful entry:

Java Library Path (location of Wrapper.DLL or libwrapper.so) wrapper.java.library.path.1=bin/wrapper/lib

To allow the GDAL native DLLs getting loaded, you have 2 possible ways:

1. Move the native DLLs on the referred path (bin/wrapper/lib)
2. Add a wrapper.java.library.path.2=path/where/you/deployed/nativelibs entry just after the wrapper.java.library.path1=bin/wrapper/lib line.

Adding support for ECW and MrSID on Windows

If you are on Windows and you want to add support for ECW and MrSID there is an extra step to perform.

In the Windows packaging ECW and MrSID are built as plugins hence they are not loaded by default but we need to place their DLLs in a location that is pointed by the *GDAL_DRIVER_PATH* environmental variable. GDAL uses internally this env variable to look up additional drivers (notice that there are a few default places where GDAL will look anyway). For additional information, please, check this link.

7.6.3 Configuring a DTED data store

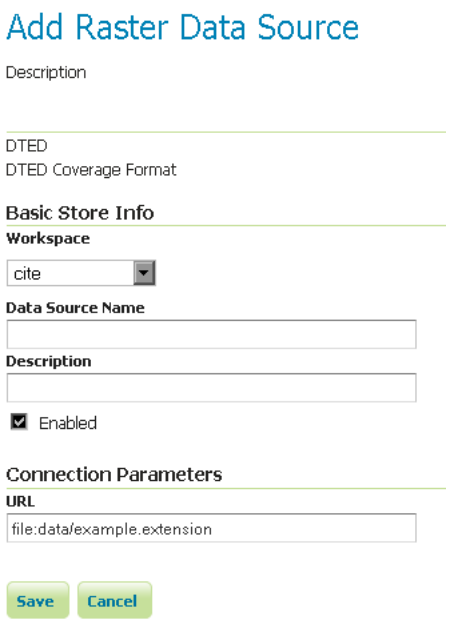


Figure 7.12: *Configuring a DTED data store*

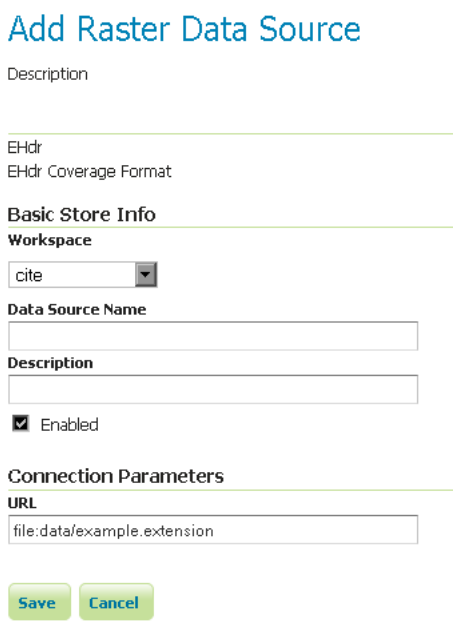


Figure 7.13: *Configuring a EHdr data store*

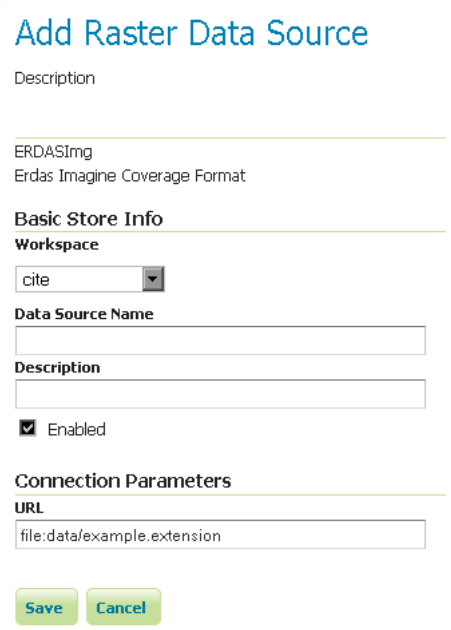


Figure 7.14: *Configuring a ERDASImg data store*

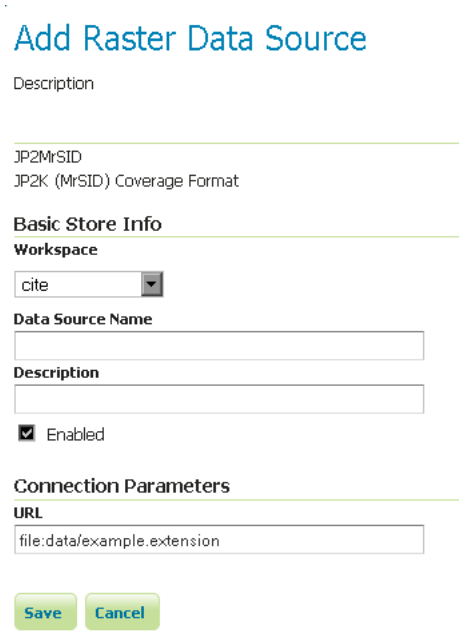


Figure 7.15: *Configuring a JP2MrSID data store*

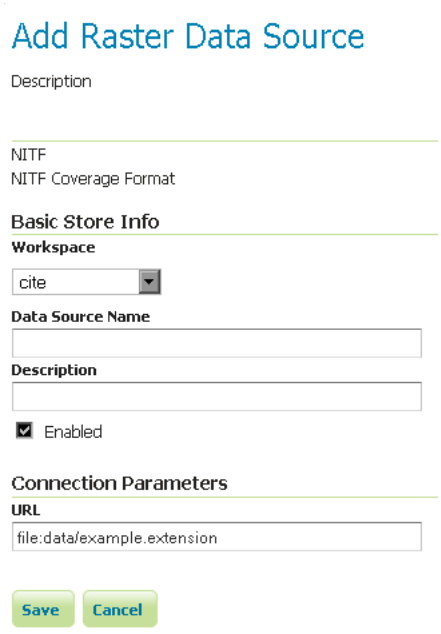


Figure 7.16: *Configuring a NITF data store*

7.6.4 Configuring a EHdr data store

7.6.5 Configuring a ERDASImg data store

7.6.6 Configuring a JP2MrSID data store

7.6.7 Configuring a NITF data store

7.7 Oracle Georaster

Note: GeoServer does not come built-in with support for Oracle Georaster; it must be installed through an extension. Proceed to *Image Mosaic JDBC* for installation details. This extension includes the support for Oracle Georaster.

7.7.1 Adding an Oracle Georaster data store

Read the geotools documentation for Oracle Georaster Support: http://docs.geotools.org/latest/userguide/library/covera
After creating the xml config file proceed to the section *Configuring GeoServer* in the *Image Mosaic JDBC Tutorial*

7.8 Postgis Raster

Note: GeoServer does not come built-in with support for Postgis raster columns, it must be installed through an extension. Proceed to *Image Mosaic JDBC* for installation details. This extension includes the support for Postgis raster.

7.8.1 Adding an Postgis raster data store

Read the geotools documentation for Postgis raster Support: http://docs.geotools.org/latest/userguide/library/coverage/
After creating the xml config file proceed to the section *Configuring GeoServer* in the *Image Mosaic JDBC Tutorial*

7.9 ImagePyramid

Note: GeoServer does not come built-in with support for Image Pyramid; it must be installed through an extension. Proceed to *Installing the ImagePyramid extension* for installation details.

An image pyramid is several layers of an image rendered at various image sizes, to be shown at different zoom levels.

7.9.1 Installing the ImagePyramid extension

1. Download the ImagePyramid extension from the GeoServer download page.

Warning: Make sure to match the version of the extension to the version of the GeoServer instance!

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

7.9.2 Adding an ImagePyramid data store

Once the extension is properly installed *ImagePyramid* will be an option in the *Raster Data Sources* list when creating a new data store.

Figure 7.17: *ImagePyramid in the list of raster data stores*

7.9.3 Configuring an ImagePyramid data store

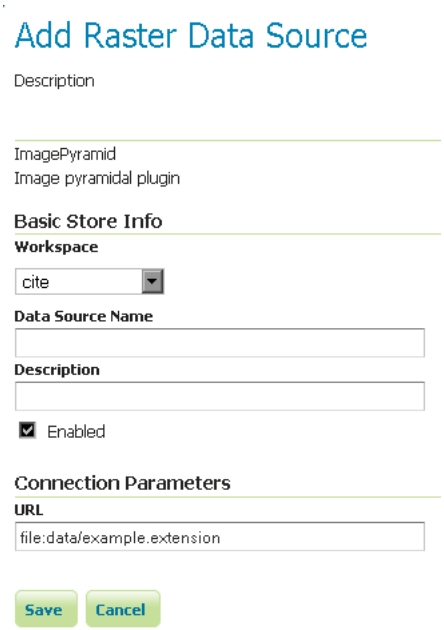


Figure 7.18: *Configuring an ImagePyramid data store*

Option	Description
`Workspace`	
`Data Source Name`	
`Description`	
`Enabled`	
`URL`	

7.10 Image Mosaic JDBC

Note: GeoServer does not come built-in with support for Image Mosaic JDBC; it must be installed through an extension. Proceed to *Installing the JDBC Image Mosaic extension* for installation details.

7.10.1 Installing the JDBC Image Mosaic extension

1. Download the JDBC Image Mosaic extension from the GeoServer download page.

   Warning: Make sure to match the version of the extension to the version of the GeoServer instance!

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

7.10.2 Adding an Image Mosaic JDBC data store

Once the extension is properly installed *Image Mosaic JDBC* will be an option in the *Raster Data Sources* list when creating a new data store.

Figure 7.19: *Image Mosaic JDBC in the list of vector data stores*

7.10.3 Configuring an Image Mosaic JDBC data store

For a detailed description, look at the *Tutorial*

7.11 Custom JDBC Access for image data

Note: GeoServer does not come built-in with support for Custom JDBC Access; it must be installed through an extension. Proceed to *Image Mosaic JDBC* for installation details. This extension includes the support for Custom JDBC Access.

7.11.1 Adding a coverage based on Custom JDBC Access

This extension is targeted to users having a special database layout for storing their image data or use a special data base extension concerning raster data.

Read the geotools documentation for Custom JDBC Access: http://docs.geotools.org/latest/userguide/library/coverage/j

After developing the custom plugin, package the classes into a jar file and copy it into the `WEB-INF/lib` directory of the geoserver installation.

Create the xml config file and proceed to the section *Configuring GeoServer* in the *Image Mosaic JDBC Tutorial*

Figure 7.20: *Configuring an Image Mosaic JDBC data store*

CHAPTER 8

Working with Databases

This section discusses the database data sources that GeoServer can access.

The standard GeoServer installation supports accessing the following databases:

8.1 PostGIS

PostGIS is an open source spatial database based on PostgreSQL, and is currently one of the most popular open source spatial databases today.

8.1.1 Adding a PostGIS database

As with all formats, adding a shapefile to GeoServer involves adding a new store to the existing *Stores* through the *Web Administration Interface*.

Using default connection

To begin, navigate to *Stores* → *Add a new store* → *PostGIS NG*.

Figure 8.1: *Adding a PostGIS database*

Option	Description
Workspace	Name of the workspace to contain the database. This will also be the prefix of any layer names created from tables in the database.
Data Source Name	Name of the database. This can be different from the name as known to PostgreSQL/PostGIS.
Description	Description of the database/store.
Enabled	Enables the store. If disabled, no data in the database will be served.
dbtype	Type of database. Leave this value as the default.
host	Host name where the database exists.
port	Port number to connect to the above host.
database	Name of the database as known on the host.
schema	Schema in the above database.
user	User name to connect to the database.
passwd	Password associated with the above user.
namespace	Namespace to be associated with the database. This field is altered by changing the workspace name.
max connections	Maximum amount of open connections to the database.
min connections	Minimum number of pooled connections.
fetch size	Number of records read with each interaction with the database.
Connection timeout	Time (in seconds) the connection pool will wait before timing out.
validate connections	Checks the connection is alive before using it.
Loose bbox	Performs only the primary filter on the bounding box. See the section on *Using loose bounding box* for details.
prepared-Statements	Enables prepared statements.

When finished, click *Save*.

Using JNDI

GeoServer can also connect to a PostGIS database using JNDI (Java Naming and Directory Interface).

To begin, navigate to *Stores* → *Add a new store* → *PostGIS NG (JNDI)*.

Option	Description
Workspace	Name of the workspace to contain the store. This will also be the prefix of all of the layer names created from the store.
Data Source Name	Name of the database. This can be different from the name as known to PostgreSQL/PostGIS.
Description	Description of the database/store.
Enabled	Enables the store. If disabled, no data in the database will be served.
dbtype	Type of database. Leave this value as the default.
jndiReference-Name	JNDI path to the database.
schema	Schema for the above database.
namespace	Namespace to be associated with the database. This field is altered by changing the workspace name.

When finished, click *Save*.

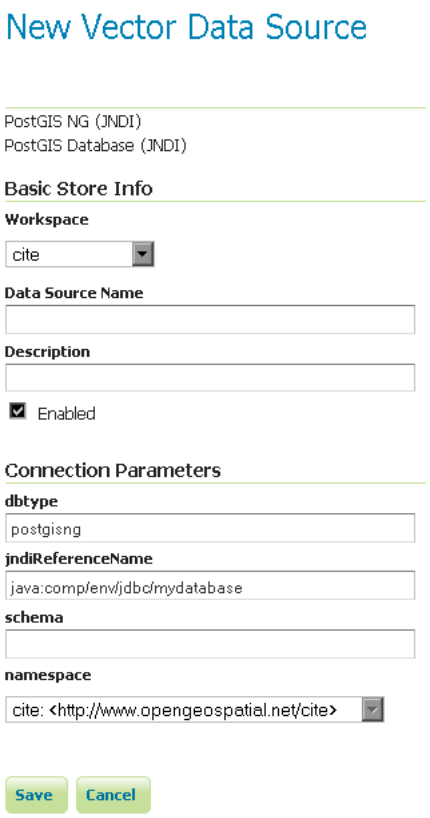


Figure 8.2: *Adding a PostGIS database (using JNDI)*

8.1.2 Configuring PostGIS layers

When properly loaded, all tables in the database will be visible to GeoServer, but they will need to be individually configured before being served by GeoServer. See the section on *Layers* for how to add and edit new layers.

8.1.3 Using loose bounding box

When the option *loose bbox* is enabled, only the bounding box of a geometry is used. This can result in a significant performance gain, but at the expense of total accuracy; some geometries may be considered inside of a bounding box when they are technically not.

If primarily connecting to this data via WMS, this flag can be set safely since a loss of some accuracy is usually acceptable. However, if using WFS and especially if making use of BBOX filtering capabilities, this flag should not be set.

8.1.4 Publishing a PostGIS view

Publishing a view follows the same process as publishing a table. The only additional step is to manually ensure that the view has an entry in the `geometry_columns` table.

For example consider a table with the schema:

```
my_table( id int PRIMARY KEY, name VARCHAR, the_geom GEOMETRY )
```

Consider also the following view:

```
CREATE VIEW my_view as SELECT id, the_geom FROM my_table;
```

Before this view can be served by GeoServer, the following step is necessary to manually create the `geometry_columns` entry:

```
INSERT INTO geometry_columns VALUES (’’,’public’,’my_view’,’my_geom’, 2, 4326, ’POINT’ );
```

8.1.5 Performance considerations

GEOS

GEOS (Geometry Engine, Open Source) is an optional component of a PostGIS installation. It is recommended that GEOS be installed with any PostGIS instance used by GeoServer, as this allows GeoServer to make use of its functionality when doing spatial operations. When GEOS is not available, these operations are performed internally which can result in degraded performance.

Spatial indexing

It is strongly recommended to create a spatial index on tables with a spatial component (i.e. containing a geometry column). Any table of which does not have a spatial index will likely respond slowly to queries.

8.1.6 Common problems

Primary keys

In order to enable transactional extensions on a table (for transactional WFS), the table must have a primary key. A table without a primary key is considered read only to GeoServer.

8.2 H2

Note: GeoServer does not come built-in with support for H2; it must be installed through an extension. Proceed to *Installing the H2 extension* for installation details.

8.2.1 Installing the H2 extension

1. Download the H2 extension from the GeoServer download page.

   Warning: Make sure to match the version of the extension to the version of the GeoServer instance!

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

8.2.2 Adding an H2 data store

Once the extension is properly installed *H2* will be an option in the *Vector Data Sources* list when creating a new data store.

Figure 8.3: *H2 in the list of vector data stores*

8.2.3 Configuring an H2 data store

8.2.4 Configuring an H2 data store with JNDI

Other data sources are supplied as GeoServer extensions. Extensions are downloadable modules that add functionality to GeoServer. Extensions are available at the GeoServer download page.

Warning: The extension version must match the version of the GeoServer instance.

Figure 8.4: *Configuring an H2 data store*

8.3 ArcSDE

Note: ArcSDE support is not enabled by default and requires the ArcSDE extension to be installed prior to use. Please see the section on *Installing the ArcSDE extension* for details.

ESRI's ArcSDE is a spatial engine that runs on top of a relational database such as Oracle or SQL Server. GeoServer with the ArcSDE extension supports ArcSDE **versions 9.2 and 9.3**. It has been tested with **Oracle 10g** and **Microsoft SQL Server 2000 Developer Edition**. The ArcSDE extension is based on the GeoTools ArcSDE driver and uses the ESRI Java API libraries. See the GeoTools ArcSDE page for more technical details.

There are two types of ArcSDE data that can be added to GeoServer: **vector** and **raster**.

8.3.1 Vector support

ArcSDE provides efficient access to vector layers, ("featureclasses" in ArcSDE jargon), over a number of relational databases. GeoServer can set up featuretypes for registered ArcSDE featureclasses and spatial views. For versioned ArcSDE featureclasses, GeoServer will work on the default database version, for both read and write access.

Transactional support is enabled for featureclasses with a properly set primary key, regardless if the featureclass is managed by a user or by ArcSDE. If a featureclass has no primary key set, it will be available as read-only.

8.3.2 Raster support

ArcSDE provides efficient access to multi-band rasters by storing the raw raster data as database blobs, dividing it into tiles and creating a pyramid. It also allows a compression method to be set for the tiled blob data and an interpolation method for the pyramid resampling.

All the bands comprising a single ArcSDE raster layer must have the same pixel depth, which can be one of 1, 4, 8, 16, and 32 bits per sample for integral data types. For 8, 16 and 32 bit bands, they may be signed or unsigned. 32 and 64 bit floating point sample types are also supported.

ArcSDE rasters may also be color mapped, as long as the raster has a single band of data typed 8 or 16 bit unsigned.

Finally, ArcSDE supports raster catalogs. A raster catalog is a mosaic of rasters with the same spectral properties but instead of the mosaic being precomputed, the rasters comprising the catalog are independent and the mosaic work performed by the application at runtime.

Technical Detail	**Status**
Compression methods	LZW, JPEG
Number of bands	Any number of bands except for 1 and 4 bit rasters (supported for single-band only).
Bit depth for color-mapped rasters	8 bit and 16 bit
Raster Catalogs	Any pixel storage type

8.3.3 Installing the ArcSDE extension

Warning: Due to licensing requirements, not all files are included with the extension. To install ArcSDE support, it is necessary to download additional files. **Just installing the ArcSDE extension will have no effect.**

GeoServer files

1. Download the ArcSDE extension from the GeoServer download page.

   Warning: Make sure to match the version of the extension to the version of the GeoServer instance!

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

Required external files

There are two files that are required but are not packaged with the GeoServer extension:

File	Notes
`jsde_sdk.jar`	Also known as `jsde##_sdk.jar` where `##` is the version number, such as `92` for ArcSDE version 9.2
`jpe_sdk.jar`	Also known as `jpe##_sdk.jar` where `##` is the version number, such as `92` for ArcSDE version 9.2

You should always make sure the `jsde_sdk.jar` and `jpe_sdk.jar` versions match your ArcSDE server version, including service pack, although client jar versions higher than the ArcSDE Server version usually work just fine.

These two files are available on your installation of the ArcSDE Java SDK from the ArcSDE insatallation media (usually `C:\Program Files\ArcGIS\ArcSDE\lib`). They may also be available on ESRI's website if there's a service pack containing them, but this is not guaranteed. To download these files from ESRI's website:

1. Navigate to http://support.esri.com/index.cfm?fa=downloads.patchesServicePacks.listPatches&PID=66
2. Find the link to the latest service pack for your version of ArcSDE
3. Scroll down to *Installing this Service Pack* → *ArcSDE SDK* → *UNIX* (regardless of your target OS)
4. Download any of the target files (but be sure to match 32/64 bit to your OS)
5. Open the archive, and extract the appropriate JARs.

Note: The JAR files may be in a nested archive inside this archive.

Note: The `icu4j##.jar` may also be on your ArcSDE Java SDK installation folder, but it is already included as part of the the GeoServer ArcSDE extension and is not necessary to install separately.

1. When downloaded, copy the two files to the `WEB-INF/lib` directory of the GeoServer installation.

After all GeoServer files and external files have been downloaded and copied, restart GeoServer.

8.3.4 Adding an ArcSDE vector data store

In order to serve vector data layers, it is first necessary to register the ArcSDE instance as a data store in GeoServer. Navigate to the **New data source** page, accessed from the *Stores* page in the *Web Administration Interface*. and an option for **ArcSDE** will be in the list of *Vector Data Stores*.

Note: If `ArcSDE` is not an option in the **Feature Data Set Description** drop down box, the extension is not properly installed. Please see the section on *Installing the ArcSDE extension*.

Figure 8.5: *ArcSDE in the list of data sources*

8.3.5 Configuring an ArcSDE vector data store

The next page contains configuration options for the ArcSDE vector data store. Fill out the form, then click *Save*.

Option	Required?	Description
`Feature Data Set ID`	N/A	The name of the data store as set on the previous page.
`Enabled`	N/A	When this box is checked the data store will be available to GeoServer
`Namespace`	Yes	The namespace associated with the data store.
`Description`	No	A description of the data store.
`server`	Yes	The URL of the ArcSDE instance.
`port`	Yes	The port that the ArcSDE instance is set to listen to. Default is 5151.
`instance`	No	The name of the specific ArcSDE instance, where applicable, depending on the underlying database.
`user`	Yes	The username to authenticate with the ArcSDE instance.
`password`	No	The password associated with the above username for authentication with the ArcSDE instance.
`pool.minConnections`	No	Connection pool configuration parameters. See the *Database Connection Pooling* section for details.
`pool.maxConnections`	No	Connection pool configuration parameters. See the *Database Connection Pooling* section for details.
`pool.timeOut`	No	Connection pool configuration parameters. See the *Database Connection Pooling* section for details.

You may now add featuretypes as you would normally do, by navigating to the *New Layer* page, accessed from the *Layers* page in the *Web Administration Interface*.

8.3.6 Configuring an ArcSDE vector data store with Direct Connect

ESRI Direct Connect[ESRI DC] allows clients to directly connect to an SDE GEODB 9.2+ without a need of an SDE server instance, and is recommended for high availability environments, as it removes the ArcSDE gateway server as a single point of failure. ESRI DC needs additional platform dependent binary drivers and a working Oracle Client ENVIRONMENT (if connecting to an ORACLE DB). See Properties of a direct connection to an ArcSDE geodatabase in the ESRI ArcSDE documentation for more information on Direct

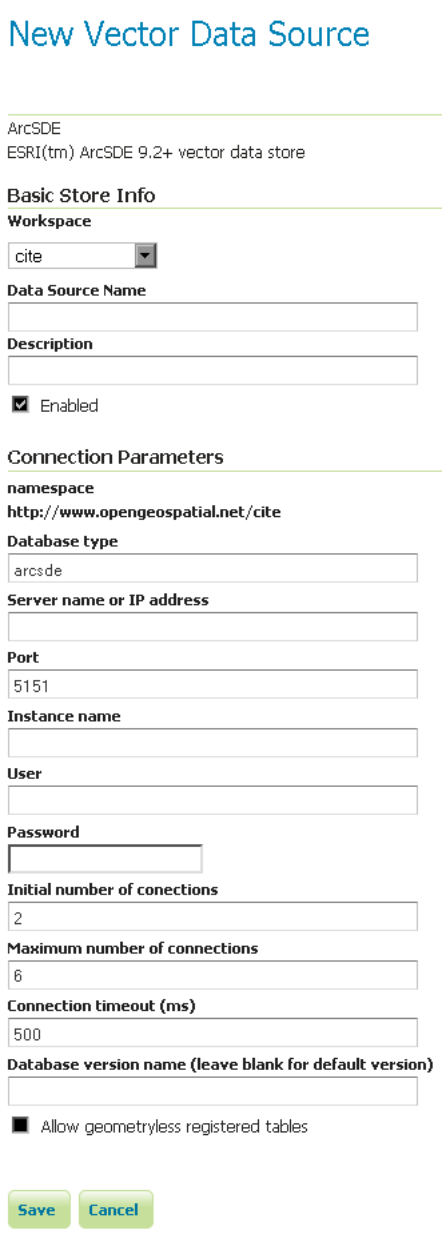


Figure 8.6: *Configuring a new ArcSDE data store*

Connect, and Setting up clients for a direct connection for information about connecting to the different databases supported by ArcSDE.

The GeoServer configuration parameters are the same as in the *Configuring an ArcSDE vector data store* section above, with a couple differences in how to format the parameters:

- server: In ESRI Direct Connect Mode a value must be given or the Direct Connect Driver will throw an error, so just put a 'none' there - any String will work!
- port: In ESRI Direct Connect Mode the port has a String representation: *sde:oracle10g*, *sde:oracle11g:/:test*, etc. For further information check ArcSDE connection syntax at the official ArcSDE documentation from ESRI.
- instance: In ESRI Direct Connect Mode a value must be given or the Direct Connect Driver will throw an error, so just put a 'none' there - any String will work!
- user: The username to authenticate with the geo database.
- password: The password associated with the above username for authentication with the geo database.

Note: Be sure to assemble the password like: password@<Oracle Net Service name> for Oracle

You may now add featuretypes as you would normally do, by navigating to the New Layer page, accessed from the Layers page in the Web Administration Interface.

8.3.7 Adding an ArcSDE vector data store with JNDI

8.3.8 Configuring an ArcSDE vector data store with JNDI

8.3.9 Adding an ArcSDE raster coveragestore

In order to serve raster layers (or coverages), it is first necessary to register the ArcSDE instance as a store in GeoServer. Navigate to the **Add new store** page, accessed from the *Stores* page in the *Web Administration Interface* and an option for **ArcSDE Raster Format** will be in list.

Note: If `ArcSDE Raster Format` is not an option in the **Coverage Data Set Description** drop down box, the extension is not properly installed. Please see the section on *Installing the ArcSDE extension*.

Figure 8.7: *ArcSDE Raster in the list of data sources*

8.3.10 Configuring an ArcSDE raster coveragestore

The next page contains configuration options for the ArcSDE instance. Fill out the form, then click *Save*.

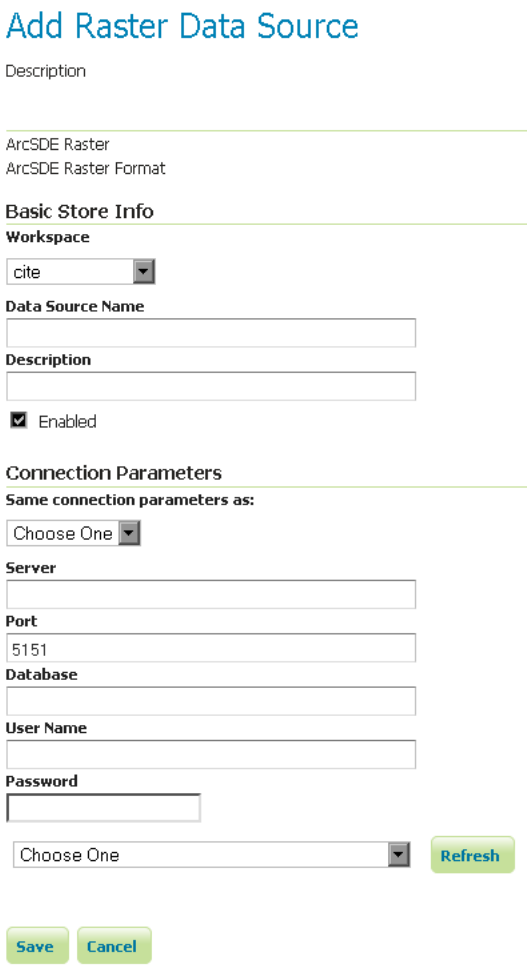


Figure 8.8: *Configuring a new ArcSDE coveragestore*

Option	Required?	Description
`Coverage Data Set ID`	N/A	The name of the coveragestore as set on the previous page.
`Enabled`	N/A	When this box is checked the coveragestore will be available to GeoServer.
`Namespace`	Yes	The namespace associated with the coveragestore.
`Type`	No	The type of coveragestore. Leave this to say `ArcSDE Raster`.
`URL`	Yes	The URL of the raster, of the form `sde://<user>:<pwd>@<server>/#<tableName>`.
`Description`	No	A description of the coveragestore.

You may now add coverages as you would normally do, by navigating to the **Add new layer** page, accessed from the *Layers* page in the *Web Administration Interface*.

8.4 DB2

Note: GeoServer does not come built-in with support for DB2; it must be installed through an extension. Proceed to *Installing the DB2 extension* for installation details.

The IBM DB2 UDB database is a commercial relational database implementing ISO SQL standards and is similar in functionality to Oracle, SQL Server, MySQL, and PostgreSQL. The DB2 Spatial Extender is a no-charge licensed feature of DB2 UDB which implements the OGC specification "Simple Features for SQL using types and functions" and the ISO "SQL/MM Part 3 Spatial" standard.

A trial copy of DB2 UDB and Spatial Extender can be downloaded from: http://www-306.ibm.com/software/data/db2/udb/edition-pde.html . There is also an "Express-C" version of DB2, that is free, comes with spatial support, and has no limits on size. It can be found at: http://www-306.ibm.com/software/data/db2/express/download.html

8.4.1 Installing the DB2 extension

Warning: Due to licensing requirements, not all files are included with the extension. To install DB2 support, it is necessary to download additional files. **Just installing the DB2 extension will have no effect.**

GeoServer files

1. Download the DB2 extension from the GeoServer download page.

   Warning: Make sure to match the version of the extension to the version of the GeoServer instance!

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

Required external files

There are two files that are required but are not packaged with the GeoServer extension: `db2jcc.jar` and `db2jcc_license_cu.jar`. These files should be available in the `java` subdirectory of your DB2

installation directory. Copy these files to the `WEB-INF/lib` directory of the GeoServer installation.

After all GeoServer files and external files have been downloaded and copied, restart GeoServer.

8.4.2 Adding a DB2 data store

When properly installed, *DB2* will be an option in the *Vector Data Sources* list when creating a new data store.

Figure 8.9: *DB2 in the list of raster data stores*

8.4.3 Configuring a DB2 data store

8.4.4 Configuring a DB2 data store with JNDI

8.4.5 Notes on usage

DB2 schema, table, and column names are all case-sensitive when working with GeoTools/GeoServer. When working with DB2 scripts and the DB2 command window, the default is to treat these names as upper-case unless enclosed in double-quote characters.

8.5 MySQL

Note: GeoServer does not come built-in with support for MySQL; it must be installed through an extension. Proceed to *Installing the MySQL extension* for installation details.

Warning: Currently the MySQL extension is unmaintained and carries unsupported status. While still usable, do not expect the same reliability as with other extensions.

MySQL is an open source relational database with some limited spatial functionality.

8.5.1 Installing the MySQL extension

1. Download the MySQL extension from the GeoServer download page.

   Warning: Make sure to match the version of the extension to the version of the GeoServer instance!

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

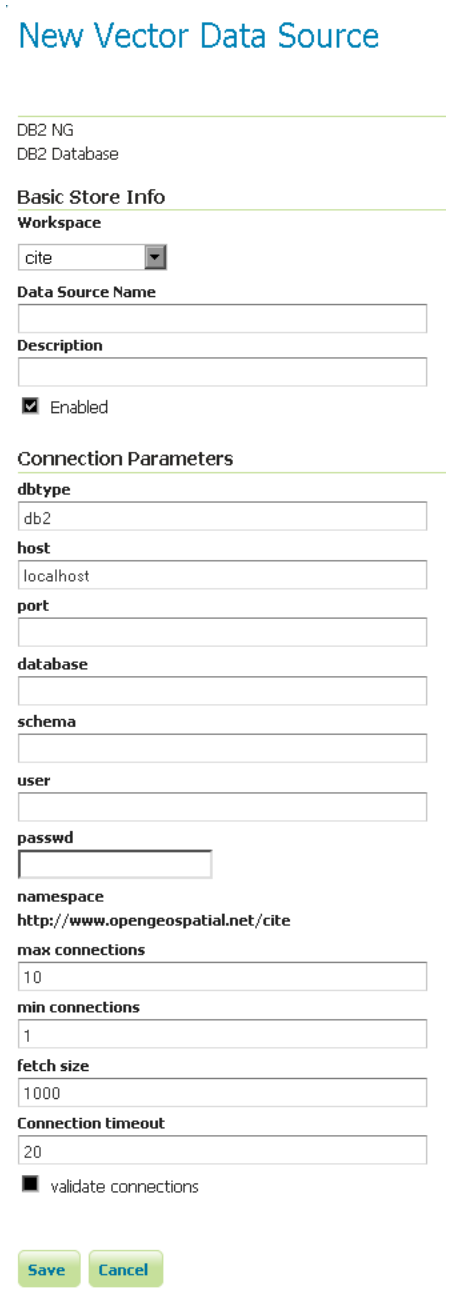


Figure 8.10: *Configuring a DB2 data store*

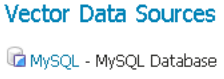


Figure 8.11: *MySQL in the list of data sources*

8.5.2 Adding a MySQL database

Once the extension is properly installed `MySQL` will show up as an option when creating a new data store.

8.5.3 Configuring a MySQL data store

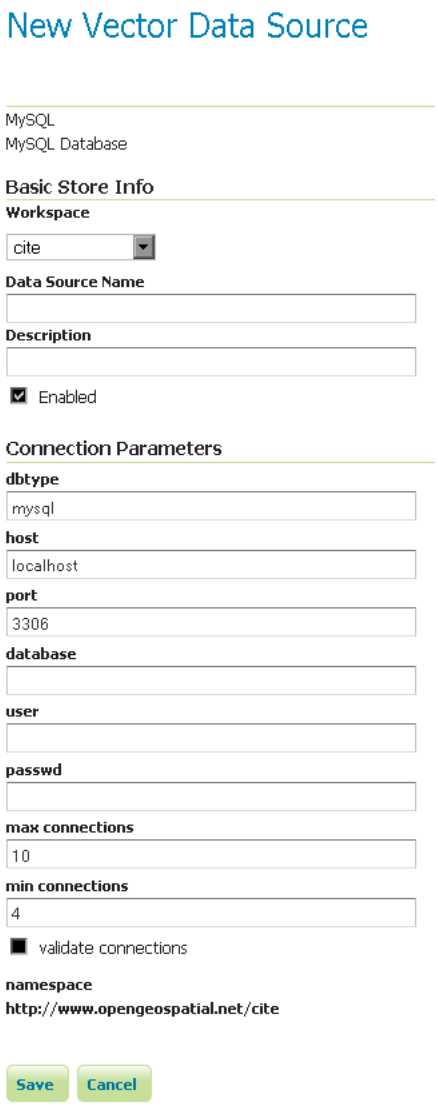


Figure 8.12: *Configuring a MySQL data store*

`host`	The mysql server host name or ip address.
`port`	The port on which the mysql server is accepting connections.
`database`	The name of the database to connect to.
`user`	The name of the user to connect to the mysql database as.
`password`	The password to use when connecting to the database. Left blank for no password.

`max connections`
`min connections`
`validate connections`
	Connection pool configuration parameters. See the *Database Connection Pooling* section for details.

8.6 Oracle

Note: GeoServer does not come built-in with support for Oracle; it must be installed through an extension. Proceed to *Installing the Oracle extension* for installation details.

Oracle Spatial and Locator are the spatial components of Oracle. **Locator** is provided with all Oracle versions, but has limited spatial functions. **Spatial** is Oracle's full-featured spatial offering, but requires a specific license to use.

8.6.1 Installing the Oracle extension

1. Download the Oracle extension from the GeoServer download page.

   Warning: Make sure to match the version of the extension to the version of the GeoServer instance!

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

8.6.2 Consider replacing the Oracle JDBC driver

The Oracle data store zip file comes with `ojdbc4.jar`, an old, Oracle 10 compatible JDBC driver that normally works fine with 11g as well. However, minor glitches have been observed with 11g (issues computing layer bounds when session initiation scripts are in use) and the driver has not been tested with 12i.

If you encounter functionality or performance issues it is advices to remote this driver and download the latest version from the Oracle web site.

8.6.3 Adding an Oracle datastore

Once the extension is properly installed *Oracle* appears as an option in the *Vector Data Sources* list when creating a new data store.

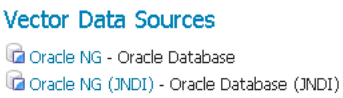


Figure 8.13: *Oracle in the list of data sources*

8.6.4 Configuring an Oracle datastore

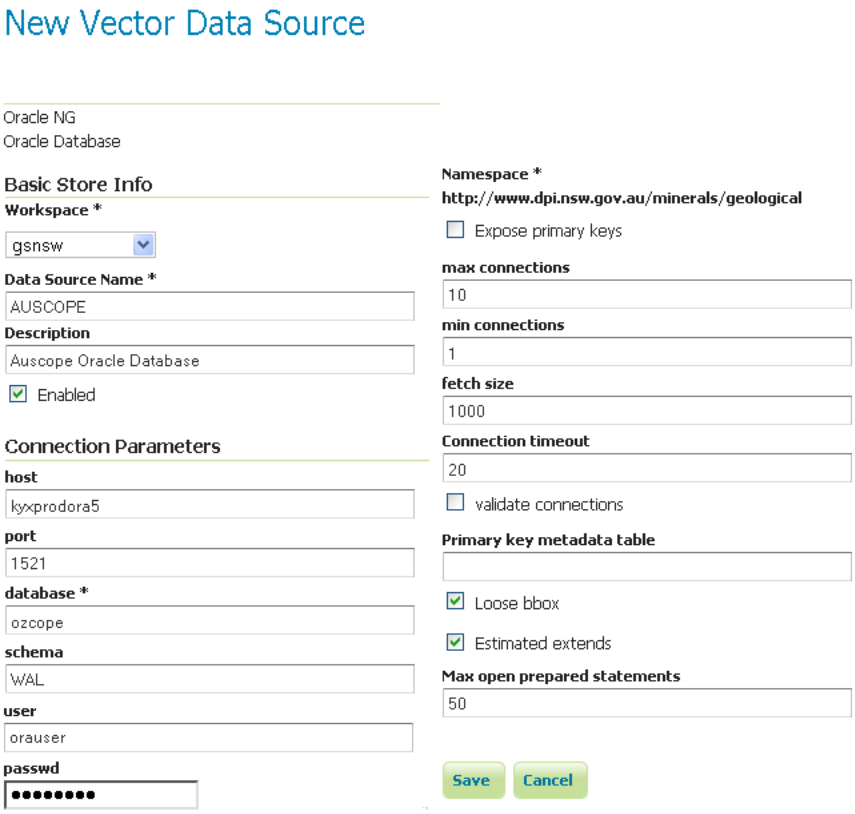


Figure 8.14: *Configuring an Oracle datastore*

Option	Description
`host`	The Oracle server host name or IP address.
`port`	The port on which the Oracle server is accepting connections (often this is port 1521).
`database`	The name of the database to connect to. By default this is interpreted as a SID name. To connect to a Service, prefix the name with a `/`.
`schema`	The database schema to access tables from. Setting this value greatly increases the speed at which the data store displays its publishable tables and views, so it is advisable to set this.
`user`	The name of the user to use when connecting to the database.
`password`	The password to use when connecting to the database. Leave blank for no password.
`max connections` `min connections` `fetch size` `Connection timeout` `validate connections`	Connection pool configuration parameters. See *Database Connection Pooling* for details.
`Loose bbox`	Controls how bounding box comparisons are made against geometries in the database. See the *Using loose bounding box* section below.

8.6.5 Connecting to an Oracle cluster

In order to connect to an Oracle RAC one can use an almost full JDBC url as the `database`, provided it starts with `(` it will be used verbatim and options "host" and "port" will be ignored. Here is an example "database" value used to connect to an Oracle RAC:

```
(DESCRIPTION=(LOAD_BALANCE=on)(ADDRESS=(PROTOCOL=TCP)(HOST=host1) (PORT=1521))(ADDRESS=(PROTOCOL=TCP)
```

More information about this syntax can be found in the Oracle documentation.

Connecting to a SID or a Service

Recent versions of Oracle support connecting to a database via either a SID name or a Service name. A SID connection descriptor has the form: `host:port:database`, while a Service connection descriptor has the format `host:port/database`. GeoServer uses the SID form by default. To connect via a Service, prefix the `database` name configuration entry with a `/`.

Connecting to database through LDAP

For instance if you want to establish a connection with the jdbc thin driver through LDAP, you can use following connect string for the input field `database` `ldap://[host]:[Port]/[db],cn=OracleContext,dc=[oracle_ldap_context]`.

If you are using referrals, enable it by placing a jndi.properties file in geoserver's CLASSPATH, which is in geoserver/WEB-INF/classes. This property file contains:

```
java.naming.referral=follow
```

Using loose bounding box

When the `Loose bbox` option is set, only the bounding box of database geometries is used in spatial queries. This results in a significant performance gain. The downside is that some geometries may be reported as intersecting a BBOX when they actually do not.

If the primary use of the database is through the *Web Map Service* this flag can be set safely, since querying more geometries does not have any visible effect. However, if using the *Web Feature Service* and making use of BBOX filtering capabilities, this flag should not be set.

Using the geometry metadata table

The Oracle data store by default looks at the `MDSYS.USER_SDO*` and `MDSYS.ALL_SDO*` views to determine the geometry type and native SRID of each geometry column. Those views are automatically populated with information about the geometry columns stored in tables that the current user owns (for the `MDSYS.USER_SDO*` views) or can otherwise access (for the `MDSYS.ALL_SDO*` views).

There are a few issues with this strategy:

- if the connection pool user cannot access the tables (because *impersonation* is used) the MDSYS views will be empty, making it impossible to determine both the geometry type and the native SRID
- the geometry type can be specified only while building the spatial indexes, as an index constraint. However such information is often not included when creating the indexes
- the views are populated dynamically based on the current user. If the database has thousands of tables and users the views can become very slow

Starting with GeoServer 2.1.4 the administrator can address the above issues by manually creating a geometry metadata table describing each geometry column. Its presence is indicated via the Oracle datastore connection parameter named *Geometry metadata table* (which may be a simple table name or a schema-qualified one). The table has the following structure (the table name is flexible, just specify the one chosen in the data store connection parameter):

```
CREATE TABLE GEOMETRY_COLUMNS(
   F_TABLE_SCHEMA VARCHAR(30) NOT NULL,
   F_TABLE_NAME VARCHAR(30) NOT NULL,
   F_GEOMETRY_COLUMN VARCHAR(30) NOT NULL,
   COORD_DIMENSION INTEGER,
   SRID INTEGER NOT NULL,
   TYPE VARCHAR(30) NOT NULL,
   UNIQUE(F_TABLE_SCHEMA, F_TABLE_NAME, F_GEOMETRY_COLUMN),
   CHECK(TYPE IN ('POINT','LINE', 'POLYGON', 'COLLECTION', 'MULTIPOINT', 'MULTILINE', 'MULTIPOLYGON',
```

When the table is present the store first searches it for information about each geometry column to be classified, and falls back on the MDSYS views only if the table does not contain any information.

8.6.6 Configuring an Oracle database with JNDI

See *Setting up a JNDI connection pool with Tomcat* for a guide on setting up an Oracle connection using JNDI.

8.7 Microsoft SQL Server and SQL Azure

Note: GeoServer does not come built-in with support for SQL Server; it must be installed through an extension. Proceed to *Installing the SQL Server extension* for installation details.

Microsoft's SQL Server is a relational database with spatial functionality. SQL Azure is the database option provided in the Azure cloud solution which is in many respects similar to SQL Server 2008.

8.7.1 Supported versions

The extension supports SQL Server 2008 and SQL Azure.

8.7.2 Installing the SQL Server extension

Warning: Due to licensing requirements, not all files are included with the extension. To install SQL Server support, it is necessary to download additional files.

GeoServer files

1. Download the SQL Server extension from the GeoServer download page.

   Warning: Make sure to match the version of the extension to the version of the GeoServer instance!

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

Microsoft files

1. Navigate to Microsoft's JDBC driver for SQL Server and SQL Azure download page.
2. Extract the contents of the archive
3. If you are running Java 6 or above, copy the file `sqljdbc4.jar` to the `WEB-INF/lib` directory of the GeoServer installation. If you are running Java 5 instead (supported up to GeoServer 2.1.x) copy the file `sqljdbc.jar` to the `WEB-INF/lib` directory
4. For GeoServer installed on Windows, copy `\x86\sqljdbc_auth.dll` and `\x86\sqljdbc_xa.dll` to `C:\Windows\System32`

8.7.3 Adding a SQL Server database

Once the extension is properly installed `SQL Server` will show up as an option when creating a new data store.

Figure 8.15: *SQL Server in the list of vector data sources*

8.7.4 Configuring a SQL Server data store

`host`	The sql server instance host name or ip address, only. Note that `server\instance` notation is not accepted - specify the port below, instead, if you have a non-default instance.
`port`	The port on which the SQL server instance is accepting connections. See the *note* below.
`database`	The name of the database to connect to. Might be left blank if the user connecting to SQL Server has a "default database" set in the user configuration
`schema`	The database schema to access tables from (optional).
`user`	The name of the user to connect to the oracle database as.
`password`	The password to use when connecting to the database. Leave blank for no password.
`max connections` `min connections`	Connection pool configuration parameters. See the *Database Connection Pooling* section for details. If you are connecting to SQL Azure make sure to set the `validate connections` flag as SQL Azure closes inactive connections after a very short delay.

Determining the port used by the SQL Server instance

You can determine the port in use by connecting to your SQL server instance using some other software, and then using **netstat** to display details on network connections. In the following example on a Windows PC, the port is 2646

```
C:\>netstat -a | find "sql1"
TCP    DPI908194:1918    maittestsql1.dpi.nsw.gov.au:2646    ESTABLISHED
```

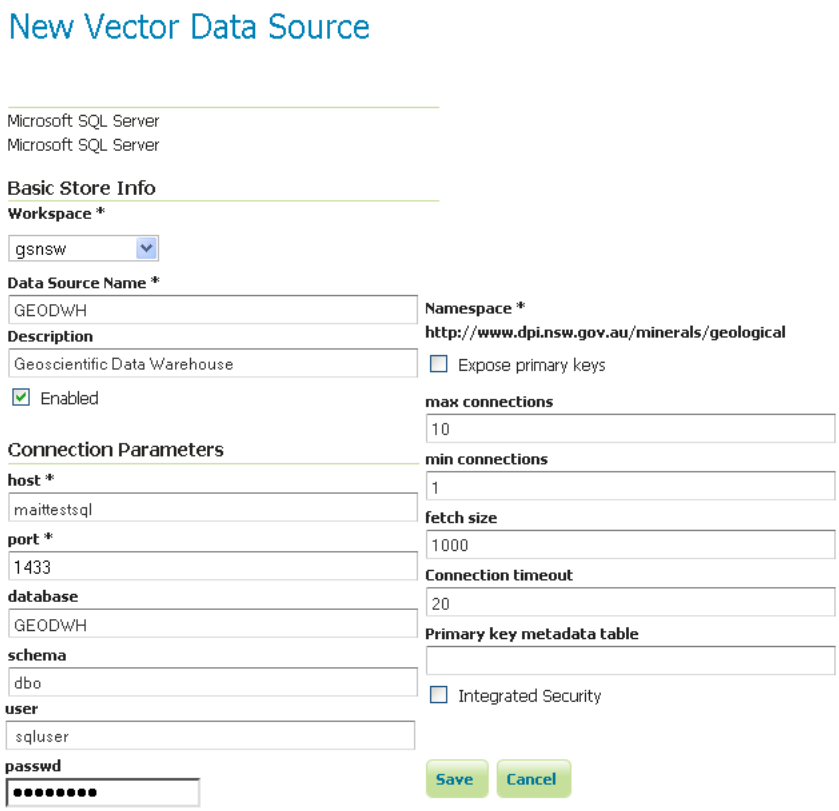


Figure 8.16: *Configuring a SQL Server data store*

Using the geometry metadata table

The SQL server data store can determine the geometry type and native SRID of a particular column only by data inspection, by looking at the first row in the table. Of course this is error prone, and works only if there is data in the table. The administrator can address the above issue by manually creating a geometry metadata table describing each geometry column. Its presence is indicated via the SQL Server datastore connection parameter named *Geometry metadata table* (which may be a simple table name or a schema-qualified one). The table has the following structure (the table name is flexible, just specify the one chosen in the data store connection parameter):

```
CREATE TABLE GEOMETRY_COLUMNS(
   F_TABLE_SCHEMA VARCHAR(30) NOT NULL,
   F_TABLE_NAME VARCHAR(30) NOT NULL,
   F_GEOMETRY_COLUMN VARCHAR(30) NOT NULL,
   COORD_DIMENSION INTEGER,
   SRID INTEGER NOT NULL,
   TYPE VARCHAR(30) NOT NULL,
   UNIQUE(F_TABLE_SCHEMA, F_TABLE_NAME, F_GEOMETRY_COLUMN),
   CHECK(TYPE IN ('POINT','LINE', 'POLYGON', 'COLLECTION', 'MULTIPOINT', 'MULTILINE', 'MULTIPOLYGON',
```

When the table is present the store first searches it for information about each geometry column to be classified, and falls back on data inspection only if the table does not contain any information.

8.8 Teradata

Note: Teradata database support is not enabled by default and requires the Teradata extension to be installed prior to use. Please see the section on *Installing the Teradata extension* for details.

The Teradata Database is a commercial relational database (RDBMS) that specializes in parallel processing and scalability. From version 12.0, Teradata has added geospatial support, closely following the SQL/MM standard (SQL Multimedia and Applications Packages). Geospatial support was available through an add-on in version 12.0 and became standard in version 13.0.

GeoServer connects to a Teradata database via JDBC.

For more information on Teradata and the Teradata Database system, please go to http://www.teradata.com.

8.8.1 Compatibility

The GeoServer Teradata extension is compatible with GeoServer 2.1.1 and higher. GeoServer can connect to Teradata databases version 12.0 or higher. Version 12.0 of the Teradata Database requires the optional geospatial extension to be installed.

8.8.2 Read/write access

The Teradata datastore in GeoServer supports full transactional capabilities, including feature creation, editing, and deleting.

To support editing, a table must have one of the following:

- a primary key
- a unique primary index

- an identity (sequential) column

Note: It is not recommended to solely use an identity column, as spatial index triggers are not supported when referencing an identity column. See the section on Spatial Indexes for more details.

8.8.3 Query Banding

The GeoServer Teradata extension supports Query Banding. Query Banding is a feature which allows any application to associate context information with each query it issues to the database. In practice this can be used for purposes of workload management (i.e. request prioritization), debugging, and logging.

GeoServer sends the following information as part of a standard request:

- Name of application (i.e. GeoServer)
- Authenticated username (if set up)
- Hostname (if available)
- Type of statement (i.e. "SELECT", "INSERT", "DELETE")

It is not possible to modify this information from within GeoServer.

8.8.4 Spatial indexes

GeoServer will read from a spatial index if its exists. The convention for a spatial index table name is:

```
[TABLENAME]_[GEOMETRYCOLUMN]_idx
```

So for a layer called "STATES" with a geometry column called "GEOM", the index table should be called *STATES_GEOM_idx*.

Warning: Make sure to match the case of all tables and columns. If the geometry column is called "GEOM" (upper case) and the index created is called *STATES_geom_idx* (lower case), the index will not be properly linked to the table.

This index table should contain two columns:

- A column that maps to the primary key of the spatial data table
- The tessellation cell ID (cellid)

The tessellation cell ID is the ID of the cell where that feature is contained.

8.8.5 Geometry column

As per the SQL/MM standard, in order to make a Teradata table spatially enabled, an entry needs to be created for that table in the `geometry_columns` table. This table is stored, like all other spatially-related tables, in the SYSSPATIAL database.

8.8.6 Tessellation

Tessellation is the name of Teradata's spatial index. In order to activate tessellation for a given layer, an entry (row) needs to be placed in the `SYSSPATIAL.tessellation` table. This table should have the following schema:

Table name	Type	Description
F_TABLE_SCHEMA	varchar	Name of the spatial database/schema containing the table
F_TABLE_NAME	varchar	Name of the spatial table
F_GEOMETRY_COLUMN	varchar	Column that contains the spatial data
U_XMIN	float	Minimum X value for the tessellation universe
U_YMIN	float	Minimum Y value for the tessellation universe
U_XMAX	float	Maximum X value for the tessellation universe
U_YMAX	float	Maximum Y value for the tessellation universe
G_NX	integer	Number of X grids
G_NY	integer	Number of Y grids
LEVELS	integer	Number of levels in the grid
SCALE	float	Scale value for the grid
SHIFT	float	Shift value for the grid

Warning: The tessellation table values are case sensitive and so must match the case of the tables and columns.

8.8.7 Installing the Teradata extension

Teradata database support is not enabled by default and requires the GeoServer Teradata extension to be installed prior to use. In addition to this extension, an additional artifact will need to be downloaded from the Teradata website.

GeoServer artifacts

1. Download the Teradata extension from the download page that matches your version of GeoServer. The extension is listed at the bottom of the download page under *Extensions*.

   Warning: Make sure to match the version of the extension to the version of GeoServer!

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

Teradata artifacts

In addition to the GeoServer artifacts, it is also necessary to download the Teradata JDBC driver. This file cannot be redistributed and so must be downloaded directly from the Teradata website.

1. Download the Teradata JDBC driver at https://downloads.teradata.com/download/connectivity/jdbc-driver.

   Note: You will need to log in to Teradata's site in order to download this artifact.

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

When all files have been downloaded and extracted, restart GeoServer. To verify that the installation was successful, see the section on *Adding a Teradata datastore*.

Note: The full list of files required are:

- `gt-jdbc-teradata-<version>.jar`
- `tdgssconfig.jar`

- `terajdbc4.jar`

8.8.8 Adding a Teradata datastore

Once the extension has been added, it will now be possible to load an existing Teradata database as a store in GeoServer. In the *Web Administration Interface*, click on *Stores* then go to *Add a new Store*. You will see a option, under *Vector Data Stores*, for *Teradata*. Select this option.

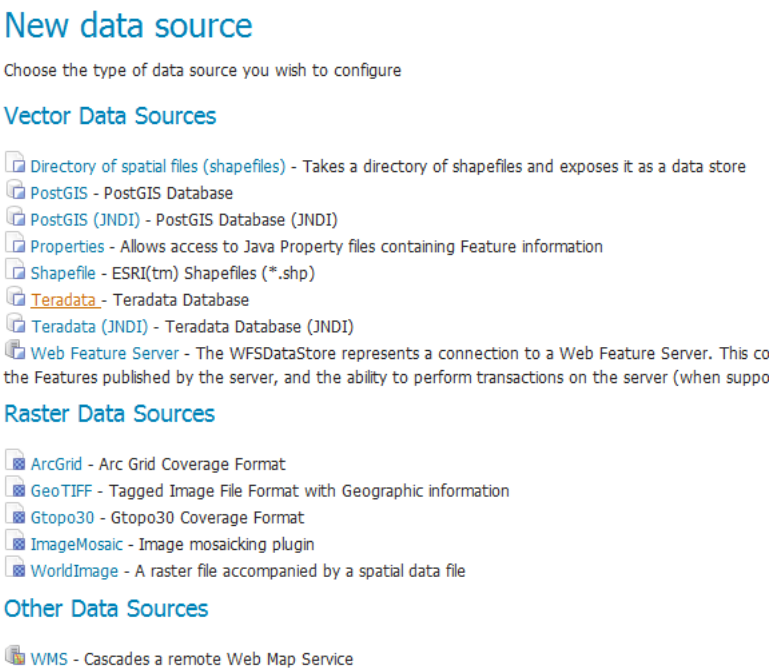


Figure 8.17: *Teradata in the list of readable stores*

Note: If you don't Teradata in this list, the extension has not been installed properly. Please ensure that the steps in the *Installing the Teradata extension* have been followed correctly.

On the next screen, enter in the details on how to connect to the Teradata database. You will need to include the following information:

Option	Description
Workspace	Name of the workspace to contain the database. This will also be the prefix of any layers server from tables in the database.
Data Source Name	Name of the database in GeoServer. This can be different from the name of the Teradata database, if desired.
Description	Description of the database/store.
Enabled	Enables the store. If disabled, no layers from the database will be served.
host	Host name where the database exists. Can be a URL or IP address.
port	Port number on which to connect to the above host.
database	Name of the Teradata database.
user	User name to connect to use to connect to the database.
passwd	Password associated with the above user.
namespace	Namespace to be associated with the database. This field is altered automatically by the above Workspace field.
Expose primary keys	Exposes primary key as a standard attribute.
max connections	Maximum amount of open/pooled connections to the database.
min connections	Minimum number of open/pooled connections.
fetch size	Number of records read with each interaction with the database.
Connection timeout	Time (in seconds) the connection pool will wait before timing out.
validate connections	Checks the connection is alive before using it.
Primary key metadata table	Name of primary key metadata table to use if unable to determine the primary key of a table.
Loose bbox	If checked, performs only the primary filter on the bounding box.
tessellationTable	The name of the database table that contains the tessellations
estimatedBounds	Enables using the geometry_columns/tessellation table bounds as an estimation instead of manual calculation.
Max open prepared statements	The maximum number of prepared statements.

When finished, click *Save*.

Using JNDI

GeoServer can also connect to a Teradata database using JNDI (Java Naming and Directory Interface).

To begin, in the *Web Administration Interface*, click on *Stores* then go to *Add a new Store*. You will see a option, under *Vector Data Stores*, for *Teradata (JNDI)*. Select this option.

On the next screen, enter in the details on how to connect to the Teradata database. You will need to include the following information:

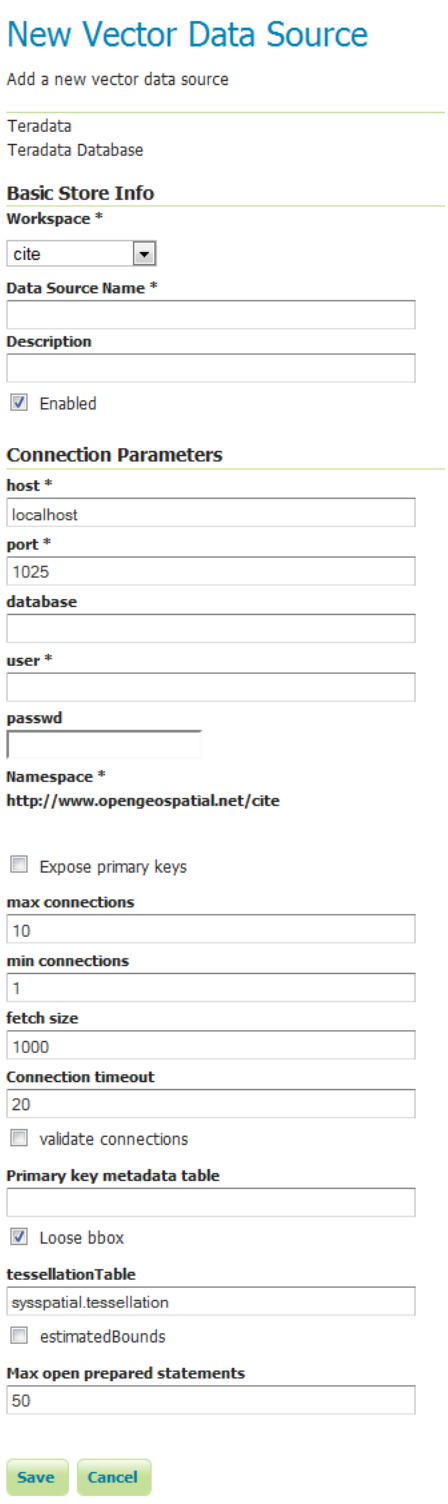


Figure 8.18: *Adding a Teradata store*

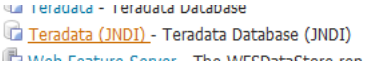


Figure 8.19: *Teradata (JNDI) in the list of readable stores*

Option	Description
Workspace	Name of the workspace to contain the database. This will also be the prefix of any layers server from tables in the database.
Data Source Name	Name of the database in GeoServer. This can be different from the name of the Teradata database, if desired.
Description	Description of the database/store.
Enabled	Enables the store. If disabled, no layers from the database will be served.
jndiReference-Name	JNDI path to the database.
schema	Schema for the above database.
namespace	Namespace to be associated with the database. This field is altered by changing the workspace name.
Expose primary keys	Exposes primary key as a standard attribute.
Primary key metadata table	Name of primary key metadata table to use if unable to determine the primary key of a table.
Loose bbox	If checked, performs only the primary filter on the bounding box.

When finished, click *Save*.

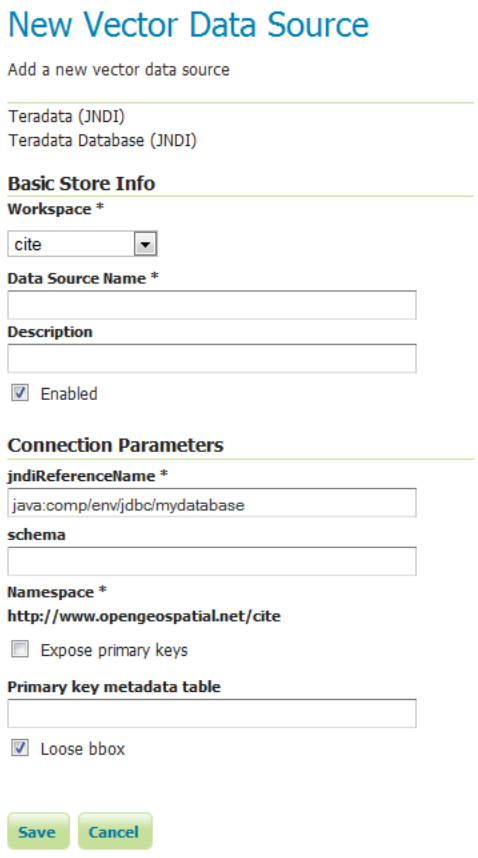


Figure 8.20: *Adding a Teradata store with JNDI*

8.8.9 Adding layers

One the store has been loaded into GeoServer, the process for loading data layers from database tables is the same as any other database source. Please see the *Layers* section for more information.

Note: Only those database tables that have spatial information and an entry in the `SYSSPATIAL.geometry_columns` table can be served through GeoServer.

GeoServer provides extensive facilities for controlling how databases are accessed. These are covered in the following sections.

8.9 Database Connection Pooling

When serving data from a spatial database *connection pooling* is an important aspect of achieving good performance. When GeoServer serves a request that involves loading data from a database table, a connection must first be established with the database. This connection comes with a cost as it takes time to set up such a connection.

The purpose served by a connection pool is to maintain connection to an underlying database between requests. The benefit of which is that connection setup only need to occur once on the first request. Subsequent requests use existing connections and achieve a performance benefit as a result.

Whenever a data store backed by a database is added to GeoServer an internal connection pool is created. This connection pool is configurable.

8.9.1 Connection pool options

max connections	The maximum number of connections the pool can hold. When the maximum number of connections is exceeded, additional requests that require a database connection will be halted until a connection from the pool becomes available. The maximum number of connections limits the number of concurrent requests that can be made against the database.
min connections	The minimum number of connections the pool will hold. This number of connections is held even when there are no active requests. When this number of connections is exceeded due to serving requests additional connections are opened until the pool reaches its maximum size (described above).
validate connections	Flag indicating whether connections from the pool should be validated before they are used. A connection in the pool can become invalid for a number of reasons including network breakdown, database server timeout, etc.. The benefit of setting this flag is that an invalid connection will never be used which can prevent client errors. The downside of setting the flag is that a performance penalty is paid in order to validate connections.
fetch size	The number of records read from the database in each network exchange. If set too low (<50) network latency will affect negatively performance, if set too high it might consume a significant portion of GeoServer memory and push it towards an `Out Of Memory` error. Defaults to 1000.
connection timeout	Time, in seconds, the connection pool will wait before giving up its attempt to get a new connection from the database. Defaults to 20 seconds.

8.10 JNDI

Many data stores and connections in GeoServer have the option of utilizing Java Naming and Directory Interface on JNDI. JNDI allows for components in a Java system to look up other objects and data by a predefined name.

A common use of JNDI is to store a JDBC data source globally in a container. This has a few benefits. First, it can lead to a much more efficient use of database resources. Database connections in Java are very resource-intensive objects, so usually they are pooled. If each component that requires a database connection is responsible for creating their own connection pool, resources will stack up fast. In addition, often those resources are under-utilized and a component may not size its connection pool accordingly. A more efficient method is to set up a global pool at the servlet container level, and have every component that requires a database connection use that.

Furthermore, JNDI consolidates database connection configuration, as not every component that requires a database connection needs to know any more details than the JNDI name. This is very useful for administrators who may have to change database parameters in a running system, as it allows the change to occur in a single place.

8.11 SQL Views

The traditional way to access database data is is to configure layers against either tables or database views. Starting with GeoServer 2.1.0, layers can also be defined as SQL Views. SQL Views allow executing a custom SQL query on each request to the layer. This avoids the need to create a database view for complex queries.

Even more usefully, SQL View queries can be parameterized via string substitution. Parameter values can be supplied in both WMS and WFS requests. Default values can be supplied for parameters, and input values can be validated by Regular Expressions to eliminate the risk of SQL injection attacks.

SQL Views are read-only, and thus cannot be updated by WFS-T transactions.

8.11.1 Creating a SQL View

In order to create a SQL View the administrator invokes the *Create new layer* page. When a database store is selected, the usual list of tables and views available for publication appears, A link *Configure new SQL view...* also appears:

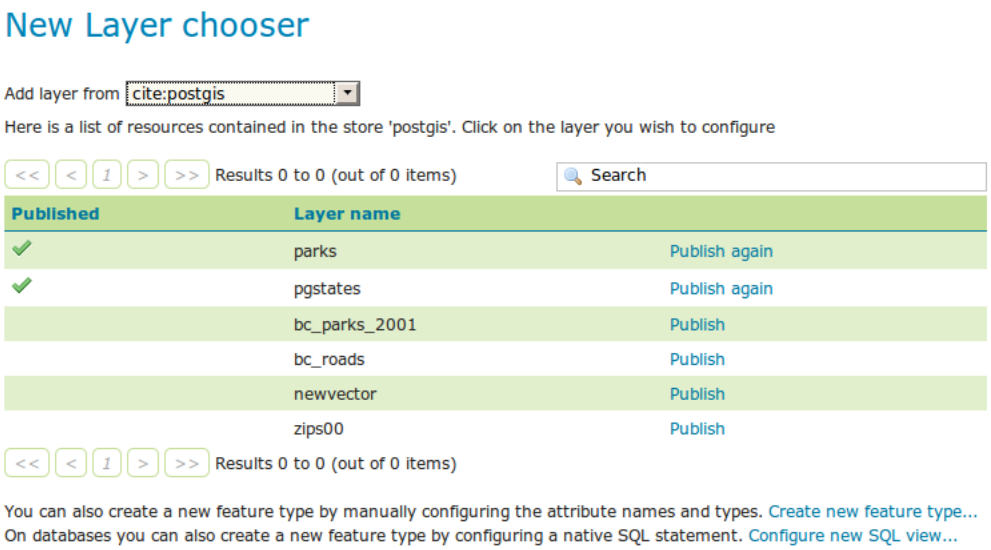
Selecting the *Configure new SQL view...* link opens a new page where the SQL view query can be specified:

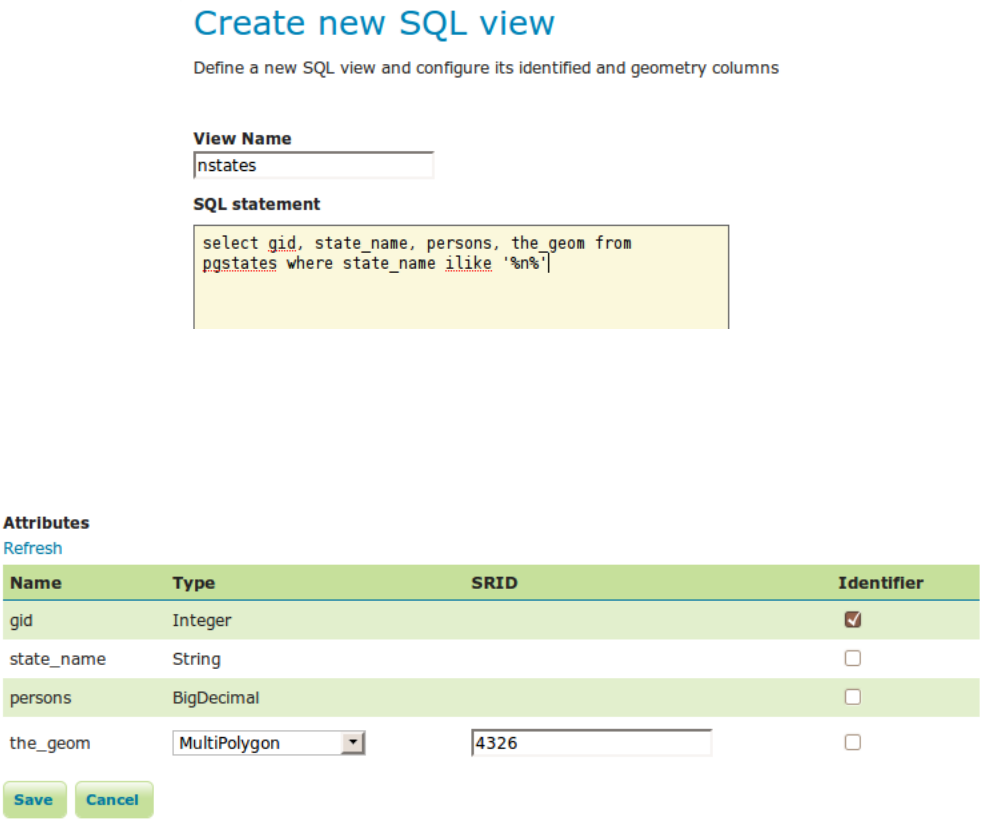
Note: The query can be any SQL statement that is valid as a subquery in a FROM clause (that is, `select * from (<the sql view>) [as] vtable`). This is the case for most SQL statements, but in some databases special syntax may be needed to call stored procedures. Also, all the columns returned by the SQL statement must have names. In some databases alias names are required for function calls.

When a valid SQL query has been entered, press the *Refresh* link in the **Attributes** table to get the list of the attribute columns determined from the query:

GeoServer attempts to determine the geometry column type and the native SRID, but these should be verified and corrected if necessary.

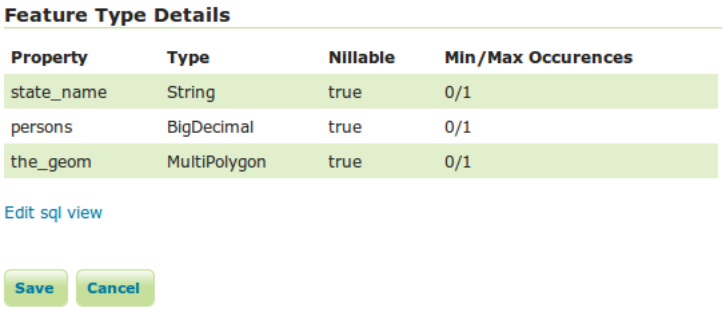
Note: Having a correct SRID (spatial reference id) is essential for spatial queries to work. In many spatial databases the SRID is equal to the EPSG code for the specific spatial reference system, but this is not always the case (for instance, Oracle has a number of non-EPSG SRID codes).


New Layer chooser
Add layer from cite:postgis
Here is a list of resources contained in the store 'postgis'. Click on the layer you wish to configure
<< < 1 > >> Results 0 to 0 (out of 0 items)
Search
Published
Layer name
parks
Publish again
pgstates
Publish again
bc_parks_2001
Publish
bc_roads
Publish
newvector
Publish
zips00
Publish
<< < 1 > >> Results 0 to 0 (out of 0 items)
You can also create a new feature type by manually configuring the attribute names and types. Create new feature type...
On databases you can also create a new feature type by configuring a native SQL statement. Configure new SQL view...


Create new SQL view
Define a new SQL view and configure its identified and geometry columns
View Name
nstates
SQL statement
select gid, state_name, persons, the_geom from
pgstates where state_name ilike '%n%'
Attributes
Refresh
Name
Type
SRID
Identifier
gid
Integer
state_name
String
persons
BigDecimal
the_geom
MultiPolygon
4326
Save
Cancel

If stable feature ids are desired for the view's features, one or more columns providing a unique id for the features should be checked in the **Identifier** column. Always ensure these attributes generate a unique key, or filtering and WFS requests will not work correctly.

Once the query and the attribute details are defined, press *Save*. The usual *New Layer* configuration page will appear. If further changes to the view are required, the page has a link to the SQL View editor at the bottom of the *Data* tab:



Once created, the SQL view layer is used in the same way as a conventional table-backed layer, with the one limitation of being read-only.

8.11.2 Parameterizing SQL Views

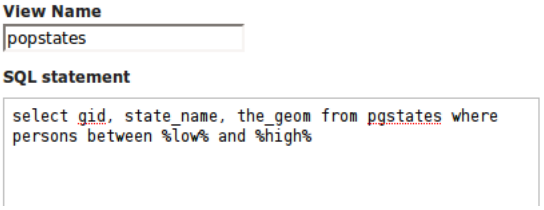
A parametric SQL view is based on a SQL query containing named parameters. The values for the parameters can be provided dynamically in WMS and WFS requests using the `viewparams` request parameter. Parameters can have default values specified, to handle the situation where they are not supplied in a request. Validation of supplied parameter values is supported by specifying validation regular expressions. Parameter values are only accepted if they match the regular expression defined for them. Appropriate parameter validation should always be used to avoid the risk of SQL injection attacks.

> **Warning:** SQL View parameter substitution should be used with caution, since improperly validated parameters open the risk of SQL injection attack. Where possible, consider using safer methods such as *dynamic filtering* in the request, or *Variable substitution in SLD*.

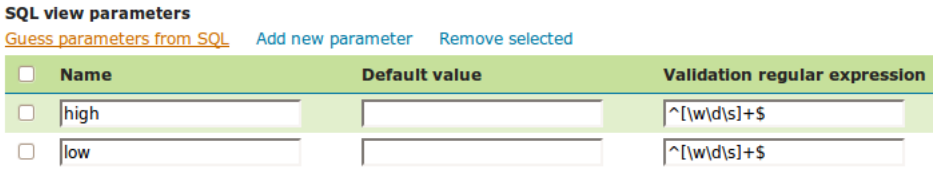
Defining parameters

Within the SQL View query, parameter names are delimited by leading and trailing `%` signs. The parameters can occur anywhere within the query text, including such uses as within SQL string constants, in place of SQL keywords, or representing entire SQL clauses.

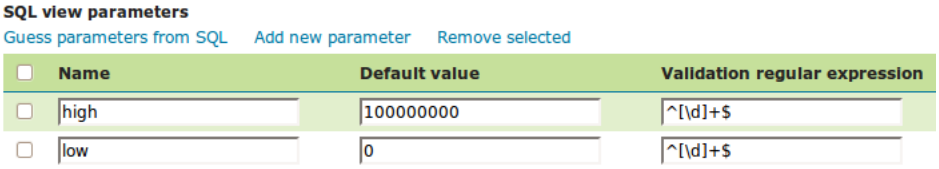
Here is an example of a SQL View query for a layer called `popstates` with two parameters, `low` and `high`:



Each parameter needs to be defined with its name, an optional default value, and a validation expression. The *Guess parameters from SQL* link can be clicked to infer the query parameters automatically, or they can be entered manually. The result is a table filled with the parameter names, default values and validation expressions:



In this case the default values should be specified, since the query cannot be executed without values for the parameters (because the expanded query `select gid, state_name, the_geom from pgstates where persons between and` is invalid SQL). Since the use of the parameters in the SQL query requires their values to be positive integer numbers, the validation regular expressions are specified to allow only numeric input (i.e. `^[\d]+$`):



Once the parameters have been defined, the **Attributes** *Refresh* link is clicked to parse the query and retrieve the attribute columns. The computed geometry type and column identifier details can be corrected if required. From this point on the workflow is the same as for a non-parameterized query.

Using a parametric SQL View

The SQL view parameters are specified by adding the `viewparams` parameter to the WMS `GetMap` or the WFS `GetFeature` request. The `viewparams` argument is a list of `key:value` pairs, separated by semicolons:

```
viewparams=p1:v1;p2:v2;...
```

If the values contain semicolons or commas these must be escaped with a backslash (e.g. `\,` and `\;`).

For example, the `popstates` SQL View layer can be displayed by invoking the *Layer Preview*. Initially no parameter values are supplied, so the defaults are used and all the states are displayed,

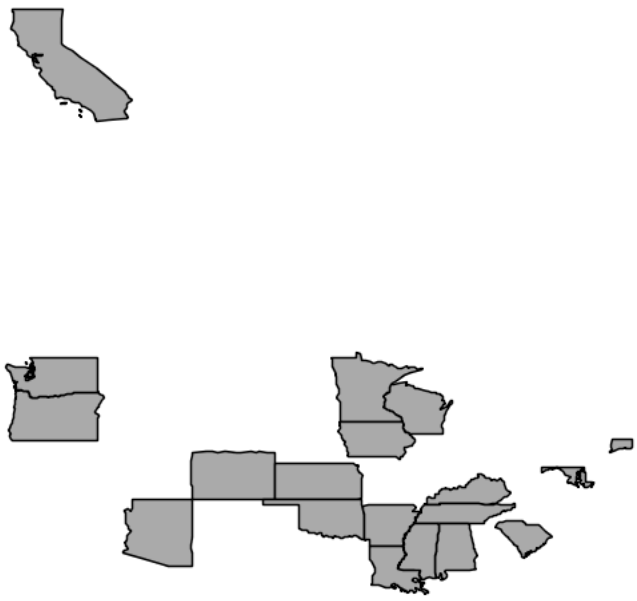
To display all states having more than 20 million inhabitants the following parameter is added to the `GetMap` request: `&viewparams=low:20000000`

To display all states having between 2 and 5 millions inhabitants the view parameters are: `&viewparams=low:2000000;high:5000000`

Parameters can be provided for multiple layers by separating each parameter map with a comma:

```
&viewparams=l1p1:v1;l1p2:v2,l2p1:v1;l2p2:v2,...
```

The number of parameter maps must match the number of layers (featuretypes) included in the request.



Parameters and validation

The value of a SQL View parameter can be an arbitrary string of text. The only constraint is that the attribute names and types returned by the view query must never change. This makes it possible to create views containing parameters representing complex SQL fragments. For example, using the view query `select * from pgstates %where%` allows specifying the WHERE clause of the query dynamically. However, this would likely require an empty validation expression. which presents a serious risk of SQL injection attacks. This technique should only be used if access to the server is restricted to trusted clients.

In general, SQL parameters must be used with care. They should always include validation regular expressions that accept only the intended parameter values. Note that while validation expressions should be constructed to prevent illegal values, they do not necessarily have to ensure the values are syntactically correct, since this will be checked by the database SQL parser. For example:

- `^[\d\.\+-eE]+$` checks that a parameter value contains valid characters for floating-point numbers (including scientific notation), but does not check that the value is actually a valid number
- `[^;']+` checks that a parameter value does not contain quotes or semicolons. This prevents common SQL injection attacks, but otherwise does not impose much limitation on the actual value

Resources for Validation Regular expressions

Defining effective validation regular expressions is important for security. Regular expressions are a complex topic that cannot be fully addressed here. The following are some resources for constructing regular expressions:

- GeoServer uses the standard Java regular expression engine. The Pattern class Javadocs contain the

full specification of the allowed syntax.

- http://www.regular-expressions.info has many tutorials and examples of regular expressions.
- The myregexp applet can be used to test regular expressions online.

8.12 Controlling feature ID generation in spatial databases

8.12.1 Introduction

All spatial database data stores (PostGIS, Oracle, MySQL and so on) normally derive the feature ID the table primary key and assume certain conventions on how to locate the next value for the key in case a new feature needs to be generated (WFS insert operation).

Common conventions rely on finding auto-increment columns (PostGIS) or finding a sequence that is named after a specific convention such as `<table>_<column>_SEQUENCE` (Oracle case).

In case none of the above is found normally the store will fall back on generating random feature IDs at each new request, making the table unsuitable for feature ID based searches and transactions.

8.12.2 Metadata table description

These defaults can be overridden manually by creating a *primary key metadata table* that specifies which columns to use and what strategy to use to generate new primary key values. The table can be created with this SQL statement (this one is valid for PostGIS and ORACLE, adapt it to your specific database, you may remove the check at the end if you want to):

```
--PostGIS DDL

CREATE TABLE gt_pk_metadata_table (
  table_schema VARCHAR(32) NOT NULL,
  table_name VARCHAR(32) NOT NULL,
  pk_column VARCHAR(32) NOT NULL,
  pk_column_idx INTEGER,
  pk_policy VARCHAR(32),
  pk_sequence VARCHAR(64),
  unique (table_schema, table_name, pk_column),
  check (pk_policy in ('sequence', 'assigned', 'autoincrement'))
)


--ORACLE DDL

CREATE TABLE gt_pk_metadata_table (
  table_schema VARCHAR2(32) NOT NULL,
  table_name VARCHAR2(32) NOT NULL,
  pk_column VARCHAR2(32) NOT NULL,
  pk_column_idx NUMBER(38),
  pk_policy VARCHAR2(32),
  pk_sequence VARCHAR2(64),
  constraint  chk_pk_policy check (pk_policy in ('sequence', 'assigned', 'autoincrement')));

CREATE UNIQUE INDEX gt_pk_metadata_table_idx01 ON gt_pk_metadata_table (table_schema, table_name, pk_
```

The table can be given a different name, in that case the name (eventually schema qualified) of the table must be specified in the *primary key metadata table* configuration parameter of the store.

The following table describes the meaning of each column in the metadata table.

Column	Description
table_schema	Name of the database schema in which the table is located.
table_name	Name of the table to be published
pk_column	Name of a column used to form the feature IDs
pk_column_idx	Index of the column in a multi-column key. In case multi column keys are needed multiple records with the same table schema and table name will be used.
pk_policy	The new value generation policy, used in case a new feature needs to be added in the table (following a WFS-T insert operation).
pk_sequence	The name of the database sequence to be used when generating a new value for the pk_column.

The possible values are:

- *assigned*. The value of the attribute in the newly inserted feature will be used (this assumes the "expose primary keys" flag has been enabled)
- *sequence*. The value of the attribute will be generated from the next value of a sequence indicated in the "pk_sequence" column.
- *autogenerated*. The column is an auto-increment one, the next value in the auto-increment will be used.

8.12.3 Using the metadata table with views

GeoServer can publish spatial views without issues, but normally results in two side effects:

- the view is treated as read only
- the feauture IDs are randomly generated

The metadata table can also refer to views, just use the view name in the `table_name` column: this will result in stable ids, and in databases supporting updatable views, it will also make the code treat the view as writable (thus, enabling usage of WFS-T on it).

8.13 Custom SQL session start/stop scripts

Starting with version 2.1.4 GeoServer support custom SQL scripts that can be run every time GeoServer grabs a connection from the connection pool, and every time the sesion is returned to the pool.

These scripts can be parametrized with the expansion of environment variables, which can be in turn set into the OGC request parameters with the same mechanism as *Variable substitution in SLD*.

In addition to the parameters provided via the request the `GSUSER` variable is guaranteed to contain the current GeoServer user, or be null if no authentication is available. This is useful if the SQL sessions scripts are used to provide tight control over database access

The SQL script can expand environment variables using the `${variableName, defaultValue}` syntax, for example the following alters the current database user to be the same as the GeoServer current user, or `geoserver` in case no user was authenticated

```
SET SESSION AUTHORIZATION ${GSUSER,geoserver}
```

8.14 Using SQL session scripts to control authorizations at the database level

GeoServer connects to a database via a connection pool, using the same rights as the user that is specified in the connection pool setup. In a setup that provides a variety of services and tables the connection pool user must have a rather large set of rights, such as table selection (WMS), table insert/update/delete (WFS-T) and even table creation (data upload via RESTConfig, WPS Import process and eventual new processes leveraging direct database connections).

What a user can do can be controlled by means of the GeoServer security subsystem, but in high security setups this might not be considered enough, and a database level access control be preferred instead. In these setups normally the connection pool user has limited access, such as simple read only access, while specific users are allowed to perform more operations.

When setting up such a solution remember the following guidelines:

- The connection pool user must be able to access all table metadata regardless of whether it is able to actually perform a select on the tables (dictionary tables/describe functionality must be always accessible)
- The connection pool must see each and every column of tables and views, in other words, the structure of the tables must not change as the current user changes
- the database users and the GeoServer user must be kept in synch with some external tools, GeoServer provides no out of the box facilities
- during the GeoServer startup the code will access the database to perform some sanity checks, in that moment there is no user authenticated in GeoServer so the code will run under whatever user was specified as the "default value" for the `GSUSER` variable.
- The user that administers GeoServer (normally `admin`, but it can be renamed, and other users given the administration roles too) must also be a database user, all administrative access on the GeoServer GUI will have that specific user controlling the session

Typical use cases:

- Give insert/update/delete rights only to users that must use WFS-T
- Only allow the administrator to create new tables
- Limit what rows of a table a user can see by using dynamic SQL views taking into account the current user to decide what rows to return

To make a point in case, if we want the PostgreSQL session to run with the current GeoServer user credentials the following scripts will be used:

The first command makes the database session use either the current GeoServer user, or the `geoserver` user if no authentication was available (anonymous user, or startup situation). The second command resets the session to the rights of the connection pool user.

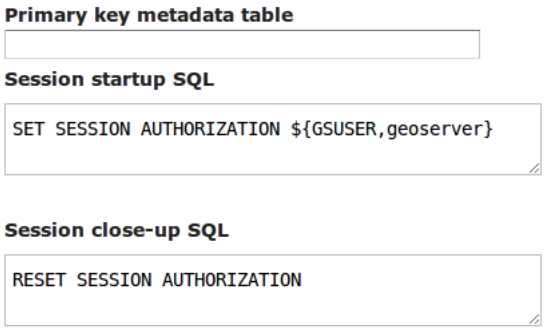


Figure 8.21: *Setting up session authorization for PostgreSQL*

CHAPTER 9

Working with Application Schemas

The application schema support (app-schema) extension provides support for *Complex Features* in GeoServer WFS.

Note: You must install the app-schema plugin to use Application Schema Support.

GeoServer provides support for a broad selection of simple feature data stores, including property files, shapefiles, and JDBC data stores such as PostGIS and Oracle Spatial. The app-schema module takes one or more of these simple feature data stores and applies a mapping to convert the simple feature types into one or more complex feature types conforming to a GML application schema.

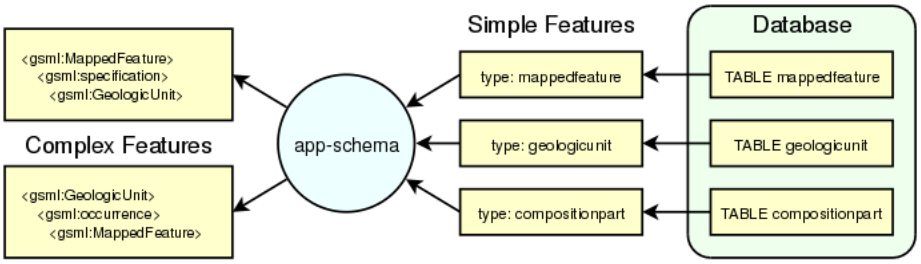


Figure 9.1: *Three tables in a database are accessed using GeoServer simple feature support and converted into two complex feature types.*

The app-schema module looks to GeoServer just like any other data store and so can be loaded and used to service WFS requests. In effect, the app-schema data store is a wrapper or adapter that converts a simple feature data store into complex features for delivery via WFS. The mapping works both ways, so queries against properties of complex features are supported.

9.1 Complex Features

To understand complex features, and why you would want use them, you first need to know a little about simple features.

9.1.1 Simple features

A common use of GeoServer WFS is to connect to a data source such as a database and access one or more tables, where each table is treated as a WFS simple feature type. Simple features contain a list of properties that each have one piece of simple information such as a string or number. (Special provision is made for geometry objects, which are treated like single items of simple data.) The Open Geospatial Consortium (OGC) defines three Simple Feature profiles; SF-0, SF-1, and SF-2. GeoServer simple features are close to OGC SF-0, the simplest OGC profile.

GeoServer WFS simple features provide a straightforward mapping from a database table or similar structure to a "flat" XML representation, where every column of the table maps to an XML element that usually contains no further structure. One reason why GeoServer WFS is so easy to use with simple features is that the conversion from columns in a database table to XML elements is automatic. The name of each element is the name of the column, in the namespace of the data store. The name of the feature type defaults to the name of the table. GeoServer WFS can manufacture an XSD type definition for every simple feature type it serves. Submit a DescribeFeatureType request to see it.

Benefits of simple features

- Easy to implement
- Fast
- Support queries on properies, including spatial queries on geometries

Drawbacks of simple features

- When GeoServer automatically generates an XSD, the XML format is tied to the database schema.
- To share data with GeoServer simple features, participants must either use the same database schema or translate between different schemas.
- Even if a community could agree on a single database schema, as more data owners with different data are added to a community, the number of columns in the table becomes unmanageable.
- Interoperability is difficult because simple features do not allow modification of only part of the schema.

Simple feature example

For example, if we had a database table `stations` containing information about GPS stations:

```
| id | code |      name      |         location         |
+----+------+----------------+--------------------------+
| 27 | ALIC | Alice Springs  | POINT(133.8855 -23.6701) |
| 4  | NORF | Norfolk Island | POINT(167.9388 -29.0434) |
| 12 | COCO | Cocos          | POINT(96.8339 -12.1883)  |
| 31 | ALBY | Albany         | POINT(117.8102 -34.9502) |
```

GeoServer would then be able to create the following simple feature WFS response fragment:

```
<gps:stations gml:id="stations.27">
    <gps:code>ALIC</gps:code>
    <gps:name>Alice Springs</gps:name>
    <gps:location>
        <gml:Point srsName="urn:x-ogc:def:crs:EPSG:4326">
```

```
            <gml:pos>-23.6701 133.8855</gml:pos>
        </gml:Point>
    </gps:location>
</gps:stations>
```

- Every row in the table is converted into a feature.
- Every column in the table is converted into an element, which contains the value for that row.
- Every element is in the namespace of the data store.
- Automatic conversions are applied to some special types like geometries, which have internal structure, and include elements defined in GML.

9.1.2 Complex features

Complex features contain properties that can contain further nested properties to arbitrary depth. In particular, complex features can contain properties that are other complex features. Complex features can be used to represent information not as an XML view of a single table, but as a collection of related objects of different types.

Simple feature	Complex feature
Properties are single data item, e.g. text, number, geometry	Properties can be complex, including complex features
XML view of a single table	Collection of related identifiable objects
Schema automatically generated based on database	Schema agreed by community
One large type	Multiple different types
Straightforward	Richly featured data standards
Interoperability relies on simplicity and customisation	Interoperability through standardisation

Benefits of complex features

- Can define information model as an object-oriented structure, an *application schema*.
- Information is modelled not as a single table but as a collection of related objects whose associations and types may vary from feature to feature (polymorphism), permitting rich expression of content.
- By breaking the schema into a collection of independent types, communities need only extend those types they need to modify. This simplifies governance and permits interoperability between related communities who can agree on common base types but need not agree on application-specific subtypes..

Drawbacks of complex features

- More complex to implement
- Complex responses might slower if more database queries are required for each feature.
- Information modelling is required to standardise an application schema. While this is beneficial, it requires effort from the user community.

Complex feature example

Let us return to our `stations` table and supplement it with a foreign key `gu_id` that describes the relationship between the GPS station and the geologic unit to which it is physically attached:

```
| id | code |      name      |         location          | gu_id |
+----+------+----------------+---------------------------+-------+
| 27 | ALIC | Alice Springs  | POINT(133.8855 -23.6701) | 32785 |
| 4  | NORF | Norfolk Island | POINT(167.9388 -29.0434) | 10237 |
| 12 | COCO | Cocos          | POINT(96.8339 -12.1883)  | 19286 |
| 31 | ALBY | Albany         | POINT(117.8102 -34.9502) | 92774 |
```

The geologic unit is is stored in the table `geologicunit`:

```
| gu_id |                       urn             |         text        |
+-------+---------------------------------------+---------------------+
| 32785 | urn:x-demo:feature:GeologicUnit:32785 | Metamorphic bedrock |
...
```

The simple features approach would be to join the `stations` table with the `geologicunit` table into one view and then deliver "flat" XML that contained all the properties of both. The complex feature approach is to deliver the two tables as separate feature types. This allows the relationship between the entities to be represented while preserving their individual identity.

For example, we could map the GPS station to a `sa:SamplingPoint` with a `gsml:GeologicUnit`. The these types are defined in the following application schemas respectively:

- http://schemas.opengis.net/sampling/1.0.0/sampling.xsd
  - Documentation: OGC 07-002r3: http://portal.opengeospatial.org/files/?artifact_id=22467
- http://www.geosciml.org/geosciml/2.0/xsd/geosciml.xsd
  - Documentation: http://www.geosciml.org/geosciml/2.0/doc/

The complex feature WFS response fragment could then be encoded as:

```
<sa:SamplingPoint gml:id="stations.27>
  <gml:name codeSpace="urn:x-demo:SimpleName">Alice Springs</gml:name>
  <gml:name codeSpace="urn:x-demo:IGS:ID">ALIC</gml:name>
  <sa:sampledFeature>
     <gsml:GeologicUnit gml:id="geologicunit.32785">
         <gml:description>Metamorphic bedrock</gml:description>
         <gml:name codeSpace="urn:x-demo:Feature">urn:x-demo:feature:GeologicUnit:32785</gml:name>
     </gsml:GeologicUnit>
  </sa:sampledFeature>
  <sa:relatedObservation xlink:href="urn:x-demo:feature:GeologicUnit:32785" />
  <sa:position>
      <gml:Point srsName="urn:x-ogc:def:crs:EPSG:4326">
          <gml:pos>-23.6701 133.8855</gml:pos>
      </gml:Point>
  </sa:position>
</sa:SamplingPoint>
```

- The property `sa:sampledFeature` can reference any other feature type, inline (included in the response) or by reference (an `xlink:href` URL or URN). This is an example of the use of polymorphism.
- The property `sa:relatedObservation` refers to the same GeologicUnit as `sa:sampledFeature`, but by reference.

- Derivation of new types provides an extension point, allowing information models to be reused and extended in a way that supports backwards compatibility.
- Multiple sampling points can share a single GeologicUnit. Application schemas can also define multivalued properties to support many-to-one or many-to-many associations.
- Each GeologicUnit could have further properties describing in detail the properties of the rock, such as colour, weathering, lithology, or relevant geologic events.
- The GeologicUnit feature type can be served separately, and could be uniquely identified through its properties as the same instance seen in the SamplingPoint.

Portrayal complex features (SF0)

Portrayal schemas are standardised schemas with flat attributes, also known as simple feature level 0 (SF0). Because a community schema is still required (e.g. GeoSciML-Portrayal), app-schema plugin is still used to map the database columns to the attributes.

- *WFS CSV output format* is supported for complex features with portrayal schemas. At the moment, propertyName selection is not yet supported with csv outputFormat, so it always returns the full set of attributes.
- Complex features with nesting and multi-valued properties are not supported with *WFS CSV output format*.

9.2 Installation

Application schema support is a GeoServer extension and is downloaded separately.

- Download the app-schema plugin zip file for the same version of GeoServer.
- Unzip the app-schema plugin zip file to obtain the jar files inside. Do not unzip the jar files.
- Place the jar files in the `WEB-INF/lib` directory of your GeoServer installation.
- Restart GeoServer to load the extension (although you might want to configure it first [see below]).

9.3 WFS Service Settings

There are two GeoServer WFS service settings that are strongly recommended for interoperable complex feature services. These can be enabled through the *Services* → *WFS* page on the GeoServer web interface or by manually editing the `wfs.xml` file in the data directory,

9.3.1 Canonical schema location

The default GeoServer behaviour is to encode WFS responses that include a `schemaLocation` for the WFS schema that is located on the GeoServer instance. A client will not know without retrieving the schema whether it is identical to the official schema hosted at `schemas.opengis.net`. The solution is to encode the `schemaLocation` for the WFS schema as the canonical location at `schemas.opengis.net`.

To enable this option, choose *one* of these:

1. Either: On the *Service* → *WFS* page under *Conformance* check *Encode canonical WFS schema location*.
2. Or: Insert the following line before the closing tag in `wfs.xml`:

```
<canonicalSchemaLocation>true</canonicalSchemaLocation>
```

9.3.2 Encode using featureMember

By default GeoServer will encode WFS 1.1 responses with multiple features in a single `gml:featureMembers` element. This will cause invalid output if a response includes a feature at the top level that has already been encoded as a nested property of an earlier feature, because there is no single element that can be used to encode this feature by reference. The solution is to encode responses using `gml:featureMember`.

To enable this option, choose *one* of these:

1. Either: On the *Service* → *WFS* page under *Encode response with* select *Multiple "featureMember" elements*.
2. Or: Insert the following line before the closing tag in `wfs.xml`:

   ```
   <encodeFeatureMember>true</encodeFeatureMember>
   ```

9.4 Configuration

Configuration of an app-schema complex feature type requires manual construction of a GeoServer data directory that contains an XML mapping file and a `datastore.xml` that points at this mapping file. The data directory also requires all the other ancillary configuration files used by GeoServer for simple features. GeoServer can serve simple and complex features at the same time.

9.4.1 Workspace layout

The GeoServer data directory contains a folder called `workspaces` with the following structure:

```
workspaces
    - gsml
        - SomeDataStore
            - SomeFeatureType
                - featuretype.xml
            - datastore.xml
            - SomeFeatureType-mapping-file.xml
```

Note: The folder inside `workspaces` must have a name (the workspace name) that is the same as the namespace prefix (`gsml` in this example).

9.4.2 Datastore

Each data store folder contains a file `datastore.xml` that contains the configuration parameters of the data store. To create an app-schema feature type, the data store must be configured to load the app-schema service module and process the mapping file. These options are contained in the `connectionParameters`:

- `namespace` defines the XML namespace of the complex feature type.
- `url` is a `file:` URL that gives the location of the app-schema mapping file relative to the root of the GeoServer data directory.

- `dbtype` must be `app-schema` to trigger the creation of an app-schema feature type.

9.5 Mapping File

An app-schema feature type is configured using a mapping file that defines the data source for the feature and the mappings from the source data to XPaths in the output XML.

9.5.1 Outline

Here is an outline of a mapping file:

```
<?xml version="1.0" encoding="UTF-8"?>
<as:AppSchemaDataAccess xmlns:as="http://www.geotools.org/app-schema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.geotools.org/app-schema AppSchemaDataAccess.xsd">
    <namespaces>...</namespaces>
    <includedTypes>...</includedTypes>
    <sourceDataStores>...</sourceDataStores>
    <catalog>...</catalog>
    <targetTypes...</targetTypes>
    <typeMappings>...</typeMappings>
</as:AppSchemaDataAccess>
```

- `namespaces` defines all the namespace prefixes used in the mapping file.
- `includedTypes` (optional) defines all the included non-feature type mapping file locations that are referred in the mapping file.
- `sourceDataStores` provides the configuration information for the source data stores.
- `catalog` is the location of the OASIS Catalog used to resolve XML Schema locations.
- `targetTypes` is the location of the XML Schema that defines the feature type.
- `typeMappings` give the relationships between the fields of the source data store and the elements of the output complex feature.

Mapping file schema

- `AppSchemaDataAccess.xsd` is optional because it is not used by GeoServer. The presence of `AppSchemaDataAccess.xsd` in the same folder as the mapping file enables XML editors to observe its grammar and provide contextual help.

9.5.2 Settings

namespaces

The `namespaces` section defines all the XML namespaces used in the mapping file:

```
<Namespace>
    <prefix>gsml</prefix>
    <uri>urn:cgi:xmlns:CGI:GeoSciML:2.0</uri>
</Namespace>
<Namespace>
```

```
    <prefix>gml</prefix>
    <uri>http://www.opengis.net/gml</uri>
</Namespace>
<Namespace>
    <prefix>xlink</prefix>
    <uri>http://www.w3.org/1999/xlink</uri>
</Namespace>
```

includedTypes (optional)

Non-feature types (eg. gsml:CompositionPart is a data type that is nested in gsml:GeologicUnit) may be mapped separately for its reusability, but we don't want to configure it as a feature type as we don't want to individually access it. Related feature types don't need to be explicitly included here as it would have its own workspace configuration for GeoServer to find it. The location path in `Include` tag is relative to the mapping file. For an example, if gsml:CompositionPart configuration file is located in the same directory as the gsml:GeologicUnit configuration:

```
<includedTypes>
    <Include>gsml_CompositionPart.xml</Include>
</includedTypes>
```

sourceDataStores

Every mapping file requires at least one data store to provide data for features. app-schema reuses GeoServer data stores, so there are many available types. See *Data Stores* for details of data store configuration. For example:

```
<sourceDataStores>
    <DataStore>
        <id>datastore</id>
        <parameters>
            ...
        </parameters>
    </DataStore>
    ...
</sourceDataStores>
```

If you have more than one `DataStore` in a mapping file, be sure to give them each a distinct `id`.

catalog (optional)

The location of an OASIS XML Catalog configuration file, given as a path relative to the mapping file. See *Application Schema Resolution* for more information. For example:

```
<catalog>../../../schemas/catalog.xml</catalog>
```

targetTypes

The `targetTypes` section lists all the application schemas required to define the mapping. Typically only one is required. For example:

```
<targetTypes>
    <FeatureType>
        <schemaUri>http://www.geosciml.org/geosciml/2.0/xsd/geosciml.xsd</schemaUri>
    </FeatureType>
</targetTypes>
```

9.5.3 Mappings

typeMappings and FeatureTypeMapping

The `typeMappings` section is the heart of the app-schema module. It defines the mapping from simple features to the the nested structure of one or more simple features. It consists of a list of `FeatureTypeMapping` elements, which each define one output feature type. For example:

```
<typeMappings>
    <FeatureTypeMapping>
        <mappingName>mappedfeature1</mappingName>
        <sourceDataStore>datastore</sourceDataStore>
        <sourceType>mappedfeature</sourceType>
        <targetElement>gsml:MappedFeature</targetElement>
        <isDenormalised>true</isDenormalised>
        <attributeMappings>
            <AttributeMapping>
                ...
```

- `mappingName` is an optional tag, to identify the mapping in *Feature Chaining* when there are multiple FeatureTypeMapping instances for the same type. This is solely for feature chaining purposes, and would not work for identifying top level features.
- `sourceDataStore` must be an identifier you provided when you defined a source data store the `sourceDataStores` section.
- `sourceType` is the simple feature type name. For example:
  - a table or view name, lowercase for PostGIS, uppercase for Oracle.
  - a property file name (without the .properties suffix)
- `targetElement` is the the element name in the target application schema. This is the same as the WFS feature type name.
- `isDenormalised` is an optional tag (default true) to indicate whether this type contains denormalised data or not. If data is not denormalised, then the global feature limit can be safely applied when querying the database. When combined with a low global feature limit (via *Services –> WFS*), setting this option to false can prevent unnecessary processing and database lookups from taking place.

attributeMappings and AttributeMapping

`attributeMappings` comprises a list of `AttributeMapping` elements:

```
<AttributeMapping>
    <targetAttribute>...</targetAttribute>
    <idExpression>...</idExpression>
    <sourceExpression>...</sourceExpression>
    <targetAttributeNode>...</targetAttributeNode>
    <isMultiple>...</isMultiple>
```

```
    <ClientProperty>...</ClientProperty>
</AttributeMapping>
```

targetAttribute

`targetAttribute` is the XPath to the output element, in the context of the target element. For example, if the containing mapping is for a feature, you should be able to map a `gml:name` property by setting the target attribute:

```
<targetAttribute>gml:name</targetAttribute>
```

Multivalued attributes resulting from *Denormalised sources* are automatically encoded. If you wish to encode multivalued attributes from different input columns as a specific instance of an attribute, you can use a (one-based) index. For example, you can set the third `gml:name` with:

```
<targetAttribute>gml:name[3]</targetAttribute>
```

The reserved name `FEATURE_LINK` is used to map data that is not encoded in XML but is required for use in *Feature Chaining*.

idExpression (optional)

A CQL expression that is used to set the custom `gml:id` of the output feature type. This should be the name of a database column on its own. Using functions would cause an exception because it is not supported with the default joining implementation.

Note: Every feature must have a `gml:id`. This requirement is an implementation limitation (strictly, `gml:id` is optional in GML).

- If idExpression is unspecified, `gml:id` will be `<the table name>.<primary key>`, e.g. `MAPPEDFEATURE.1`.
- In the absence of primary keys, this will be `<the table name>.<generated gml id>`, e.g. `MAPPEDFEATURE.fid--46fd41b8_1407138b56f_-7fe0`.
- If using property files instead of database tables, the default `gml:id` will be the row key found before the equals ("=") in the property file, e.g. the feature with row "mf1=Mudstone|POINT(1 2)|..." will have gml:id `mf1`.

Note: `gml:id` must be an NCName.

---

sourceExpression (optional)

Use a `sourceExpression` tag to set the element content from source data. For example, to set the element content from a column called `DESCRIPTION`:

```
<sourceExpression><OCQL>DESCRIPTION</OCQL></sourceExpression>
```

If `sourceExpression` is not present, the generated element is empty (unless set by another mapping).

You can use CQL expressions to calculate the content of the element. This example concatenated strings from two columns and a literal:

```
<sourceExpression>
    <OCQL>strConCat(FIRST , strConCat(' followed by ', SECOND))</OCQL>
</sourceExpression>
```

You can also use *CQL functions* for vocabulary translations.

Warning: Avoid use of CQL expressions for properties that users will want to query, because the current implementation cannot reverse these expressions to generate efficient SQL, and will instead read all features to calculate the property to find the features that match the filter query. Falling back to brute force search makes queries on CQL-calculated expressions very slow. If you must concatenate strings to generate content, you may find that doing this in your database is much faster.

linkElement and linkField (optional)

The presence of `linkElement` and `linkField` change the meaning of `sourceExpression` to a *Feature Chaining* mapping, in which the source of the mapping is the feature of type `linkElement` with property `linkField` matching the expression. For example, the following `sourceExpression` uses as the result of the mapping the (possibly multivalued) `gsml:MappedFeature` for which `gml:name[2]` is equal to the value of `URN` for the source feature. This is in effect a foreign key relation:

```
<sourceExpression>
    <OCQL>URN</OCQL>
    <linkElement>gsml:MappedFeature</linkElement>
    <linkField>gml:name[2]</linkField>
</sourceExpression>
```

The feature type `gsml:MappedFeature` might be defined in another mapping file. The `linkField` can be `FEATURE_LINK` if you wish to relate the features by a property not exposed in XML. See *Feature Chaining* for a comprehensive discussion.

For special cases, `linkElement` could be an OCQL function, and `linkField` could be omitted. See *Polymorphism* for further information.

targetAttributeNode (optional)

`targetAttributeNode` is required wherever a property type contains an abstract element and app-schema cannot determine the type of the enclosed attribute.

In this example, `om:result` is of `xs:anyType`, which is abstract. We can use `targetAttributeNode` to set the type of the property type to a type that encloses a non-abstract element:

```
<AttributeMapping>
      <targetAttribute>om:result</targetAttribute>
      <targetAttributeNode>gml:MeasureType<targetAttributeNode>
      <sourceExpression>
          <OCQL>TOPAGE</OCQL>
      </sourceExpression>
      <ClientProperty>
          <name>xsi:type</name>
          <value>'gml:MeasureType'</value>
      </ClientProperty>
      <ClientProperty>
          <name>uom</name>
          <value>'http://www.opengis.net/def/uom/UCUM/0/Ma'</value>
```

```
      </ClientProperty>
</AttributeMapping>
```

If the casting type is complex, the specific type is implicitly determined by the XPath in targetAttribute and targetAttributeNode is not required. E.g., in this example `om:result` is automatically specialised as a MappedFeatureType:

```
<AttributeMapping>
      <targetAttribute>om:result/gsml:MappedFeature/gml:name</targetAttribute>
      <sourceExpression>
          <OCQL>NAME</OCQL>
      </sourceExpression>
</AttributeMapping>
```

Although it is not required, we may still specify targetAttributeNode for the root node, and map the children attributes as per normal. This mapping must come before the mapping for the enclosed elements. By doing this, app-schema will report an exception if a mapping is specified for any of the children attributes that violates the type in targetAttributeNode. E.g.:

```
<AttributeMapping>
      <targetAttribute>om:result</targetAttribute>
      <targetAttributeNode>gsml:MappedFeatureType<targetAttributeNode>
</AttributeMapping>
<AttributeMapping>
      <targetAttribute>om:result/gsml:MappedFeature/gml:name</targetAttribute>
      <sourceExpression>
          <OCQL>NAME</OCQL>
      </sourceExpression>
</AttributeMapping>
```

Note that the GML encoding rules require that complex types are never the direct property of another complex type; they are always contained in a property type to ensure that their type is encoded in a surrounding element. Encoded GML is always type/property/type/property. This is also known as the GML "stripping" rule. The consequence of this for app-schema mapping files is that `targetAttributeNode` must be applied to the property and the type must be set to the XSD property type, not to the type of the contained attribute (`gsml:CGI_TermValuePropertyType` not `gsml:CGI_TermValueType`). Because the XPath refers to a property type not the encoded content, `targetAttributeNode` appears in a mapping with `targetAttribute` and no other elements when using with complex types.

encodeIfEmpty (optional)

The `encodeIfEmpty` element will determine if an attribute will be encoded if it contains a null or empty value. By default `encodeIfEmpty` is set to false therefore any attribute that does not contain a value will be skipped:

```
<encodeIfEmpty>true</encodeIfEmpty>
```

`encodeIfEmpty` can be used to bring up attributes that only contain client properties such as `xlink:title`.

isMultiple (optional)

The `isMultiple` element states whether there might be multiple values for this attribute, coming from denormalised rows. Because the default value is `false` and it is omitted in this case, it is most usually seen as:

```
<isMultiple>true</isMultiple>
```

For example, the table below is denormalised with `NAME` column having multiple values:

ID	NAME	DESCRIPTION
gu.25678	Yaugher Volcanic Group 1	Olivine basalt, tuff, microgabbro
gu.25678	Yaugher Volcanic Group 2	Olivine basalt, tuff, microgabbro

The configuration file specifies `isMultiple` for `gml:name` attribute that is mapped to the `NAME` column:

```
<AttributeMapping>
    <targetAttribute>gml:name</targetAttribute>
    <sourceExpression>
        <OCQL>NAME</OCQL>
    </sourceExpression>
    <isMultiple>true</isMultiple>
    <ClientProperty>
        <name>codeSpace</name>
        <value>'urn:ietf:rfc:2141'</value>
    </ClientProperty>
</AttributeMapping>
```

The output produces multiple `gml:name` attributes for each feature grouped by the id:

```
<gsml:GeologicUnit gml:id="gu.25678">
    <gml:description>Olivine basalt, tuff, microgabbro</gml:description>
    <gml:name codeSpace="urn:ietf:rfc:2141">Yaugher Volcanic Group 1</gml:name>
    <gml:name codeSpace="urn:ietf:rfc:2141">Yaugher Volcanic Group 2</gml:name>
 ...
```

isList (optional)

The `isList` element states whether there might be multiple values for this attribute, concatenated as a list. The usage is similar with `isMultiple`, except the values appear concatenated inside a single node instead of each value encoded in a separate node. Because the default value is `false` and it is omitted in this case, it is most usually seen as:

```
<isList>true</isList>
```

For example, the table below has multiple `POSITION` for each feature:

ID	POSITION
ID1.2	1948-05
ID1.2	1948-06
ID1.2	1948-07
ID1.2	1948-08
ID1.2	1948-09

The configuration file uses `isList` on `timePositionList` attribute mapped to `POSITION` column:

```
<AttributeMapping>
    <targetAttribute>csml:timePositionList</targetAttribute>
    <sourceExpression>
        <OCQL>POSITION</OCQL>
    </sourceExpression>
    <isList>true</isList>
</AttributeMapping>
```

The output produced:

```
<csml:pointSeriesDomain>
    <csml:TimeSeries gml:id="ID1.2">
        <csml:timePositionList>1949-05 1949-06 1949-07 1949-08 1949-09</csml:timePositionList>
    </csml:TimeSeries>
</csml:pointSeriesDomain>
```

ClientProperty (optional, multivalued)

A mapping can have one or more `ClientProperty` elements which set XML attributes on the mapping target. Each `ClientProperty` has a `name` and a `value` that is an arbitrary CQL expression. No `OCQL` element is used inside `value`.

This example of a `ClientProperty` element sets the `codeSpace` XML attribute to the literal string `urn:ietf:rfc:2141`. Note the use of single quotes around the literal string. This could be applied to any target attribute of GML CodeType:

```
<ClientProperty>
    <name>codeSpace</name>
    <value>'urn:ietf:rfc:2141'</value>
</ClientProperty>
```

When the GML association pattern is used to encode a property by reference, the `xlink:href` attribute is set and the element is empty. This `ClientProperty` element sets the `xlink:href` XML attribute to to the value of the `RELATED_FEATURE_URN` field in the data source (for example, a column in an Oracle database table). This mapping could be applied to any property type, such a `gml:FeaturePropertyType`, or other type modelled on the GML association pattern:

```
<ClientProperty>
    <name>xlink:href</name>
    <value>RELATED_FEATURE_URN</value>
</ClientProperty>
```

See the discussion in *Feature Chaining* for the special case in which `xlink:href` is created for multivalued properties by reference.

9.5.4 CQL

CQL functions enable data conversion and conditional behaviour to be specified in mapping files.

- See *CQL functions* for information on additional functions provided by the app-schema plugin.
- The uDig manual includes a list of CQL functions:
  - http://udig.refractions.net/confluence/display/EN/Constraint+Query+Language
- CQL string literals are enclosed in single quotes, for example `'urn:ogc:def:nil:OGC:missing'`.

9.5.5 Database identifiers

When referring to database table/view names or column names, use:

- lowercase for PostGIS
- UPPERCASE for Oracle Spatial and ArcSDE

9.5.6 Denormalised sources

Multivalued properties from denormalised sources (the same source feature ID appears more than once) are automatically encoded. For example, a view might have a repeated `id` column with varying `name` so that an arbitrarily large number of `gml:name` properties can be encoded for the output feature.

Warning: Denormalised sources must grouped so that features with duplicate IDs are provided without any intervening features. This can be achieved by ensuring that denormalised source features are sorted by ID. Failure to observe this restriction will result in data corruption. This restriction is however not necessary when using *Joining Support For Performance* because then ordering will happen automatically.

9.6 Application Schema Resolution

To be able to encode XML responses conforming to a GML application schema, the app-schema plugin must be able to locate the application schema files (XSDs) that define the schema. This page describes the schema resolution process.

9.6.1 Schema downloading is now automatic for most users

GeoServer will automatically download and cache (see Cache below) all the schemas it needs the first time it starts if:

1. All the application schemas you use are accessed via http/https URLs, and
2. Your GeoServer instance is deployed on a network that permits it to download them.

Note: This is the recommended way of using GeoServer app-schema for most users.

If cached downloading is used, no manual handling of schemas will be required. The rest of this page is for those with more complicated arrangements, or who wish to clear the cache.

9.6.2 Resolution order

The order of sources used to resolve application schemas is:

1. OASIS Catalog
2. Classpath
3. Cache

Every attempt to load a schema works down this list, so imports can be resolved from sources other than that used for the originating document. For example, an application schema in the cache that references a schema found in the catalog will use the version in the catalog, rather than caching it. This allows users to supply unpublished or modified schemas sourced from, for example, the catalog, at the cost of interoperability (how do WFS clients get them?).

9.6.3 OASIS Catalog

An OASIS XML Catalog is a standard configuration file format that instructs an XML processing system how to process entity references. The GeoServer app-schema resolver uses catalog URI semantics to locate

application schemas, so `uri` or `rewriteURI` entries should be present in your catalog. The optional mapping file `catalog` element provides the location of the OASIS XML Catalog configuration file, given as a path relative to the mapping file, for example:

```
<catalog>../../../schemas/catalog.xml</catalog>
```

Earlier versions of the app-schema plugin required all schemas to be present in the catalog. This is no longer the case. Because the catalog is searched first, existing catalog-based deployments will continue to work as before.

To migrate an existing GeoServer app-schema deployment that uses an OASIS Catalog to instead use cached downloads (see Cache below), remove all `catalog` elements from your mapping files and restart GeoServer.

9.6.4 Classpath

Java applications such as GeoServer can load resources from the Java classpath. GeoServer app-schema uses a simple mapping from an http or https URL to a classpath resource location. For example, an application schema published at `http://schemas.example.org/exampleml/exml.xsd` would be found on the classpath if it was stored either:

- at `/org/example/schemas/exampleml/exml.xsd` in a JAR file on the classpath (for example, a JAR file in `WEB-INF/lib`) or,
- on the local filesystem at `WEB-INF/classes/org/example/schemas/exampleml/exml.xsd`.

The ability to load schemas from the classpath is intended to support testing, but may be useful to users whose communities supply JAR files containing their application schemas.

9.6.5 Cache

If an application schema cannot be found in the catalog or on the classpath, it is downloaded from the network and stored in a subdirectory `app-schema-cache` of the GeoServer data directory.

- Once schemas are downloaded into the cache, they persist indefinitely, including over GeoServer restarts.
- No attempt will be made to retrieve new versions of cached schemas.
- To clear the cache, remove the subdirectory `app-schema-cache` of the GeoServer data directory and restart GeoServer.

GeoServer app-schema uses a simple mapping from an http or https URL to local filesystem path. For example, an application schema published at `http://schemas.example.org/exampleml/exml.xsd` would be downloaded and stored as `app-schema-cache/org/example/schemas/exampleml/exml.xsd`. Note that:

- Only `http` and `https` URLs are supported.
- Port numbers, queries, and fragments are ignored.

If your GeoServer instance is deployed on a network whose firewall rules prevent outgoing TCP connections on port 80 (http) or 443 (https), schema downloading will not work. (For security reasons, some service networks ["demilitarised zones"] prohibit such outgoing connections.) If schema downloading is not permitted on your network, there are three solutions:

1. Either: Install and configure GeoServer on another network that can make outgoing TCP connections, start GeoServer to trigger schema download, and then manually copy the `app-schema-cache` directory to the production server. This is the easiest option because GeoServer automatically downloads all the schemas it needs, including dependencies.
2. Or: Deploy JAR files containing all required schema files on the classpath (see Classpath above).
3. Or: Use a catalog (see OASIS Catalog above).

9.7 Supported GML Versions

9.7.1 GML 3.1.1

- GML 3.1.1 application schemas are supported for WFS 1.1.0.
- Clients must specify WFS 1.1.0 in requests because the GeoServer default is WFS 2.0.0.
- GET URLs must contain `version=1.1.0` to set the WFS version to 1.1.0.

9.7.2 GML 3.2.1

- GML 3.2.1 application schemas are supported for WFS 1.1.0 and (incomplete) WFS 2.0.0.
- WFS 2.0.0 features not in WFS 1.1.0 such as paging (STARTINDEX) are not yet supported.
- Clients using WFS 1.1.0 must specify WFS 1.1.0 in requests and select the `gml32` output format for GML 3.2.1.
- To use WFS 1.1.0 for GML 3.2.1, GET URLs must contain `version=1.1.0` to set the WFS version to 1.1.0 and `outputFormat=gml32` to set the output format to GML 3.2.1.
- The default WFS version is 2.0.0, for which the default output format is GML 3.2.1.
- All GML 3.2.1 responses are contained in a WFS 2.0.0 `FeatureCollection` element, even for WFS 1.1.0 requests, because a WFS 1.1.0 `FeatureCollection` cannot contain GML 3.2.1 features.

Secondary namespace for GML 3.2.1 required

GML 3.2.1 WFS responses are delivered in a WFS 2.0.0 `FeatureCollection`. Unlike WFS 1.1.0, WFS 2.0.0 does not depend explicitly on any GML version. As a consequence, the GML namespace is secondary and must be defined explicitly as a secondary namespace. See *Secondary Namespaces* for details.

For example, to use the prefix `gml` for GML 3.2, create `workspaces/gml/namespace.xml` containing:

```
<namespace>
    <id>gml_namespace</id>
    <prefix>gml</prefix>
    <uri>http://www.opengis.net/gml/3.2</uri>
</namespace>
```

and `workspaces/gml/workspace.xml` containing:

```
<workspace>
    <id>gml_workspace</id>
    <name>gml</name>
</workspace>
```

Failure to define the `gml` namespace prefix with a secondary namespace will result in errors like:

```
java.io.IOException: The prefix "null" for element "null:name" is not bound.
```

while encoding a response (in this case one containing `gml:name`), even if the namespace prefix is defined in the mapping file.

GML 3.2.1 geometries require gml:id

GML 3.2.1 requires that all geometries have a `gml:id`. While GeoServer will happily encode WFS responses without `gml:id` on geometries, these will be schema-invalid. Encoding a `gml:id` on a geometry can be achieved by setting an `idExpression` in the mapping for the geometry property. For example, `gsml:shape` is a geometry property and its `gml:id` might be generated with:

```
<AttributeMapping>
    <targetAttribute>gsml:shape</targetAttribute>
    <idExpression>
        <OCQL>strConcat('shape.', getId())</OCQL>
    </idExpression>
    <sourceExpression>
        <OCQL>SHAPE</OCQL>
    </sourceExpression>
</AttributeMapping>
```

In this example, `getId()` returns the `gml:id` of the containing feature, so each geometry will have a unique `gml:id` formed by appending the `gml:id` of the containing feature to the string `"shape."`.

If a multigeometry (such as a `MultiPoint` or `MultiSurface`) is assigned a `gml:id` of (for example) `parentid`, to permit GML 3.2.1 schema-validity, each geometry that the multigeometry contains will be automatically assigned a `gml:id` of the form `parentid.1`, `parentid.2`, ... in order.

9.7.3 GML 3.3

The proposed GML 3.3 is itself a GML 3.2.1 application schema; preliminary testing with drafts of GML 3.3 indicates that it works with app-schema as expected.

9.8 Secondary Namespaces

9.8.1 What is a secondary namespace?

A secondary namespace is one that is referenced indirectly by the main schema, that is, one schema imports another one as shown below:

```
a.xsd imports b.xsd
b.xsd imports c.xsd
```

(using a, b and c as the respective namespace prefixes for a.xsd, b.xsd and c.xsd):

```
a.xsd declares b:prefix
b.xsd declares c:prefix
```

The GeoTools encoder does not honour these namespaces and writes out:

```
"a:" , "b:" but NOT "c:"
```

The result is c's element being encoded as:

```
<null:cElement/>
```

9.8.2 When to configure for secondary namespaces

If your application spans several namespaces which may be very common in application schemas.

A sure sign that calls for secondary namespace configuration is when prefixes for namespaces are printed out as the literal string "null" or error messages like:

```
java.io.IOException: The prefix "null" for element "null:something" is not bound.
```

Note: When using secondary namespaces, requests involving complex featuretypes must be made to the **global OWS service** only, not to *Virtual OWS Services*. This is because virtual services are restricted to a single namespace, and thus are not able to access secondary namespaces.

In order to allow GeoServer App-Schema to support secondary namespaces, please follow the steps outlined below:

Using the sampling namespace as an example.

9.8.3 Step 1:Create the Secondary Namespace folder

Create a folder to represent the secondary namespace in the data/workspaces directory, in our example that will be the "sa" folder.

9.8.4 Step 2:Create files

Create two files below in the "sa" folder:

1. namespace.xml
2. workspace.xml

9.8.5 Step 3:Edit content of files

Contents of these files are as follows:

namespace.xml(uri is a valid uri for the secondary namespace, in this case the sampling namespace uri):

```
<namespace>
        <id>sa_workspace</id>
        <prefix>sa</prefix>
        <uri>http://www.opengis.net/sampling/1.0</uri>
</namespace>
```

workspace.xml:

```
<workspace>
        <id>sa_workspace</id>
        <name>sa</name>
</workspace>
```

That's it.

Your workspace is now configured to use a Secondary Namespace.

9.9 CQL functions

CQL functions enable data conversion and conditional behaviour to be specified in mapping files. Some of these functions are provided by the app-schema plugin specifically for this purpose.

- The uDig manual includes a list of CQL functions:
  - http://udig.refractions.net/confluence/display/EN/Constraint+Query+Language
- CQL string literals are enclosed in single quotes, for example `'urn:ogc:def:nil:OGC:missing'`.
- Single quotes are represented in CQL string literals as two single quotes, just as in SQL. For example, `'yyyy-MM-dd''T''HH:mm:ss''Z'''` for the string `yyyy-MM-dd'T'HH:mm:ss'Z'`.

9.9.1 Vocabulary translation

This section describes how to serve vocabulary translations using some function expressions in application schema mapping file. If you're not familiar with application schema mapping file, read *Mapping File*.

Recode

This is similar to *if_then_else* function, except that there is no default clause. You have to specify a translation value for every vocabulary key.

Syntax:

```
Recode(COLUMN_NAME, key1, value1, key2, value2,...)
```

- **COLUMN_NAME**: column name to get values from

Example:

```
<AttributeMapping>
  <targetAttribute>gml:name</targetAttribute>
  <sourceExpression>
      <OCQL>Recode(ABBREVIATION, '1GRAV', 'urn:cgi:classifier:CGI:SimpleLithology:2008:gravel',
                                 '1TILL', 'urn:cgi:classifier:CGI:SimpleLithology:2008:diamictite',
                                 '6ALLU', 'urn:cgi:classifier:CGI:SimpleLithology:2008:sediment')
      </OCQL>
  </sourceExpression>
</AttributeMapping>
```

The above example will map **gml:name** value to *urn:cgi:classifier:CGI:SimpleLithology:2008:gravel* if the ABBREVIATION column value is *1GRAV*.

Categorize

This is more suitable for numeric keys, where the translation value is determined by the key's position within the thresholds.

Syntax:

```
Categorize(COLUMN_NAME, default_value, threshold 1, value 1, threshold 2, value 2, ..., [preceding/su
```

- **COLUMN_NAME**: data source column name
- **default_value**: default value to be mapped if COLUMN_NAME value is not within the threshold
- **threshold(n)**: threshold value
- **value(n)**: value to be mapped if the threshold is met
- **preceding/succeeding:**
  - optional, succeeding is used by default if not specified.
  - not case sensitive.
  - preceding: value is within threshold if COLUMN_NAME value > threshold
  - succeeding: value is within threshold if COLUMN_NAME value >= threshold

Example:

```
<AttributeMapping>
  <targetAttribute>gml:description</targetAttribute>
  <sourceExpression>
      <OCQL>Categorize(CGI_LOWER_RANGE, 'missing_value', 1000, 'minor', 5000, 'significant')</OCQL>
  </sourceExpression>
</AttributeMapping>
```

The above example means **gml:description** value would be *significant* if CGI_LOWER_RANGE column value is >= *5000*.

Vocab

This function is more useful for bigger vocabulary pairs. Instead of writing a long key-to-value pairs in the function, you can keep them in a separate properties file. The properties file serves as a lookup table to the function. It has no header, and only contains the pairs in ''<key>=<value>" format.

Syntax:

```
Vocab(COLUMN_NAME, properties file URI)
```

- **COLUMN_NAME**: column name to get values from
- **properties file URI**: absolute path of the properties file or relative to the mapping file location

Example:

Properties file:

```
1GRAV=urn:cgi:classifier:CGI:SimpleLithology:2008:gravel
1TILL=urn:cgi:classifier:CGI:SimpleLithology:2008:diamictite
6ALLU=urn:cgi:classifier:CGI:SimpleLithology:2008:sediment
```

Mapping file:

```
<AttributeMapping>
  <targetAttribute>gml:name</targetAttribute>
  <sourceExpression>
      <OCQL>Vocab(ABBREVIATION, '/test-data/mapping.properties')</OCQL>
  </sourceExpression>
</AttributeMapping>
```

The above example will map **gml:name** to *urn:cgi:classifier:CGI:SimpleLithology:2008:gravel* if ABBREVIATION value is *1GRAV*.

9.9.2 Geometry creation

toDirectPosition

This function converts double values to `DirectPosition` geometry type. This is needed when the data store doesn't have geometry type columns. This function expects:

Literal `'SRS_NAME'` (optional)

Expression expression of SRS name if `'SRS_NAME'` is present as the first argument

Expression name of column pointing to first double value

Expression name of column pointing to second double value (optional, only for 2D)

ToEnvelope

`ToEnvelope` function can take in the following set of parameters and return as either `Envelope` or `ReferencedEnvelope` type:

Option 1 (1D Envelope):

```
ToEnvelope(minx,maxx)
```

Option 2 (1D Envelope with crsname):

```
ToEnvelope(minx,maxx,crsname)
```

Option 3 (2D Envelope):

```
ToEnvelope(minx,maxx,miny,maxy)
```

Option 4 (2D Envelope with crsname):

```
ToEnvelope(minx,maxx,miny,maxy,crsname)
```

toPoint

This function converts double values to a 2D Point geometry type. This is needed when the data store doesn't have geometry type columns. This function expects:

Literal `'SRS_NAME'` (optional)

Expression expression of SRS name if `'SRS_NAME'` is present as the first argument

Expression name of column pointing to first double value

Expression name of column pointing to second double value

Expression expression of gml:id (optional)

toLineString

This function converts double values to 1D LineString geometry type. This is needed to express 1D borehole intervals with custom (non EPSG) CRS.

Literal `'SRS_NAME'` (EPSG code or custom SRS)

Expression name of column pointing to first double value

Expression name of column pointing to second double value

9.9.3 Reference

toXlinkHref

This function redirects an attribute to be encoded as xlink:href, instead of being encoded as a full attribute. This is useful in polymorphism, where static client property cannot be used when the encoding is conditional. This function expects:

Expression REFERENCE_VALUE (could be another function or literal)

9.9.4 Date/time formatting

FormatDateTimezone

A function to format a date/time using a SimpleDateFormat pattern in a time zone supported by Java. This function improves on `dateFormat`, which formats date/time in the server time zone and can produce unintended results. Note that the term "date" is derived from a Java class name; this class represents a date/time, not just a single day.

Syntax:

```
FormatDateTimezone(pattern, date, timezone)
```

pattern formatting pattern supported by SimpleDateFormat, for example `'yyyy-MM-dd'`. Use two single quotes to include a literal single quote in a CQL string literal, for example `'yyyy-MM-dd''T''HH:mm:ss''Z'''`.

date the date/time to be formatted or its string representation, for example `'1948-01-01T00:00:00Z'`. An exception will be returned if the date is invalid. Database types with time zone information are recommended.

timezone the name of a time zone supported by Java, for example `'UTC'` or `'Canada/Mountain'`. Note that unrecognised timezones will silently be converted to UTC.

This example formats date/times from a column `POSITION` in UTC for inclusion in a `csml:TimeSeries`:

```
<AttributeMapping>
    <targetAttribute>csml:timePositionList</targetAttribute>
    <sourceExpression>
        <OCQL>FormatDateTimezone('yyyy-MM-dd''T''HH:mm:ss''Z''', POSITION, 'UTC')</OCQL>
    </sourceExpression>
    <isList>true</isList>
</AttributeMapping>
```

Note that any of the arguments could be sourced from a database column.

9.10 Property Interpolation

Interpolation in this context means the substitution of variables into strings. GeoServer app-schema supports the interpolation of properties (the Java equivalent of environment variables) into app-schema mapping files. This can be used, for example, to simplify the management of database connection parameters that would otherwise be hardcoded in a particular mapping file. This enables data directories to be given to third parties without inapplicable authentication or system configuration information. Externalising these parameters make management easier.

9.10.1 Defining properties

- If the system property `app-schema.properties` is not set, properties are loaded from `WEB-INF/classes/app-schema.properties` (or another resource `/app-schema.properties` on the classpath).
- If the system property `app-schema.properties` is set, properties are loaded from the file named as the value of the property. This is principally intended for debugging, and is designed to be used in an Eclipse launch configuration.
  - For example, if the JVM is started with `-Dapp-schema.properties=/path/to/some/local.properties`, properties are loaded from `/path/to/some/local.properties`.
- System properties override properties defined in a configuration file, so if you define `-Dsome.property` at the java command line, it will override a value specified in the `app-schema.properties` file. This is intended for debugging, so you can set a property file in an Eclipse launch configuration, but override some of the properties contained in the file by setting them explicitly as system properties.
- All system properties are available for interpolation in mapping files.

9.10.2 Using properties

- Using `${some.property}` anywhere in the mapping file will cause it to be replaced by the value of the property `some.property`.
- It is an error for a property that has not been set to be used for interpolation.
- Interpolation is performed repeatedly, so values can contain new interpolations. Use this behaviour with caution because it may cause an infinite loop.
- Interpolation is performed before XML parsing, so can be used to include arbitrary chunks of XML.

9.10.3 Example of property interpolation

This example defines an Oracle data store, where the connection parameter are interpolated from properties:

```
<sourceDataStores>
    <DataStore>
        <id>datastore</id>
        <parameters>
            <Parameter>
                <name>dbtype</name>
                <value>Oracle</value>
            </Parameter>
```

```
            <Parameter>
                <name>host</name>
                <value>${example.host}</value>
            </Parameter>
            <Parameter>
                <name>port</name>
                <value>1521</value>
            </Parameter>
            <Parameter>
                <name>database</name>
                <value>${example.database}</value>
            </Parameter>
            <Parameter>
                <name>user</name>
                <value>${example.user}</value>
            </Parameter>
            <Parameter>
                <name>passwd</name>
                <value>${example.passwd}</value>
            </Parameter>
        </parameters>
    </DataStore>
</sourceDataStores>
```

9.10.4 Example property file

This sample property file gives the property values that are interpolated into the mapping file fragment above. These properties can be installed in `WEB-INF/classes/app-schema.properties` in your GeoServer installation:

```
example.host = database.example.com
example.database = example
example.user = dbuser
example.passwd = s3cr3t
```

9.11 Data Stores

The app-schema *Mapping File* requires you to specify your data sources in the `sourceDataStores` section. For GeoServer simple features, these are configured using the web interface, but because app-schema lacks a web configuration interface, data stores must be configured by editing the mapping file.

Many configuration options may be externalised through the use of *Property Interpolation*.

9.11.1 The DataStore element

A `DataStore` configuration consists of

- an `id`, which is an opaque identifier used to refer to the data store elsewhere in a mapping file, and
- one or more `Parameter` elements, which each contain the `name` and `value` of one parameter, and are used to configure the data store.

An outline of the `DataStore` element:

```
<DataStore>
     <id>datastore</id>
     <parameters>
         <Parameter>
             <name>...</name>
             <value>...</value>
         </Parameter>
         ...
     </parameters>
</DataStore>
```

Parameter order is not significant.

9.11.2 Database options

Databases such as PostGIS, Oracle, and ArcSDE share some common or similar configuration options.

`name`	Meaning	`value` examples
`dbtype`	Database type	`postgisng`, `Oracle`, `arcsde`
`host`	Host name or IP address of database server	`database.example.org`, `192.168.3.12`
`port`	TCP port on database server	Default if omitted: `1521` (Oracle), `5432` (PostGIS), `5151` (ArcSDE)
`database`	PostGIS/Oracle database	
`instance`	ArcSDE instance	
`schema`	The database schema	
`user`	The user name used to login to the database server	
`passwd`	The password used to login to the database server	
`Expose primary keys`	Columns with primary keys available for mapping	Default is `false`, set to `true` to use primary key columns in mapping

9.11.3 PostGIS

Set the parameter `dbtype` to `postgisng` to use the PostGIS NG (New Generation) driver bundled with GeoServer 2.0 and later.

Example:

```
<DataStore>
    <id>datastore</id>
    <parameters>
        <Parameter>
            <name>dbtype</name>
            <value>postgisng</value>
        </Parameter>
        <Parameter>
            <name>host</name>
            <value>postgresql.example.org</value>
        </Parameter>
        <Parameter>
            <name>port</name>
            <value>5432</value>
        </Parameter>
```

```
        <Parameter>
            <name>database</name>
            <value>test</value>
        </Parameter>
        <Parameter>
            <name>user</name>
            <value>test</value>
        </Parameter>
        <Parameter>
            <name>passwd</name>
            <value>test</value>
        </Parameter>
    </parameters>
</DataStore>
```

Note: PostGIS support is included in the main GeoServer bundle, so a separate plugin is not required.

9.11.4 Oracle

Set the parameter `dbtype` to `Oracle` to use the Oracle Spatial NG (New Generation) driver compatible with GeoServer 2.0 and later.

Example:

```
<DataStore>
    <id>datastore</id>
    <parameters>
        <Parameter>
            <name>dbtype</name>
            <value>Oracle</value>
        </Parameter>
        <Parameter>
            <name>host</name>
            <value>oracle.example.org</value>
        </Parameter>
        <Parameter>
            <name>port</name>
            <value>1521</value>
        </Parameter>
        <Parameter>
            <name>database</name>
            <value>demodb</value>
        </Parameter>
        <Parameter>
            <name>user</name>
            <value>orauser</value>
        </Parameter>
        <Parameter>
            <name>passwd</name>
            <value>s3cr3t</value>
        </Parameter>
    </parameters>
</DataStore>
```

Note: You must install the Oracle plugin to connect to Oracle Spatial databases.

9.11.5 ArcSDE

This example connects to an ArcSDE database:

```
<DataStore>
    <id>datastore</id>
    <parameters>
        <Parameter>
            <name>dbtype</name>
            <value>arcsde</value>
        </Parameter>
        <Parameter>
            <name>server</name>
            <value>arcsde.example.org</value>
        </Parameter>
        <Parameter>
            <name>port</name>
            <value>5151</value>
        </Parameter>
        <Parameter>
            <name>instance</name>
            <value>sde</value>
        </Parameter>
        <Parameter>
            <name>user</name>
            <value>demo</value>
        </Parameter>
        <Parameter>
            <name>password</name>
            <value>s3cr3t</value>
        </Parameter>
        <Parameter>
            <name>datastore.allowNonSpatialTables</name>
            <value>true</value>
        </Parameter>
    </parameters>
</DataStore>
```

The use of non-spatial tables aids delivery of application schemas that use non-spatial properties.

Note: You must install the ArcSDE plugin to connect to ArcSDE databases.

9.11.6 Shapefile

Shapefile data sources are identified by the presence of a parameter `url`, whose value should be the file URL for the .shp file.

In this example, only the `url` parameter is required. The others are optional:

```
<DataStore>
    <id>shapefile</id>
    <parameters>
        <Parameter>
            <name>url</name>
            <value>file:/D:/Workspace/shapefiles/VerdeRiverBuffer.shp</value>
        </Parameter>
        <Parameter>
```

```
            <name>memory mapped buffer</name>
            <value>false</value>
        </Parameter>
        <Parameter>
            <name>create spatial index</name>
            <value>true</value>
        </Parameter>
        <Parameter>
            <name>charset</name>
            <value>ISO-8859-1</value>
        </Parameter>
    </parameters>
</DataStore>
```

Note: The `url` in this case is an example of a Windows filesystem path translated to URL notation.

Note: Shapefile support is included in the main GeoServer bundle, so a separate plugin is not required.

9.11.7 Property file

Property files are configured by specifying a `directory` that is a `file:` URI.

- If the directory starts with `file:./` it is relative to the mapping file directory. (This is an invalid URI, but it works.)

For example, the following data store is used to access property files in the same directory as the mapping file:

```
<DataStore>
    <id>propertyfile</id>
    <parameters>
        <Parameter>
            <name>directory</name>
            <value>file:./</value>
        </Parameter>
    </parameters>
</DataStore>
```

A property file data store contains *all* the feature types stored in .properties files in the directory. For example, if the directory contained River.properties and station.properties, the data store would be able to serve them as the feature types `River` and `station`. Other file extensions are ignored.

Note: Property file support is included in the main GeoServer bundle, so a separate plugin is not required.

9.11.8 JNDI

Defining a JDBC data store with a `jndiReferenceName` allows you to use a connection pool provided by your servlet container. This allows detailed configuration of connection pool parameters and sharing of connections between data sources, and even between servlets.

To use a JNDI connection provider:

1. Specify a `dbtype` parameter to to indicate the database type. These values are the same as for the non-JNDI examples above.

2. Give the `jndiReferenceName` you set in your servlet container. Both the abbreviated form `jdbc/oracle` form, as in Tomcat, and the canonical form `java:comp/env/jdbc/oracle` are supported.

This example uses JNDI to obtain Oracle connections:

```
<DataStore>
    <id>datastore</id>
    <parameters>
        <Parameter>
            <name>dbtype</name>
            <value>Oracle</value>
        </Parameter>
        <Parameter>
            <name>jndiReferenceName</name>
            <value>jdbc/oracle</value>
        </Parameter>
    </parameters>
</DataStore>
```

Your servlet container my require you to add a `resource-ref` section at the end of your `geoserver/WEB-INF/web.xml`. (Tomcat requires this, Jetty does not.) For example:

```
<resource-ref>
    <description>Oracle Spatial Datasource</description>
    <res-ref-name>jdbc/oracle</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
</resource-ref>
```

Here is an example of a Tomcat 6 context in `/etc/tomcat6/server.xml` that includes an Oracle connection pool:

```
<Context
    path="/geoserver"
    docBase="/usr/local/geoserver"
    crossContext="false"
    reloadable="false">
    <Resource
        name="jdbc/oracle"
        auth="Container"
        type="javax.sql.DataSource"
        url="jdbc:oracle:thin:@YOUR_DATABASE_HOSTNAME:1521:YOUR_DATABASE_NAME"
        driverClassName="oracle.jdbc.driver.OracleDriver"
        username="YOUR_DATABASE_USERNAME"
        password="YOUR_DATABASE_PASSWORD"
        maxActive="20"
        maxIdle="10"
        minIdle="0"
        maxWait="10000"
        minEvictableIdleTimeMillis="300000"
        timeBetweenEvictionRunsMillis="300000"
        numTestsPerEvictionRun="20"
        poolPreparedStatements="true"
        maxOpenPreparedStatements="100"
        testOnBorrow="true"
        validationQuery="SELECT SYSDATE FROM DUAL" />
</Context>
```

Firewall timeouts can silently sever idle connections to the database and cause GeoServer to hang. If there is

a firewall between GeoServer and the database, a connection pool configured to shut down idle connections before the firewall can drop them will prevent GeoServer from hanging. This JNDI connection pool is configured to shut down idle connections after 5 to 10 minutes.

See also *Setting up a JNDI connection pool with Tomcat*.

9.11.9 Expose primary keys

By default, GeoServer conceals the existence of database columns with a primary key. To make such columns available for use in app-schema mapping files, set the data store parameter `Expose primary keys` to `true`:

```
<Parameter>
    <name>Expose primary keys</name>
   <value>true</value>
</Parameter>
```

This is known to work with PostGIS, Oracle, and JNDI data stores.

9.12 Feature Chaining

9.12.1 Scope

This page describes the use of "Feature Chaining" to compose complex features from simpler components, and in particular to address some requirements that have proven significant in practice.

- Handling multiple cases of multi-valued properties within a single Feature Type
- Handing nesting of multi-valued properties within other multi-valued properties
- Linking related (through association) Feature Types, and in particular allowing re-use of the related features types (for example the O&M pattern has relatedObservation from a samplingFeature, but Observation may be useful in its own right)
- Encoding the same referenced property object as links when it appears in multiple containing features
- Eliminating the need for large denormalized data store views of top level features and their related features. Denormalized views would still be needed for special cases, such as many-to-many relationships, but won't be as large.

For non-application schema configurations, please refer to *Data Access Integration*.

9.12.2 Mapping steps

Create a mapping file for every complex type

We need one mapping file per complex type that is going to be nested, including non features, e.g. gsml:CompositionPart.

Non-feature types that cannot be individually accessed (eg. CompositionPart as a Data Type) can still be mapped separately for its reusability. For this case, the containing feature type has to include these types in its mapping file. The include tag should contain the nested mapping file path relative to the location of the containing type mapping file. In `GeologicUnit_MappingFile.xml`:

```
<includedTypes>
    <Include>CGITermValue_MappingFile.xml</Include>
    <Include>CompositionPart_MappingFile.xml</Include>
</includedTypes>
```

Feature types that can be individually accessed don't need to be explicitly included in the mapping file, as they would be configured for GeoServer to find. Such types would have their mapping file associated with a corresponding datastore.xml file, which means that it can be found from the data store registry. In other words, if the type is associated with a datastore.xml file, it doesn't need to be explicitly included if referred from another mapping file.

Example:

For this output: `MappedFeature_Output.xml`, here are the mapping files:

- `MappedFeature_MappingFile.xml`
- `GeologicUnit_MappingFile.xml`
- `CompositionPart_MappingFile.xml`
- `GeologicEvent_MappingFile.xml`
- `CGITermValue_MappingFile.xml`

GeologicUnit type

You can see within GeologicUnit features, both gml:composition (CompositionPart type) and gsml:geologicHistory (GeologicEvent type) are multi-valued properties. It shows how multiple cases of multi-valued properties can be configured within a single Feature Type. This also proves that you can "chain" non-feature type, as CompositionPart is a Data Type.

GeologicEvent type

Both gsml:eventEnvironment (CGI_TermValue type) and gsml:eventProcess (also of CGI_TermValue type) are multi-valued properties. This also shows that "chaining" can be done on many levels, as GeologicEvent is nested inside GeologicUnit. Note that gsml:eventAge properties are configured as inline attributes, as there can only be one event age per geologic event, thus eliminating the need for feature chaining.

Configure nesting on the nested feature type

In the nested feature type, make sure we have a field that can be referenced by the parent feature. If there isn't any existing field that can be referred to, the system field *FEATURE_LINK* can be mapped to hold the foreign key value. This is a multi-valued field, so more than one instances can be mapped in the same feature type, for features that can be nested by different parent types. Since this field doesn't exist in the schema, it wouldn't appear in the output document.

In the source expression tag:

- OCQL: the value of this should correspond to the OCQL part of the parent feature

Example One: Using *FEATURE_LINK* in CGI TermValue type, which is referred by GeologicEvent as gsml:eventProcess and gsml:eventEnvironment.

In GeologicEvent (the container feature) mapping:

```
<AttributeMapping>
      <targetAttribute>gsml:eventEnvironment</targetAttribute>
      <sourceExpression>
              <OCQL>id</OCQL>
              <linkElement>gsml:CGI_TermValue</linkElement>
```

```
                <linkField>FEATURE_LINK[1]</linkField>
        </sourceExpression>
        <isMultiple>true</isMultiple>
</AttributeMapping>
<AttributeMapping>
        <targetAttribute>gsml:eventProcess</targetAttribute>
        <sourceExpression>
                <OCQL>id</OCQL>
                <linkElement>gsml:CGI_TermValue</linkElement>
                <linkField>FEATURE_LINK[2]</linkField>
        </sourceExpression>
        <isMultiple>true</isMultiple>
</AttributeMapping>
```

In CGI_TermValue (the nested feature) mapping:

```
<AttributeMapping>
  <!-- FEATURE_LINK[1] is referred by geologic event as environment -->
  <targetAttribute>FEATURE_LINK[1]</targetAttribute>
  <sourceExpression>
      <OCQL>ENVIRONMENT_OWNER</OCQL>
  </sourceExpression>
</AttributeMapping>
<AttributeMapping>
  <!-- FEATURE_LINK[2] is referred by geologic event as process -->
  <targetAttribute>FEATURE_LINK[2]</targetAttribute>
  <sourceExpression><
      <OCQL>PROCESS_OWNER</OCQL>
  </sourceExpression>
</AttributeMapping>
```

The ENVIRONMENT_OWNER column in CGI_TermValue view corresponds to the ID column in GeologicEvent view.

Geologic Event property file:

id	GEOLOGIC_UNIT_ID:String	ghmin-age:String	ghmax-age:String	ghage_cdspace:String
ge.26931120	gu.25699	Oligocene	Paleocene	urn:cgi:classifierScheme:ICS:StratChart:2008
ge.26930473	gu.25678	Holocene	Pleistocene	urn:cgi:classifierScheme:ICS:StratChart:2008
ge.26930960	gu.25678	Pliocene	Miocene	urn:cgi:classifierScheme:ICS:StratChart:2008
ge.26932959	gu.25678	LowerOrdovician	LowerOrdovician	urn:cgi:classifierScheme:ICS:StratChart:2008

CGI Term Value property file:

id	VALUE:String	PROCESS_OWNER:String	ENVIRONMENT_OWNER:String
3	fluvial	NULL	ge.26931120
4	swamp/marsh/bog	NULL	ge.26930473
5	marine	NULL	ge.26930960
6	submarine fan	NULL	ge.26932959
7	hemipelagic	NULL	ge.26932959
8	detrital deposition still water	ge.26930473	NULL
9	water [process]	ge.26932959	NULL
10	channelled stream flow	ge.26931120	NULL
11	turbidity current	ge.26932959	NULL

The system field *FEATURE_LINK* doesn't get encoded in the output:

```
<gsml:GeologicEvent>
  <gml:name codeSpace="urn:cgi:classifierScheme:GSV:GeologicalUnitId">gu.25699</gml:name>
  <gsml:eventAge>
    <gsml:CGI_TermRange>
       <gsml:lower>
          <gsml:CGI_TermValue>
            <gsml:value codeSpace="urn:cgi:classifierScheme:ICS:StratChart:2008">Oligocene</gsml:valu
          </gsml:CGI_TermValue>
       </gsml:lower>
       <gsml:upper>
          <gsml:CGI_TermValue>
            <gsml:value codeSpace="urn:cgi:classifierScheme:ICS:StratChart:2008">Paleocene</gsml:valu
          </gsml:CGI_TermValue>
       </gsml:upper>
    </gsml:CGI_TermRange>
  </gsml:eventAge>
  <gsml:eventEnvironment>
    <gsml:CGI_TermValue>
       <gsml:value>fluvial</gsml:value>
    </gsml:CGI_TermValue>
  </gsml:eventEnvironment>
  <gsml:eventProcess>
    <gsml:CGI_TermValue>
       <gsml:value>channelled stream flow</gsml:value>
    </gsml:CGI_TermValue>
  </gsml:eventProcess>
```

Example Two: Using existing field (gml:name) to hold the foreign key, see `MappedFeature_MappingFile.xml`:

gsml:specification links to gml:name in GeologicUnit:

```
<AttributeMapping>
  <targetAttribute>gsml:specification</targetAttribute>
  <sourceExpression>
    <OCQL>GEOLOGIC_UNIT_ID</OCQL>
    <linkElement>gsml:GeologicUnit</linkElement>
    <linkField>gml:name[3]</linkField>
  </sourceExpression>
</AttributeMapping>
```

In `GeologicUnit_MappingFile.xml`:

GeologicUnit has 3 gml:name properties in the mapping file, so each has a code space to clarify them:

```
<AttributeMapping>
  <targetAttribute>gml:name[1]</targetAttribute>
  <sourceExpression>
    <OCQL>ABBREVIATION</OCQL>
  </sourceExpression>
  <ClientProperty>
    <name>codeSpace</name>
    <value>'urn:cgi:classifierScheme:GSV:GeologicalUnitCode'</value>
  </ClientProperty>
</AttributeMapping>
<AttributeMapping>
  <targetAttribute>gml:name[2]</targetAttribute>
  <sourceExpression>
    <OCQL>NAME</OCQL>
```

```
  </sourceExpression>
  <ClientProperty>
    <name>codeSpace</name>
    <value>'urn:cgi:classifierScheme:GSV:GeologicalUnitName'</value>
  </ClientProperty>
</AttributeMapping>
<AttributeMapping>
  <targetAttribute>gml:name[3]</targetAttribute>
  <sourceExpression>
    <OCQL>id</OCQL>
  </sourceExpression>
  <ClientProperty>
    <name>codeSpace</name>
    <value>'urn:cgi:classifierScheme:GSV:MappedFeatureReference'</value>
  </ClientProperty>
</AttributeMapping>
```

The output with multiple gml:name properties and their code spaces:

```
<gsml:specification>
  <gsml:GeologicUnit gml:id="gu.25678">
      <gml:description>Olivine basalt, tuff, microgabbro, minor sedimentary rocks</gml:description>
      <gml:name codeSpace="urn:cgi:classifierScheme:GSV:GeologicalUnitCode">-Py</gml:name>
      <gml:name codeSpace="urn:cgi:classifierScheme:GSV:GeologicalUnitName">Yaugher Volcanic Group</g
      <gml:name codeSpace="urn:cgi:classifierScheme:GSV:MappedFeatureReference">gu.25678</gml:name>
```

If this is the "one" side of a one-to-many or many-to-one database relationship, we can use the feature id as the source expression field, as you can see in above examples. See `one_to_many_relationship.JPG` as an illustration.

If we have a many-to-many relationship, we have to use one denormalized view for either side of the nesting. This means we can either use the feature id as the referenced field, or assign a column to serve this purpose. See `many_to_many_relationship.JPG` as an illustration.

Note:

- For many-to-many relationships, we can't use the same denormalized view for both sides of the nesting.

Test this configuration by running a getFeature request for the nested feature type on its own.

Configure nesting on the "containing" feature type

When nesting another complex type, you need to specify in your source expression:

- **OCQL**: OGC's Common Query Language expression of the data store column
- **linkElement:**
  - the nested element name, which is normally the targetElement or mappingName of the corresponding type.
  - on some cases, it has to be an OCQL function (see *Polymorphism*)
- **linkField**: the indexed XPath attribute on the nested element that OCQL corresponds to

Example: Nesting composition part in geologic unit feature.

In Geologic Unit mapping file:

```
<AttributeMapping>
    <targetAttribute>gsml:composition</targetAttribute>
    <sourceExpression>
            <OCQL>id</OCQL>
            <linkElement>gsml:CompositionPart</linkElement>
            <linkField>FEATURE_LINK</linkField>
    </sourceExpression>
    <isMultiple>true</isMultiple>
</AttributeMapping>
```

- *OCQL*: id is the geologic unit id
- *linkElement*: links to gsml:CompositionPart type
- *linkField*: FEATURE_LINK, the linking field mapped in gsml:CompositionPart type that also stores the geologic unit id. If there are more than one of these attributes in the nested feature type, make sure the index is included, e.g. FEATURE_LINK[2].

Geologic Unit property file:

id	ABBREVIATAION:String	NAME:String	TEXTDESCRIPTION:String
gu.25699	-Py	Yaugher Volcanic Group	Olivine basalt, tuff, microgabbro, minor sedimentary rocks
gu.25678	-Py	Yaugher Volcanic Group	Olivine basalt, tuff, microgabbro, minor sedimentary rocks

Composition Part property file:

id	COMPONENT_ROLE:String	PROPORTION:String	GEOLOGIC_UNIT_ID:String
cp.167775491936278812	interbedded component	significant	gu.25699
cp.167775491936278856	interbedded component	minor	gu.25678
cp.167775491936278844	sole component	major	gu.25678

Run the getFeature request to test this configuration. Check that the nested features returned in Step 2 are appropriately lined inside the containing features. If they are not there, or exceptions are thrown, scroll down and read the "Trouble Shooting" section.

9.12.3 Multiple mappings of the same type

At times, you may find the need to have different FeatureTypeMapping instances for the same type. You may have two different attributes of the same type that need to be nested. For example, in gsml:GeologicUnit, you have gsml:exposureColor and gsml:outcropCharacter that are both of gsml:CGI_TermValue type.

This is when the optional mappingName tag mentioned in *Mapping File* comes in. Instead of passing in the nested feature type's targetElement in the containing type's linkElement, specify the corresponding mappingName.

Note:

- The mappingName is namespace aware and case sensitive.
- When the referred mappingName contains special characters such as '-', it must be enclosed with single quotes in the linkElement. E.g. <linkElement>'observation-method'</linkElement>.
- Each mappingName must be unique against other mappingName and targetElement tags across the application.

- The mappingName is only to be used to identify the chained type from the nesting type. It is not a solution for multiple FeatureTypeMapping instances where > 1 of them can be queried as top level features.
- When queried as a top level feature, the normal targetElement is to be used. Filters involving the nested type should still use the targetElement in the PropertyName part of the query.
- You can't have more than 1 FeatureTypeMapping of the same type in the same mapping file if one of them is a top level feature. This is because featuretype.xml would look for the targetElement and wouldn't know which one to get.

The solution for the last point above is to break them up into separate files and locations with only 1 featuretype.xml in the intended top level feature location. E.g.

- You can have 2 FeatureTypeMapping instances in the same file for gsml:CGI_TermValue type since it's not a feature type.
- You can have 2 FeatureTypeMapping instances for gsml:MappedFeature, but they have to be broken up into separate files. The one that can be queried as top level feature type would have featuretype.xml in its location.

9.12.4 Nesting simple properties

You don't need to chain multi-valued simple properties and map them separately. The original configuration would still work.

9.12.5 Filtering nested attributes on chained features

Filters would work as usual. You can supply the full XPath of the attribute, and the code would handle this. E.g. You can run the following filter on gsml:MappedFeatureUseCase2A:

```
<ogc:Filter>
      <ogc:PropertyIsEqualTo>
          <ogc:Function name="contains_text">
              <ogc:PropertyName>gsml:specification/gsml:GeologicUnit/gml:description</ogc:PropertyNar
              <ogc:Literal>Olivine basalt, tuff, microgabbro, minor sedimentary rocks</ogc:Literal>
          </ogc:Function>
          <ogc:Literal>1</ogc:Literal>
      </ogc:PropertyIsEqualTo>
</ogc:Filter>
```

9.12.6 Multi-valued properties by reference (*xlink:href*)

You may want to use feature chaining to set multi-valued properties by reference. This is particularly handy to avoid endless loop in circular relationships. For example, you may have a circular relationship between gsml:MappedFeature and gsml:GeologicUnit. E.g.

- gsml:MappedFeature has gsml:GeologicUnit as gsml:specification
- gsml:GeologicUnit has gsml:MappedFeature as gsml:occurrence

Obviously you can only encode one side of the relationship, or you'll end up with an endless loop. You would need to pick one side to "chain" and use xlink:href for the other side of the relationship.

For this example, we are nesting gsml:GeologicUnit in gsml:MappedFeature as gsml:specification.

- Set up nesting on the container feature type mapping as usual:

```
<AttributeMapping>
  <targetAttribute>gsml:specification</targetAttribute>
  <sourceExpression>
      <OCQL>GEOLOGIC_UNIT_ID</OCQL>
        <linkElement>gsml:GeologicUnit</linkElement>
        <linkField>gml:name[2]</linkField>
  </sourceExpression>
</AttributeMapping>
```

- Set up xlink:href as client property on the other mapping file:

```
<AttributeMapping>
  <targetAttribute>gsml:occurrence</targetAttribute>
  <sourceExpression>
        <OCQL>id</OCQL>
        <linkElement>gsml:MappedFeature</linkElement>
        <linkField>gsml:specification</linkField>
  </sourceExpression>
  <isMultiple>true</isMultiple>
  <ClientProperty>
         <name>xlink:href</name>
         <value>strConcat('urn:cgi:feature:MappedFeature:', ID)</value>
  </ClientProperty>
</AttributeMapping>
```

As we are getting the client property value from a nested feature, we have to set it as if we are chaining the feature; but we also add the client property containing *xlink:href* in the attribute mapping. The code will detect the *xlink:href* setting, and will not proceed to build the nested feature's attributes, and we will end up with empty attributes with *xlink:href* client properties.

This would be the encoded result for gsml:GeologicUnit:

```
<gsml:GeologicUnit gml:id="gu.25678">
         <gsml:occurrence xlink:href="urn:cgi:feature:MappedFeature:mf2"/>
         <gsml:occurrence xlink:href="urn:cgi:feature:MappedFeature:mf3"/>
```

Note:

- Don't forget to add *XLink* in your mapping file namespaces section, or you could end up with a StackOverflowException as the *xlink:href* client property won't be recognized and the mappings would chain endlessly.
- *Resolving* may be used to force app-schema to do full feature chaining up to a certain level, even if an xlink reference is specified.

9.13 Polymorphism

Polymorphism in this context refers to the ability of an attribute to have different forms. Depending on the source value, it could be encoded with a specific structure, type, as an xlink:href reference, or not encoded at all. To achieve this, we reuse feature chaining syntax and allow OCQL functions in the linkElement tag. Read more about *Feature Chaining*, if you're not familiar with the syntax.

9.13.1 Data-type polymorphism

You can use normal feature chaining to get an attribute to be encoded as a certain type. For example:

```
<AttributeMapping>
    <targetAttribute>ex:someAttribute</targetAttribute>
    <sourceExpression>
        <OCQL>VALUE_ID</OCQL>
          <linkElement>NumericType</linkElement>
          <linkField>FEATURE_LINK</linkField>
    </sourceExpression>
</AttributeMapping>
<AttributeMapping>
    <targetAttribute>ex:someAttribute</targetAttribute>
    <sourceExpression>
          <OCQL>VALUE_ID</OCQL>
          <linkElement>gsml:CGI_TermValue</linkElement>
          <linkField>FEATURE_LINK</linkField>
    </sourceExpression>
</AttributeMapping>
```

Note: NumericType here is a mappingName, whereas gsml:CGI_TermValue is a targetElement.

In the above example, ex:someAttribute would be encoded with the configuration in NumericType if the foreign key matches the linkField. Both instances would be encoded if the foreign key matches the candidate keys in both linked configurations. Therefore this would only work for 0 to many relationships.

Functions can be used for single attribute instances. See useful functions for a list of commonly used functions. Specify the function in the linkElement, and it would map it to the first matching FeatureTypeMapping. For example:

```
<AttributeMapping>
    <targetAttribute>ex:someAttribute</targetAttribute>
    <sourceExpression>
          <OCQL>VALUE_ID</OCQL>
          <linkElement>
            Recode(CLASS_TEXT, 'numeric', 'NumericType', 'literal', 'gsml:CGI_TermValue')
        </linkElement>
          <linkField>FEATURE_LINK</linkField>
    </sourceExpression>
    <isMultiple>true</isMultiple>
</AttributeMapping>
```

The above example means, if the CLASS_TEXT value is 'numeric', it would link to 'NumericType' FeatureTypeMapping, with VALUE_ID as foreign key to the linked type. It would require all the potential matching types to have a common attribute that is specified in linkField. In this example, the linkField is FEATURE_LINK, which is a fake attribute used only for feature chaining. You can omit the linkField and OCQL if the FeatureTypeMapping being linked to has the same sourceType with the container type. This would save us from unnecessary extra queries, which would affect performance. For example:

FeatureTypeMapping of the container type:

```
<FeatureTypeMapping>
    <sourceDataStore>PropertyFiles</sourceDataStore>
    <sourceType>PolymorphicFeature</sourceType>
```

FeatureTypeMapping of NumericType points to the same table:

```
<FeatureTypeMapping>
    <mappingName>NumericType</mappingName>
```

```
    <sourceDataStore>PropertyFiles</sourceDataStore>
    <sourceType>PolymorphicFeature</sourceType>
```

FeatureTypeMapping of gsml:CGI_TermValue also points to the same table:

```
<FeatureTypeMapping>
    <sourceDataStore>PropertyFiles</sourceDataStore>
    <sourceType>PolymorphicFeature</sourceType>
    <targetElement>gsml:CGI_TermValue</targetElement>
```

In this case, we can omit linkField in the polymorphic attribute mapping:

```
<AttributeMapping>
    <targetAttribute>ex:someAttribute</targetAttribute>
    <sourceExpression>
          <linkElement>
            Recode(CLASS_TEXT, 'numeric', 'NumericType', 'literal', 'gsml:CGI_TermValue')
        </linkElement>
    </sourceExpression>
    <isMultiple>true</isMultiple>
</AttributeMapping>
```

9.13.2 Referential polymorphism

This is when an attribute is set to be encoded as an xlink:href reference on the top level. When the scenario only has reference cases in it, setting a function in Client Property will do the job. E.g.:

```
<AttributeMapping>
      <targetAttribute>ex:someAttribute</targetAttribute>
      <ClientProperty>
            <name>xlink:href</name>
            <value>if_then_else(isNull(NUMERIC_VALUE), 'urn:ogc:def:nil:OGC:1.0:missing', strConcat('
    </ClientProperty>
</AttributeMapping>
```

The above example means, if NUMERIC_VALUE is null, the attribute should be encoded as:

```
<ex:someAttribute xlink:href="urn:ogc:def:nil:OGC:1.0:missing">
```

Otherwise, it would be encoded as:

```
<ex:someAttribute xlink:href="#123">
    where NUMERIC_VALUE = '123'
```

However, this is not possible when we have cases where a fully structured attribute is also a possibility. The toxlinkhref function can be used for this scenario. E.g.:

```
<AttributeMapping>
    <targetAttribute>ex:someAttribute</targetAttribute>
    <sourceExpression>
          <linkElement>
            if_then_else(isNull(NUMERIC_VALUE), toXlinkHref('urn:ogc:def:nil:OGC:1.0:missing'),
              if_then_else(lessEqualThan(NUMERIC_VALUE, 1000), 'numeric_value', toXlinkHref('urn:ogc:
        </linkElement>
    </sourceExpression>
</AttributeMapping>
```

The above example means, if NUMERIC_VALUE is null, the output would be encoded as:

```
<ex:someAttribute xlink:href="urn:ogc:def:nil:OGC:1.0:missing">
```

Otherwise, if NUMERIC_VALUE is less or equal than 1000, it would be encoded with attributes from FeatureTypeMapping with 'numeric_value' mappingName. If NUMERIC_VALUE is greater than 1000, it would be encoded as the first scenario.

9.13.3 Useful functions

if_then_else function

Syntax:

```
if_then_else(BOOLEAN_EXPRESSION, value, default value)
```

- **BOOLEAN_EXPRESSION**: could be a Boolean column value, or a Boolean function
- **value**: the value to map to, if BOOLEAN_EXPRESSION is true
- **default value**: the value to map to, if BOOLEAN_EXPRESSION is false

Recode function

Syntax:

```
Recode(EXPRESSION, key1, value1, key2, value2,...)
```

- **EXPRESSION**: column name to get values from, or another function
- **key-n:**
  - key expression to map to value-n
  - if the evaluated value of EXPRESSION doesn't match any key, nothing would be encoded for the attribute.
- **value-n**: value expression which translates to a mappingName or targetElement

lessEqualThan

Returns true if ATTRIBUTE_EXPRESSION evaluates to less or equal than LIMIT_EXPRESSION.

Syntax:

```
lessEqualThan(ATTRIBUTE_EXPRESSION, LIMIT_EXPRESSION)
```

- **ATTRIBUTE_EXPRESSION**: expression of the attribute being evaluated.
- **LIMIT_EXPRESSION**: expression of the numeric value to be compared against.

lessThan

Returns true if ATTRIBUTE_EXPRESSION evaluates to less than LIMIT_EXPRESSION.

Syntax:

```
lessThan(ATTRIBUTE_EXPRESSION, LIMIT_EXPRESSION)
```

- **ATTRIBUTE_EXPRESSION**: expression of the attribute being evaluated.

- **LIMIT_EXPRESSION**: expression of the numeric value to be compared against.

equalTo

Compares two expressions and returns true if they're equal.

Syntax:

```
equalTo(LHS_EXPRESSION, RHS_EXPRESSION)
```

isNull

Returns a Boolean that is true if the expression evaluates to null.

Syntax:

```
isNull(EXPRESSION)
```

- **EXPRESSION**: expression to be evaluated.

toXlinkHref

Special function written for referential polymorphism and feature chaining, not to be used outside of linkElement. It infers that the attribute should be encoded as xlink:href.

Syntax:

```
toXlinkHref(XLINK_HREF_EXPRESSION)
```

- **XLINK_HREF_EXPRESSION:**
  - could be a function or a literal
  - has to be wrapped in single quotes if it's a literal

Note:

- To get toXlinkHref function working, you need to declare xlink URI in the namespaces.

Other functions

Please refer to *Filter Function Reference*.

Combinations

You can combine functions, but it might affect performance. E.g.:

```
if_then_else(isNull(NUMERIC_VALUE), toXlinkHref('urn:ogc:def:nil:OGC:1.0:missing'),
    if_then_else(lessEqualThan(NUMERIC_VALUE, 1000), 'numeric_value', toXlinkHref('urn:ogc:def:nil:OG
```

Note:

- When specifying a mappingName or targetElement as a value in functions, make sure they're enclosed in single quotes.
- Some functions have no null checking, and will fail when they encounter null.

- The workaround for this is to wrap the expression with isNull() function if null is known to exist in the data set.

9.13.4 Null or missing value

To skip the attribute for a specific case, you can use Expression.NIL as a value in if_then_else or not include the key in Recode function . E.g.:

```
if_then_else(isNull(VALUE), Expression.NIL, 'gsml:CGI_TermValue')
    means the attribute would not be encoded if VALUE is null.

Recode(VALUE, 'term_value', 'gsml:CGI_TermValue')
    means the attribute would not be encoded if VALUE is anything but 'term_value'.
```

To encode an attribute as xlink:href that represents missing value on the top level, see Referential Polymorphism.

9.13.5 Any type

Having xs:anyType as the attribute type itself infers that it is polymorphic, since they can be encoded as any type.

If the type is pre-determined and would always be the same, we might need to specify *targetAttributeNode (optional)*. E.g.:

```
<AttributeMapping>
      <targetAttribute>om:result</targetAttribute>
      <targetAttributeNode>gml:MeasureType<targetAttributeNode>
      <sourceExpression>
          <OCQL>TOPAGE</OCQL>
      </sourceExpression>
      <ClientProperty>
          <name>xsi:type</name>
          <value>'gml:MeasureType'</value>
      </ClientProperty>
      <ClientProperty>
          <name>uom</name>
          <value>'http://www.opengis.net/def/uom/UCUM/0/Ma'</value>
      </ClientProperty>
</AttributeMapping>
```

If the casting type is complex, this is not a requirement as app-schema is able to automatically determine the type from the XPath in targetAttribute. E.g., in this example `om:result` is automatically specialised as a MappedFeatureType:

```
<AttributeMapping>
      <targetAttribute>om:result/gsml:MappedFeature/gml:name</targetAttribute>
      <sourceExpression>
          <OCQL>NAME</OCQL>
      </sourceExpression>
</AttributeMapping>
```

Alternatively, we can use feature chaining. For the same example above, the mapping would be:

```
<AttributeMapping>
      <targetAttribute>om:result</targetAttribute>
```

```
      <sourceExpression>
            <OCQL>LEX_D</OCQL>
            <linkElement>gsml:MappedFeature</linkElement>
            <linkField>gml:name</linkField>
      </sourceExpression>
</AttributeMapping>
```

If the type is conditional, the mapping style for such attributes is the same as any other polymorphic attributes. E.g.:

```
<AttributeMapping>
      <targetAttribute>om:result</targetAttribute>
      <sourceExpression>
            <linkElement>
               Recode(NAME, Expression.Nil, toXlinkHref('urn:ogc:def:nil:OGC::missing'),'numeric',
               toXlinkHref(strConcat('urn:numeric-value::', NUMERIC_VALUE)), 'literal', 'TermValue2')
            </linkElement>
      </sourceExpression>
</AttributeMapping>
```

9.13.6 Filters

Filters should work as usual, as long as the users know what they want to filter. For example, when an attribute could be encoded as gsml:CGI_TermValue or gsml:CGI_NumericValue, users can run filters with property names of:

- ex:someAttribute/gsml:CGI_TermValue/gsml:value to return matching attributes that are encoded as gsml:CGI_TermValue and satisfy the filter.
- likewise, ex:someAttribute/gsml:CGI_NumericValue/gsml:principalValue should return matching gsml:CGI_NumericValue attributes.

Another limitation is filtering attributes of an xlink:href attribute pointing to an instance outside of the document.

9.14 Data Access Integration

This page assumes prior knowledge of *Working with Application Schemas* and *Feature Chaining*. To use feature chaining, the nested features can come from any complex feature data access, as long as: * it has valid data referred by the "container" feature type, * the data access is registered via DataAccessRegistry, * if FEATURE_LINK is used as the link field, the feature types were created via ComplexFeatureTypeFactoryImpl

However, the "container" features must come from an application schema data access. The rest of this article describes how we can create an application data access from an existing non-application schema data access, in order to "chain" features. The input data access referred in this article is assumed to be the non-application schema data access.

9.14.1 How to connect to the input data access

Configure the data store connection in "sourceDataStores" tag as usual, but also specify the additional "isDataAccess" tag. This flag marks that we want to get the registered complex feature source of the specified "sourceType", when processing the source data store. This assumes that the input data access is registered in DataAccessRegistry upon creation, for the system to find it.

Example:

```
<sourceDataStores>
  <DataStore>
      <id>EarthResource</id>
      <parameters>
         <Parameter>
           <name>directory</name>
           <value>file:./</value>
         </Parameter>
      </parameters>
      <isDataAccess>true</isDataAccess>
  </DataStore>
</sourceDataStores>
...
<typeMappings>
  <FeatureTypeMapping>
    <sourceDataStore>EarthResource</sourceDataStore>
      <sourceType>EarthResource</sourceType>
...
```

9.14.2 How to configure the mapping

Use "inputAttribute" in place of "OCQL" tag inside "sourceExpression", to specify the input XPath expressions.

Example:

```
<AttributeMapping>
  <targetAttribute>gsml:classifier/gsml:ControlledConcept/gsml:preferredName</targetAttribute>
  <sourceExpression>
      <inputAttribute>mo:classification/mo:MineralDepositModel/mo:mineralDepositGroup</inputAttribute
  </sourceExpression>
</AttributeMapping>
```

9.14.3 How to chain features

Feature chaining works both ways for the re-mapped complex features. You can chain other features inside these features, and vice-versa. The only difference is to use "inputAttribute" for the input XPath expressions, instead of "OCQL" as mentioned above.

Example:

```
<AttributeMapping>
  <targetAttribute>gsml:occurence</targetAttribute>
  <sourceExpression>
      <inputAttribute>mo:commodityDescription</inputAttribute>
      <linkElement>gsml:MappedFeature</linkElement>
      <linkField>gml:name[2]</linkField>
  </sourceExpression>
  <isMultiple>true</isMultiple>
</AttributeMapping>
```

9.14.4 How to use filters

From the user point of view, filters are configured as per normal, using the mapped/output target attribute XPath expressions. However, when one or more attributes in the expression is a multi-valued property, we need to specify a function such as "contains_text" in the filter. This is because when multiple values are returned, comparing them to a single value would only return true if there is only one value returned, and it is the same value. Please note that the "contains_text" function used in the following example is not available in Geoserver API, but defined in the database.

Example:

Composition is a multi-valued property:

```
<ogc:Filter>
  <ogc:PropertyIsEqualTo>
    <ogc:Function name="contains_text">
        <ogc:PropertyName>gsml:composition/gsml:CompositionPart/gsml:proportion/gsml:CGI_TermValue/gs
        <ogc:Literal>Olivine basalt, tuff, microgabbro, minor sedimentary rocks</ogc:Literal>
    </ogc:Function>
    <ogc:Literal>1</ogc:Literal>
  </ogc:PropertyIsEqualTo>
</ogc:Filter>
```

9.15 WMS Support

App-schema supports WMS requests as well as WFS requests. This page provides some useful examples for configuring the WMS service to work with complex features.

Note that the rendering performance of WMS can be significantly slower when using app-schema data stores. We strongly recommend employing *Joining Support For Performance* when using WMS with feature chaining, which can make response time for large data requests several orders of magnitude faster.

9.15.1 Configuration

For WMS to be applicable to complex feature data, it is necessary that the complex feature types are recognised by GeoServer as layers. This must be configured by adding an extra configuration file named 'layer.xml' in the data directory of each feature type that we want to use as a WMS layer.

This will expand the structure of the `workspaces` folder in the GeoServer data directory as follows (`workspaces`) (see *Configuration*):

```
workspaces
    - gsml
        - SomeDataStore
            - SomeFeatureType
                - featuretype.xml
                - layer.xml
            - datastore.xml
            - SomeFeatureType-mapping-file.xml
```

The file layer.xml must have the following contents:

```
<layer>
  <id>[mylayerid]</id>
  <name>[mylayername]</name>
  <path>/</path>
```

```
  <type>VECTOR</type>
  <defaultStyle>
        <name>[mydefaultstyle]</name>
  </defaultStyle>
  <resource class="featureType">
        <id>[myfeaturetypeid]</id>
  </resource>
  <enabled>true</enabled>
  <attribution>
        <logoWidth>0</logoWidth>
        <logoHeight>0</logoHeight>
  </attribution>
</layer>
```

Replace the fields in between brackets with the following values:

- **[mylayerid]** must be a custom id for the layer.
- **[mylayername]** must be a custom name for the layer.
- **[mydefaultstyle]** the default style used for this layer (when a style is not specified in the wms request). The style must exist in the GeoServer configuration.
- **[myfeaturetypeid]** is the id of the feature type. This *must* the same as the id specified in the file featuretype.xml of the same directory.

9.15.2 GetMap

Read *GetMap* for general information on the GetMap request. Read *Styling* for general information on how to style WMS maps with SLD files. When styling complex features, you can use XPaths to specify nested properties in your filters, as explained in *Filtering nested attributes on chained features*. However, in WMS styling filters X-paths do not support handling referenced features (see *Multi-valued properties by reference (xlink:href)*) as if they were actual nested features (because the filters are applied after building the features rather than before.) The prefix/namespace context that is used in the XPath expression is defined locally in the XML tags of the style file. This is an example of a Style file for complex features:

```
<?xml version="1.0" encoding="UTF-8"?>
<StyledLayerDescriptor version="1.0.0"
    xsi:schemaLocation="http://www.opengis.net/sld StyledLayerDescriptor.xsd"
    xmlns:ogc="http://www.opengis.net/ogc"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:gsml="urn:cgi:xmlns:CGI:GeoSciML:2.0"
    xmlns:sld="http://www.opengis.net/sld"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <sld:NamedLayer>
  <sld:Name>geology-lithology</sld:Name>
  <sld:UserStyle>
    <sld:Name>geology-lithology</sld:Name>
    <sld:Title>Geological Unit Lithology Theme</sld:Title>
    <sld:Abstract>The colour has been creatively adapted from Moyer,Hasting
         and Raines, 2005 (http://pubs.usgs.gov/of/2005/1314/of2005-1314.pdf)
         which provides xls spreadsheets for various color schemes.
         plus some creative entries to fill missing entries.
    </sld:Abstract>
    <sld:IsDefault>1</sld:IsDefault>
    <sld:FeatureTypeStyle>
      <sld:Rule>
```

```
        <sld:Name>acidic igneous material</sld:Name>
        <sld:Abstract>Igneous material with more than 63 percent SiO2.
                      (after LeMaitre et al. 2002)
        </sld:Abstract>
        <ogc:Filter>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>gsml:specification/gsml:GeologicUnit/gsml:composition/
                 gsml:CompositionPart/gsml:lithology/@xlink:href</ogc:PropertyName>
            <ogc:Literal>urn:cgi:classifier:CGI:SimpleLithology:200811:
                         acidic_igneous_material</ogc:Literal>
          </ogc:PropertyIsEqualTo>
        </ogc:Filter>
        <sld:PolygonSymbolizer>
          <sld:Fill>
            <sld:CssParameter name="fill">#FFCCB3</sld:CssParameter>
          </sld:Fill>
        </sld:PolygonSymbolizer>
      </sld:Rule>
      <sld:Rule>
        <sld:Name>acidic igneous rock</sld:Name>
        <sld:Abstract>Igneous rock with more than 63 percent SiO2.
                    (after LeMaitre et al. 2002)
        </sld:Abstract>
        <ogc:Filter>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>gsml:specification/gsml:GeologicUnit/gsml:composition/
                 gsml:CompositionPart/gsml:lithology/@xlink:href</ogc:PropertyName>
            <ogc:Literal>urn:cgi:classifier:CGI:SimpleLithology:200811:
                         acidic_igneous_rock</ogc:Literal>
            </ogc:PropertyIsEqualTo>
        </ogc:Filter>
        <sld:PolygonSymbolizer>
          <sld:Fill>
            <sld:CssParameter name="fill">#FECDB2</sld:CssParameter>
          </sld:Fill>
        </sld:PolygonSymbolizer>
      </sld:Rule>
      ...
    </sld:FeatureTypeStyle>
  </sld:UserStyle>
 </sld:NamedLayer>
</sld:StyledLayerDescriptor>
```

9.15.3 GetFeatureInfo

Read *GetFeatureInfo* for general information on the GetFeatureInfo request. Read the tutorial on *GetFeatureInfo Templates* for information on how to template the html output. If you want to store a separate standard template for complex feature collections, save it under the filename `complex_content.ftl` in the template directory.

Read the tutorial on *Freemarker Templates* for more information on how to use the freemarker templates. Freemarker templates support recursive calls, which can be useful for templating complex content. For example, the following freemarker template creates a table of features with a column for each property, and will create another table inside each cell that contains a feature as property:

```
<#--
Macro's used for content
-->

<#macro property node>
    <#if !node.isGeometry>
      <#if node.isComplex>
      <td> <@feature node=node.rawValue type=node.type /> </td>
      <#else>
      <td>${node.value?string}</td>
      </#if>
    </#if>
</#macro>

<#macro header typenode>
<caption class="featureInfo">${typenode.name}</caption>
  <tr>
  <th>fid</th>
<#list typenode.attributes as attribute>
  <#if !attribute.isGeometry>
    <#if attribute.prefix == "">
        <th >${attribute.name}</th>
    <#else>
        <th >${attribute.prefix}:${attribute.name}</th>
    </#if>
  </#if>
</#list>
  </tr>
</#macro>

<#macro feature node type>
<table class="featureInfo">
  <@header typenode=type />
  <tr>
  <td>${node.fid}</td>
  <#list node.attributes as attribute>
      <@property node=attribute />
  </#list>
  </tr>
</table>
</#macro>

<#--
Body section of the GetFeatureInfo template, it's provided with one feature collection, and
will be called multiple times if there are various feature collections
-->
<table class="featureInfo">
  <@header typenode=type />

<#assign odd = false>
<#list features as feature>
  <#if odd>
    <tr class="odd">
  <#else>
    <tr>
  </#if>
  <#assign odd = !odd>
```

```
  <td>${feature.fid}</td>
  <#list feature.attributes as attribute>
    <@property node=attribute />
  </#list>
  </tr>
</#list>
</table>
<br/>
```

9.16 WFS 2.0 Support

9.16.1 Resolving

Local resolve is supported in app-schema. This can be done by setting the 'resolve' parameter to either 'local' or 'all'. (Remote Resolving is not supported.) The parameter 'resolveDepth' specifies how many levels of references will be resolved. The parameter 'resolveTimeOut' may be used to specify, in seconds, an upper limit to how long app-schema should search for the feature needed for resolving. If the time out limit is reached, the feature is not resolved.

When resolving without Feature Chaining (see below), a GML ID is extracted from the x-link reference and a brute force is done on all feature types to find a feature with this GML ID. The extraction of this GML ID from the Xlink Reference is done using the following rules:

- In case of a URN: The GML ID comes after last colon in the URN. Make sure that the *full* GML ID is included after the last colon (including a possible feature type prefix).
- In case of a URL: The GML ID comes after the # symbol.

Failing to respect one of these rules will result in failure of resolve.

Resolving and Feature Chaining By Reference

The 'resolve' and 'resolveDepth' parameters may also be used in the case of *Multi-valued properties by reference (xlink:href)*. In this case, no brute force will take place, but resolving will instruct App-Schema to do full feature chaining rather than inserting a reference. The URI will not be used to find the feature, but the feature chaining parameters specified in the mapping, as with normal feature chaining. Because of this, the parameter 'resolveTimeOut' will be ignored in this case.

However, be aware that every feature can only appear once in a response. If resolving would break this rule, for example with circular references, the encoder will change the resolved feature back to an (internal) x-link reference.

9.16.2 GetPropertyValue

The GetPropertyValue request is now fully supported. Resolving is also possible in this request, following the same rules as described above.

9.16.3 Paging

Paging is currently not supported in App-Schema yet. The parameter `count` is however supported (identical to the WFS 1.0 and 1.1 `maxFeatures` parameter).

9.17 Joining Support For Performance

App-schema joining is a optional configuration parameter that tells app-schema to use a different implementation for *Feature Chaining*, which in many cases can improve performance considerably, by reducing the amount of SQL queries sent to the DBMS.

9.17.1 Conditions

In order to use App-schema Joining, the following configuration conditions must be met:

- All feature mappings used must be mapped to JDBC datastores.
- All feature mappings that are chained to each other must map to the same physical database.
- In your mappings, there are restrictions on the CQL expressions specified in the <SourceExpression> of both the referencing field in the parent feature as well as the referenced field in the nested feature (like FEATURE_LINK). Any operators or functions used in this expression must be supported by the filter capibilities, i.e. geotools must be able to translate them directly to SQL code. This can be different for each DBMS, though as a general rule it can assumed that comparison operators, logical operators and arithmetic operators are all supported but functions are not. Using simple field names for feature chaining is guaranteed to always work.

Failing to comply with any of these three restrictions when turning on Joining will result in exceptions thrown at run-time.

When using app-schema with Joining turned on, the following restrictions exist with respect to normal behaviour:

- XPaths specified inside Filters do not support handling referenced features (see *Multi-valued properties by reference (xlink:href)*) as if they were actual nested features, i.e. XPaths can only be evaluated when they can be evaluated against the actual XML code produced by WFS according to the XPath standard.

9.17.2 Configuration

Joining is turned on by default. It is disabled by adding this simple line to your app-schema.properties file (see *Property Interpolation*)

```
app-schema.joining = false
```

Or, alternatively, by setting the value of the Java System Property "app-schema.joining" to "false", for example

```
java -DGEOSERVER_DATA_DIR=... -Dapp-schema.joining=false Start
```

Not specifying "app-schema.joining" parameter will enable joining by default.

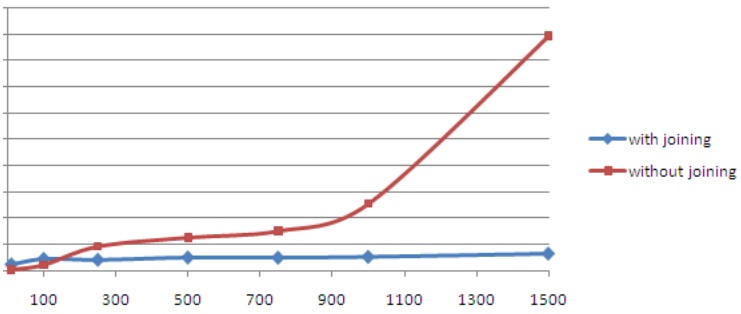
9.17.3 Database Design Guidelines

- Databases should be optimised for fast on-the-fly joining and ordering.
- Make sure to put indexes on all fields used as identifiers and for feature chaining, unique indexes where possible. Lack of indices may result in data being encoded in the wrong order or corrupted output when feature chaining is involved.
- Map your features preferably to normalised tables.

- It is recommended to apply feature chaining to regular one-to-many relationships, i.e. there should be a unique constraint defined on one of the fields used for the chaining, and if possible a foreign key constraint defined on the other field.

9.17.4 Effects on Performance

Typical curves of response time for configurations with and without joining against the amount of features produced will be shaped like this:



In the default implementation, response time increases rapidly with respect to the amount of produced features. This is because feature chaining is implemented by sending multiple SQL requests to the DBMS per feature, so the amount of requests increases with the amount of features produced. When Joining is turned on, response time will be almost constant with respect to the number of features. This is because in this implementation a small amount of larger queries is sent to the DBMS, independant of the amount of features produced. In summary, difference in performance becomes greater as the amount of features requested gets bigger. General performance of joining will be dependant on database and mapping design (see above) and database size.

Using joining is strongly recommended when a large number of features need to be produced, for example when producing maps with WMS (see *WMS Support*).

Optimising the performance of the database will maximise the benefit of using joining, including for small queries.

9.18 Tutorial

This tutorial demonstrates how to configure two complex feature types using the app-schema plugin and data from two property files.

9.18.1 GeoSciML

This example uses Geoscience Markup Language (GeoSciML) 2.0, a GML application schema:

> *"GeoSciML is an application schema that specifies a set of feature-types and supporting structures for information used in the solid-earth geosciences."*

The tutorial defines two feature types:

1. `gsml:GeologicUnit`, which describes "a body of material in the Earth".
2. `gsml:MappedFeature`, which describes the representation on a map of a feature, in this case `gsml:GeologicUnit`.

Because a single gsml:GeologicUnit can be observed at several distinct locations on the Earth's surface, it can have a multivalued gsml:occurrence property, each being a gsml:MappedFeature.

9.18.2 Installation

- Install GeoServer as usual.
- Install the app-schema plugin geoserver-*-app-schema-plugin.zip:
  - Place the jar files in WEB-INF/lib.
  - The tutorial folder contains the GeoServer configuraration (data directory) used for this tutorial.
    * Either replace your existing data directory with the tutorial data directory,
    * Or edit WEB-INF/web.xml to set GEOSERVER_DATA_DIR to point to the tutorial data directory. (Be sure to uncomment the section that sets GEOSERVER_DATA_DIR.)
- Perform any configuration required by your servlet container, and then start the servlet. For example, if you are using Tomcat, configure a new context in server.xml and then restart Tomcat.
- The first time GeoServer starts with the tutorial configuration, it will download all the schema (XSD) files it needs and store them in the app-schema-cache folder in the data directory. **You must be connected to the internet for this to work.**

9.18.3 datastore.xml

Each data store configuration file datastore.xml specifies the location of a mapping file and triggers its loading as an app-schema data source. This file should not be confused with the source data store, which is specified inside the mapping file.

For gsml_GeologicUnit the file is workspaces/gsml/gsml_GeologicUnit/datastore.xml:

```
<dataStore>
    <id>gsml_GeologicUnit_datastore</id>
    <name>gsml_GeologicUnit</name>
    <enabled>true</enabled>
    <workspace>
        <id>gsml_workspace</id>
    </workspace>
    <connectionParameters>
        <entry key="namespace">urn:cgi:xmlns:CGI:GeoSciML:2.0</entry>
        <entry key="url">file:workspaces/gsml/gsml_GeologicUnit/gsml_GeologicUnit.xml</entry>
        <entry key="dbtype">app-schema</entry>
    </connectionParameters>
</dataStore>
```

For gsml:MappedFeature the file is workspaces/gsml/gsml_MappedFeature/datastore.xml:

```
<dataStore>
    <id>gsml_MappedFeature_datastore</id>
    <name>gsml_MappedFeature</name>
    <enabled>true</enabled>
    <workspace>
        <id>gsml_workspace</id>
    </workspace>
    <connectionParameters>
        <entry key="namespace">urn:cgi:xmlns:CGI:GeoSciML:2.0</entry>
```

```
        <entry key="url">file:workspaces/gsml/gsml_MappedFeature/gsml_MappedFeature.xml</entry>
        <entry key="dbtype">app-schema</entry>
    </connectionParameters>
</dataStore>
```

Note: Ensure that there is no whitespace inside an `entry` element.

9.18.4 Mapping files

Configuration of app-schema feature types is performed in mapping files:

- `workspaces/gsml/gsml_GeologicUnit/gsml_GeologicUnit.xml`
- `workspaces/gsml/gsml_MappedFeature/gsml_MappedFeature.xml`

Namespaces

Each mapping file contains namespace prefix definitions:

```
<Namespace>
    <prefix>gml</prefix>
    <uri>http://www.opengis.net/gml</uri>
</Namespace>
<Namespace>
    <prefix>gsml</prefix>
    <uri>urn:cgi:xmlns:CGI:GeoSciML:2.0</uri>
</Namespace>
<Namespace>
    <prefix>xlink</prefix>
    <uri>http://www.w3.org/1999/xlink</uri>
</Namespace>
```

Only those namespace prefixes used in the mapping file need to be declared, so the mapping file for `gsml:GeologicUnit` has less.

Source data store

The data for this tutorial is contained in two property files:

- `workspaces/gsml/gsml_GeologicUnit/gsml_GeologicUnit.properties`
- `workspaces/gsml/gsml_MappedFeature/gsml_MappedFeature.properties`

Java Properties describes the format of property files.

For this example, each feature type uses an identical source data store configuration. This `directory` parameter indicates that the source data is contained in property files named by their feature type, in the same directory as the corresponding mapping file:

```
<sourceDataStores>
     <DataStore>
         <id>datastore</id>
         <parameters>
             <Parameter>
                 <name>directory</name>
                 <value>file:./</value>
```

```
            </Parameter>
         </parameters>
      </DataStore>
 </sourceDataStores>
```

See *Data Stores* for a description of how to use other types of data stores such as databases.

Target types

Both feature types are defined by the same XML Schema, the top-level schema for GeoSciML 2.0. This is specified in the `targetTypes` section. The type of the output feature is defined in `targetElement` in the `typeMapping` section below:

```
<targetTypes>
    <FeatureType>
        <schemaUri>http://www.geosciml.org/geosciml/2.0/xsd/geosciml.xsd</schemaUri>
    </FeatureType>
</targetTypes>
```

In this case the schema is published, but because the OASIS XML Catalog is used for schema resolution, a private or modified schema in the catalog can be used if desired.

Mappings

The `typeMappings` element begins with configuration elements. From the mapping file for `gsml:GeologicUnit`:

```
<typeMappings>
    <FeatureTypeMapping>
        <sourceDataStore>datastore</sourceDataStore>
        <sourceType>gsml_GeologicUnit</sourceType>
        <targetElement>gsml:GeologicUnit</targetElement>
```

- The mapping starts with `sourceDataStore`, which gives the arbitrary identifier used above to name the source of the input data in the `sourceDataStores` section.
- `sourceType` gives the name of the source simple feature type. In this case it is the simple feature type `gsml_GeologicUnit`, sourced from the rows of the file `gsml_GeologicUnit.properties` in the same directory as the mapping file.
- When working with databases `sourceType` is the name of a table or view. Database identifiers must be lowercase for PostGIS or uppercase for Oracle Spatial.
- `targetElement` is the name of the output complex feature type.

gml:id mapping

The first mapping sets the `gml:id` to be the feature id specified in the source property file:

```
<AttributeMapping>
    <targetAttribute>
        gsml:GeologicUnit
    </targetAttribute>
    <idExpression>
        <OCQL>ID</OCQL>
```

```
    </idExpression>
</AttributeMapping>
```

- `targetAttribute` is the XPath to the element for which the mapping applies, in this case, the top-level feature type.
- `idExpression` is a special form that can only be used to set the `gml:id` on a feature. Any field or CQL expression can be used, if it evaluates to an NCName.

Ordinary mapping

Most mappings consist of a target and source. Here is one from `gsml:GeologicUnit`:

```
<AttributeMapping>
    <targetAttribute>
        gml:description
        </targetAttribute>
    <sourceExpression>
        <OCQL>DESCRIPTION</OCQL>
    </sourceExpression>
</AttributeMapping>
```

- In this case, the value of `gml:description` is just the value of the `DESCRIPTION` field in the property file.
- For a database, the field name is the name of the column (the table/view is set in `sourceType` above). Database identifiers must be lowercase for PostGIS or uppercase for Oracle Spatial.
- CQL expressions can be used to calculate content. Use caution because queries on CQL-calculated values prevent the construction of efficient SQL queries.
- Source expressions can be CQL literals, which are single-quoted.

Client properties

In addition to the element content, a mapping can set one or more "client properties" (XML attributes). Here is one from `gsml:MappedFeature`:

```
<AttributeMapping>
    <targetAttribute>
        gsml:specification
    </targetAttribute>
    <ClientProperty>
        <name>xlink:href</name>
        <value>GU_URN</value>
    </ClientProperty>
</AttributeMapping>
```

- This mapping leaves the content of the `gsml:specification` element empty but sets an `xlink:href` attribute to the value of the `GU_URN` field.
- Multiple `ClientProperty` mappings can be set.

In this example from the mapping for `gsml:GeologicUnit` both element content and an XML attribute are provided:

```
<AttributeMapping>
    <targetAttribute>
        gml:name[1]
        </targetAttribute>
    <sourceExpression>
        <OCQL>NAME</OCQL>
    </sourceExpression>
    <ClientProperty>
        <name>codeSpace</name>
        <value>'urn:x-test:classifierScheme:TestAuthority:GeologicUnitName'</value>
    </ClientProperty>
</AttributeMapping>
```

- The `codespace` XML attribute is set to a fixed value by providing a CQL literal.
- There are multiple mappings for `gml:name`, and the index `[1]` means that this mapping targets the first.

targetAttributeNode

If the type of a property is abstract, a `targetAttributeNode` mapping must be used to specify a concrete type. This mapping must occur before the mapping for the content of the property.

Here is an example from the mapping file for `gsml:MappedFeature`:

```
<AttributeMapping>
    <targetAttribute>gsml:positionalAccuracy</targetAttribute>
    <targetAttributeNode>gsml:CGI_TermValuePropertyType</targetAttributeNode>
</AttributeMapping>
<AttributeMapping>
    <targetAttribute>gsml:positionalAccuracy/gsml:CGI_TermValue/gsml:value</targetAttribute>
    <sourceExpression>
        <OCQL>'urn:ogc:def:nil:OGC:missing'</OCQL>
    </sourceExpression>
    <ClientProperty>
        <name>codeSpace</name>
        <value>'urn:ietf:rfc:2141'</value>
    </ClientProperty>
</AttributeMapping>
```

- `gsml:positionalAccuracy` is of type `gsml:CGI_TermValuePropertyType`, which is abstract, so must be mapped to its concrete subtype `gsml:CGI_TermValuePropertyType` with a `targetAttributeNode` mapping before its contents can be mapped.
- This example also demonstrates that mapping can be applied to nested properties to arbitrary depth. This becomes unmanageable for deep nesting, where feature chaining is preferred.

Feature chaining

In feature chaining, one feature type is used as a property of an enclosing feature type, by value or by reference:

```
<AttributeMapping>
    <targetAttribute>
        gsml:occurrence
    </targetAttribute>
    <sourceExpression>
```

```
        <OCQL>URN</OCQL>
        <linkElement>gsml:MappedFeature</linkElement>
        <linkField>gml:name[2]</linkField>
    </sourceExpression>
    <isMultiple>true</isMultiple>
</AttributeMapping>
```

- In this case from the mapping for `gsml:GeologicUnit`, we specify a mapping for its `gsml:occurrence`.
- The `URN` field of the source `gsml_GeologicUnit` simple feature is use as the "foreign key", which maps to the second `gml:name` in each `gsml:MappedFeature`.
- Every `gsml:MappedFeature` with `gml:name[2]` equal to the `URN` of the `gsml:GeologicUnit` under construction is included as a `gsml:occurrence` property of the `gsml:GeologicUnit` (by value).

9.18.5 WFS response

When GeoServer is running, test app-schema WFS in a web browser. If GeoServer is listening on `localhost:8080` you can query the two feature types using these links:

- http://localhost:8080/geoserver/wfs?request=GetFeature&version=1.1.0&typeName=gsml:GeologicUnit
- http://localhost:8080/geoserver/wfs?request=GetFeature&version=1.1.0&typeName=gsml:MappedFeature

You can also obtain WFS responses by using the *Demo requests* page in the GeoServer web interface. (Note that the web interface does not yet support app-schema store or layer administration.)

- http://localhost:8080/geoserver/web/?wicket:bookmarkablePage=:org.geoserver.web.demo.DemoRequestsPage

gsml:GeologicUnit

Feature chaining has been used to construct the multivalued property `gsml:occurrence` of `gsml:GeologicUnit`. This property is a `gsml:MappedFeature`. The WFS response for `gsml:GeologicUnit` combines the output of both feature types into a single response. The first `gsml:GeologicUnit` has two `gsml:occurrence` properties, while the second has one. The relationships between the feature instances are data driven.

Because the mapping files in the tutorial configuration do not contain attribute mappings for all mandatory properties of these feature types, the WFS response is not *schema-valid* against the GeoSciML 2.0 schemas. Schema-validity can be achieved by adding more attribute mappings to the mapping files.

9.18.6 Acknowledgements

`gsml_GeologicUnit.properties` and `gsml_MappedFeature.properties` are derived from data provided by the Department of Primary Industries, Victoria, Australia. For the purposes of this tutorial, this data has been modified to the extent that it has no real-world meaning.

CHAPTER 10

Working with Cascaded Services

This section discusses how GeoServer can proxy external OGC services. This is known as **cascading** services.

GeoServer supports cascading the following services:

10.1 External Web Feature Server

GeoServer has the ability to load data from a remote Web Feature Server (WFS). This is useful if the remote WFS lacks certain functionality that GeoServer contains. For example, if the remote WFS is not also a Web Map Server (WMS), data from the WFS can be cascaded through GeoServer to utilize GeoServer's WMS. If the remote WFS has a WMS but that WMS cannot output KML, data can be cascaded through GeoServer's WMS to output KML.

10.1.1 Adding an external WFS

To connect to an external WFS, it is necessary to load it as a new datastore. To start, navigate to *Stores* → *Add a new store* → *Web Feature Server*.

Option	Description
Workspace	Name of the workspace to contain the store. This will also be the prefix of all of the layer names created from the store.
Data Source Name	Name of the store as known to GeoServer.
Description	Description of the store.
Enabled	Enables the store. If disabled, no data from the external WFS will be served.
GET_CAPABILITIES_URL	URL to access the capabilities document of the remote WFS.
PROTOCOL	When checked, connects with POST, otherwise uses GET.
USERNAME	The user name to connect to the external WFS.
PASSWORD	The password associated with the above user name.
ENCODING	The character encoding of the XML requests sent to the server. Defaults to `UTF-8`.
TIMEOUT	Time (in milliseconds) before timing out. Default is `3000`.
BUFFER_SIZE	Specifies a buffer size (in number of features). Default is `10` features.
TRY_GZIP	Specifies that the server should transfer data using compressed HTTP if supported by the server.
LENIENT	When checked, will try to render features that don't match the appropriate schema. Errors will be logged.
MAXFEATURES	Maximum amount of features to retrieve for each featuretype. Default is no limit.

When finished, click *Save*.

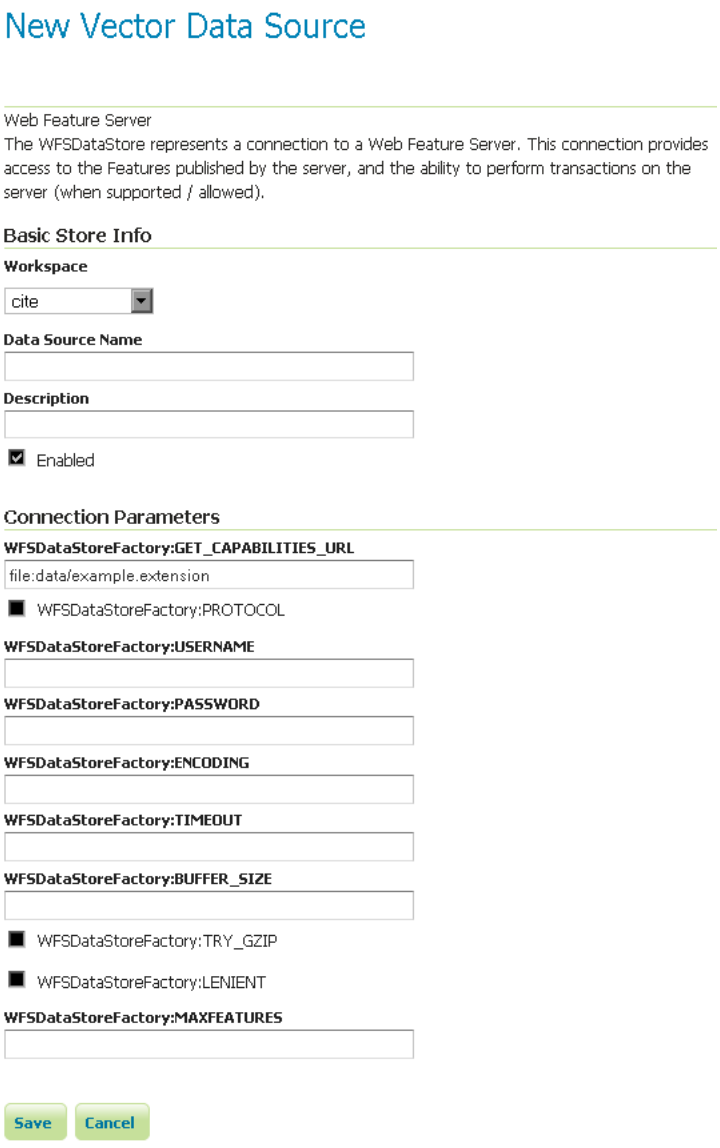


Figure 10.1: *Adding an external WFS as a store*

10.1.2 Configuring external WFS layers

When properly loaded, all layers served by the external WFS will be available to GeoServer. Before they can be served, however, they will need to be individually configured as new layers. See the section on *Layers* for how to add and edit new layers.

10.1.3 Connecting to an external WFS layer via a proxy server

In a corporate environment it may be necessary to connect to an external WFS through a proxy server. To achieve this, various java variables need to be set.

For a Windows install running Geoserver as a service, this is done by modifying the wrapper.conf file. For a default Windows install, modify `C:\Program Files\GeoServer x.x.x\wrapper\wrapper.conf` similarly to the following.

```
# Java Additional Parameters

wrapper.java.additional.1=-Djetty.home=. wrapper.java.additional.2=-
DGEOSERVER_DATA_DIR="%GEOSERVER_DATA_DIR%" wrapper.java.additional.3=-
Dhttp.proxySet=true wrapper.java.additional.4=-Dhttp.proxyHost=maitproxy
wrapper.java.additional.5=-Dhttp.proxyPort=8080 wrapper.java.additional.6=-
Dhttps.proxyHost=maitproxy wrapper.java.additional.7=-Dhttps.proxyPort=8080
wrapper.java.additional.8=-Dhttp.nonProxyHosts="mait*|dpi*|localhost"
```

Note that the **http.proxySet=true** parameter is required. Also, the parameter numbers must be consecutive - ie. no gaps.

For a Windows install not running Geoserver as a service, modify `startup.bat` so that the **java** command runs with similar -D parameters.

For a Linux/UNIX install, modify `startup.sh` so that the **java** command runs with similar -D parameters.

10.2 External Web Map Server

GeoServer has the ability to proxy a remote Web Map Service (WMS). This process is sometimes known as **Cascading WMS**. Loading a remote WMS is useful for many reasons. If you don't manage or have access to the remote WMS, you can now manage its output as if it were local. Even if the remote WMS is not GeoServer, you can use GeoServer features to treat its output (watermarking, decoration, printing, etc).

To access a remote WMS, it is necessary to load it as a store in GeoServer. GeoServer must be able to access the capabilities document of the remote WMS for the store to be successfully loaded.

10.2.1 Adding an external WMS

To connect to an external WMS, it is necessary to load it as a new store. To start, in the *Web Administration Interface*, navigate to *Stores* → *Add a new store* → *WMS*. The option is listed under *Other Data Sources*.

Figure 10.2: *Adding an external WMS as a store*

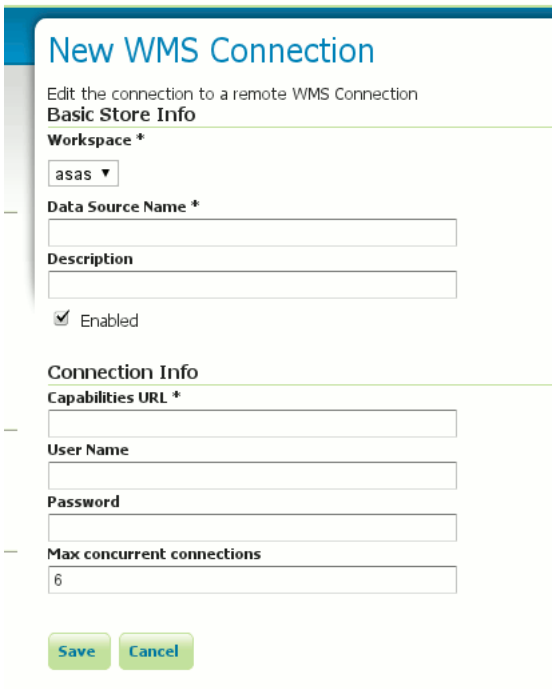


Figure 10.3: *Configuring a new external WMS store*

Option	Description
Workspace	Name of the workspace to contain the store. This will also be the prefix of all of the layer names published from the store. **The workspace name on the remote WMS is not cascaded.**
Data Source Name	Name of the store as known to GeoServer.
Description	Description of the store.
Enabled	Enables the store. If disabled, no data from the remote WMS will be served.
Capabilities URL	The full URL to access the capabilities document of the remote WMS.
User Name	If the WMS requires authentication, the user name to connect as.
Password	If the WMS requires authentication, the password to connect with.
Max concurrent connections	The maximum number of persistent connections to keep for this WMS.

When finished, click *Save*.

10.2.2 Configuring external WMS layers

When properly loaded, all layers served by the external WMS will be available to GeoServer. Before they can be served, however, they will need to be individually configured (published) as new layers. See the section on *Layers* for how to add and edit new layers. Once published, these layers will show up in the *Layer Preview* and as part of the WMS capabilities document.

10.2.3 Features

Connecting a remote WMS allows for the following features:

- **Dynamic reprojection**. While the default projection for a layer is cascaded, it is possible to pass the SRS parameter through to the remote WMS. Should that SRS not be valid on the remote server, GeoServer will dynamically reproject the images sent to it from the remote WMS.
- **GetFeatureInfo**. WMS GetFeatureInfo requests will be passed to the remote WMS. If the remote WMS supports the `application/vnd.ogc.gml` format the request will be successful.
- Full **REST Configuration**. See the *REST configuration* section for more information about the GeoServer REST interface.

10.2.4 Limitations

Layers served through an external WMS have some, but not all of the functionality of a local WMS.

- Layers cannot be styled with SLD.
- Alternate (local) styles cannot be used.
- Extra request parameters (`time`, `elevation`, `cql_filter`, etc.) cannot be used.
- GetLegendGraphic requests aren't supported.
- Image format cannot be specified. GeoServer will attempt to request PNG images, and if that fails will use the remote server's default image format.
- Authentication for the remote WMS isn't supported. The remote WMS must be unsecured.

CHAPTER 11

Filtering in GeoServer

Filtering allows selecting features that satisfy a specific set of conditions. Filters can be used in several contexts in GeoServer:

- in WMS requests, to select which features should be displayed on a map
- in WFS requests, to specify the features to be returned
- in SLD documents, to apply different symbolization to features on a thematic map.

11.1 Supported filter languages

Data filtering in GeoServer is based on the concepts found in the OGC Filter Encoding Specification.

GeoServer accepts filters encoded in two different languages: *Filter Encoding* and *Common Query Language*.

11.1.1 Filter Encoding

The **Filter Encoding** language is an XML-based method for defining filters. XML Filters can be used in the following places in GeoServer:

- in WMS `GetMap` requests, using the `filter` parameter
- in WFS `GetFeature` requests, using the `filter` parameter
- in SLD Rules, in the *Filter* element

The Filter Encoding language is defined in the following OGC specifications:

- OGC Filter encoding specification v 1.0, used in WFS 1.0 and SLD 1.0
- OGC Filter encoding specification v 1.1, used in WFS 1.1

11.1.2 CQL/ECQL

CQL (Common Query Language) is a plain-text language created for the *OGC Catalog* specification. GeoServer has adapted it to be an easy-to-use filtering mechanism. GeoServer actually implements a more powerful extension called **ECQL (Extended CQL)**, which allows expressing the full range of filters that *OGC Filter 1.1* can encode. ECQL is accepted in many places in GeoServer:

- in WMS `GetMap` requests, using the *cql_filter* parameter
- in WFS `GetFeature` requests, using the *cql_filter* parameter

- in SLD *dynamic symbolizers*

The *ECQL Reference* describes the features of the ECQL language. The *CQL and ECQL* tutorial shows examples of defining filters.

The CQL and ECQL languages are defined in:

- OpenGIS Catalog Services Specification contains the standard definition of CQL
- ECQL Grammar is the grammar defining the GeoTools ECQL implementation

11.2 Filter Encoding Reference

This is a reference for the **Filter Encoding** language implemented in GeoServer. The Filter Encoding language uses an XML-based syntax. It is defined by the OGC Filter Encoding standard.

Filters are used to select features or other objects from the context in which they are evaluated. They are similar in functionality to the SQL "WHERE" clause. A filter is specified using a **condition**.

11.2.1 Condition

A condition is a single *Predicate* element, or a combination of conditions by *Logical operators*.

11.2.2 Predicate

Predicates are boolean-valued expressions which compute relationships between values. A predicate is specified by using a **comparison operator** or a **spatial operator**. The operators are used to compare properties of the features being filtered to other feature properties or to literal data.

Comparison operators

Comparison operators are used to specify conditions on non-spatial attributes.

Binary Comparison operators

The **binary comparison operators** are:

- `<PropertyIsEqualTo>`
- `<PropertyIsNotEqualTo>`
- `<PropertyIsLessThan>`
- `<PropertyIsLessThanOrEqualTo>`
- `<PropertyIsGreaterThan>`
- `<PropertyIsGreaterThanOrEqualTo>`

They contain the elements:

Element	Required?	Description
Expression	Yes	The first value to compare. Often a `<PropertyName>`.
Expression	Yes	The second value to compare

Binary comparison operator elements may include an optional `matchCase` attribute, with the value `true` or `false`. If this attribute is `true` (the default), string comparisons are case-sensitive. If the attribute is `false` strings comparisons do not check case.

PropertyIsLike operator

The `<PropertyIsLike>` operator matches a string property value against a text **pattern**. It contains the elements:

Element	Required?	Description
`<PropertyName>`	Yes	Contains a string specifying the name of the property to test
`<Literal>`	Yes	Contains a pattern string to be matched

The pattern is specified by a sequence of regular characters and three special pattern characters. The pattern characters are defined by the following *required* attributes of the `<PropertyIsLike>` element:

- `wildCard` specifies the pattern character which matches any sequence of zero or more string characters
- `singleChar` specifies the pattern character which matches any single string character
- `escapeChar` specifies the escape character which can be used to escape the pattern characters

PropertyIsNull operator

The `<PropertyIsNull>` operator tests whether a property value is null. It contains the element:

Element	Required?	Description
`<PropertyName>`	Yes	contains a string specifying the name of the property to be tested

PropertyIsBetweeen operator

The `<PropertyIsBetween>` operator tests whether an expression value lies within a range given by a lower and upper bound (inclusive). It contains the elements:

Element	Required?	Description
Expression	Yes	The value to test
`<LowerBoundary>`	Yes	Contains an *Expression* giving the lower bound of the range
`<UpperBoundary>`	Yes	Contains an *Expression* giving the upper bound of the range

Spatial operators

Spatial operators are used to specify conditions on the geometric attributes of a feature. The following spatial operators are available:

Topological operators

These operators test topological spatial relationships using the standard OGC Simple Features predicates:

- `<Intersects>` - Tests whether two geometries intersect
- `<Disjoint>` - Tests whether two geometries are disjoint
- `<Contains>` - Tests whether a geometry contains another one

- `<Within>` - Tests whether a geometry is within another one
- `<Touches>` - Tests whether two geometries touch
- `<Crosses>` - Tests whether two geometries cross
- `<Overlaps>` - Tests whether two geometries overlap
- `<Equals>` - Tests whether two geometries are topologically equal

These contains the elements:

Element	Required?	Description
`<PropertyName>`	Yes	Contains a string specifying the name of the geometry-valued property to be tested.
GML Geometry	Yes	A GML literal value specifying the geometry to test against

Distance operators

These operators test distance relationships between a geometry property and a geometry literal:

- `<DWithin>`
- `<Beyond>`

They contain the elements:

Element	Required?	Description
`<PropertyName>`	Yes	Contains a string specifying the name of the property to be tested. If omitted, the *default geometry attribute* is assumed.
GML Geometry	Yes	A literal value specifying a geometry to compute the distance to. This may be either a geometry or an envelope in GML 3 format
`<Distance>`	Yes	Contains the numeric value for the distance tolerance. The element may include an optional `units` attribute.

Bounding Box operator

The `<BBOX>` operator tests whether a geometry-valued property intersects a fixed bounding box. It contains the elements:

Element	Required?	Description
`<PropertyName>`	No	Contains a string specifying the name of the property to be tested. If omitted, the *default geometry attribute* is assumed.
`<gml:Box>`	Yes	A GML Box literal value specifying the bounding box to test against

Examples

- This filter selects features with a geometry that intersects the point (1,1).

```
<Intersects>
  <PropertyName>GEOMETRY</PropertyName>
  <gml:Point>
    <gml:coordinates>1 1</gml:coordinates>
```

```
  </gml:Point>
</Intersects>
```

- This filter selects features with a geometry that overlaps a polygon.

```
<Overlaps>
  <PropertyName>Geometry</PropertyName>
  <gml:Polygon srsName="http://www.opengis.net/gml/srs/epsg.xml#63266405">
    <gml:outerBoundaryIs>
      <gml:LinearRing>
         <gml:posList> ... </gml:posList>
      </gml:LinearRing>
    </gml:outerBoundaryIs>
  </gml:Polygon>
 </Overlaps>
```

- This filter selects features with a geometry that intersects the geographic extent [-10,0 : 10,10].

```
<BBOX>
  <PropertyName>GEOMETRY</PropertyName>
  <gml:Box srsName="urn:x-ogc:def:crs:EPSG:4326">
    <gml:coord>
      <gml:X>-10</gml:X> <gml:Y>0</gml:Y>
    </gml:coord>
    <gml:coord>
      <gml:X>10</gml:X> <gml:Y>10</gml:Y>
    </gml:coord>
  </gml:Box>
</BBOX>
```

11.2.3 Logical operators

Logical operators are used to specify logical combinations of *Condition* elements (which may be either *Predicate* elements or other **logical operators**). They may be nested to any depth.

The following logical operators are available:

- `<And>` - computes the logical conjunction of the operands
- `<Or>` - computes the logical disjunction of the operands

The content for `<And>` and `<Or>` is two operands given by *Condition* elements.

- `<Not>` - computes the logical negation of the operand

The content for `<Not>` is a single operand given by a *Condition* element.

Examples

- This filter uses `<And>` to combine a comparison predicate and a spatial predicate:

```
<And>
   <PropertyIsEqualTo>
      <PropertyName>NAME</PropertyName>
      <Literal>New York</Literal>
   </PropertyIsEqualTo>
   <Intersects>
      <PropertyName>GEOMETRY</PropertyName>
      <Literal>
```

```
        <gml:Point>
            <gml:coordinates>1 1</gml:coordinates>
        </gml:Point>
      </Literal>
   </Intersects>
</And>
```

11.2.4 Expression

Filter expressions specify constant, variable or computed data values. An expression is formed from one of the following elements (some of which contain sub-expressions, meaning that expressions may be of arbitrary depth):

Arithmetic operators

The **arithmetic operator** elements compute arithmetic operations on numeric values.

- `<Add>` - adds the two operands
- `<Sub>` - subtracts the second operand from the first
- `<Mul>` - multiplies the two operands
- `<Div>` - divides the first operand by the second

Each arithmetic operator element contains two *Expression* elements providing the operands.

Function

The `<Function>` element specifies a filter function to be evaluated. The required `name` attribute gives the function name. The element contains a sequence of zero or more *Expression* elements providing the values of the function arguments.

See the *Filter Function Reference* for details of the functions provided by GeoServer.

Property Value

The `<PropertyName>` element refers to the value of a feature attribute. It contains a **string** or an **XPath expression** specifying the attribute name.

Literal

The `<Literal>` element specifies a constant value. It contains data of one of the following types:

Type	Description
Numeric	A string representing a numeric value (integer or decimal).
Boolean	A boolean value of `true` or `false`.
String	A string value. XML-incompatible text may be included by using **character entities** or `<![CDATA[ ]]>` delimiters.
Date	A string representing a date.
Geometry	An element specifying a geometry in GML3 format.

11.3 ECQL Reference

This section provides a reference for the syntax of the ECQL language. The full language grammar is documented in the the GeoTools ECQL BNF definition

11.3.1 Syntax Notes

The sections below describe the major language constructs. Each construct lists all syntax options for it. Each option is defined as a sequence of other constructs, or recursively in terms of itself.

- Symbols which are part of the ECQL language are shown in `code font`. All other symbols are part of the grammar description.
- ECQL keywords are not case-sensitive.
- A vertical bar symbol '|' indicates that a choice of keyword can be made.
- Brackets '[...]' delimit syntax that is optional.
- Braces '{ ... }' delimit syntax that may be present zero or more times.

11.3.2 Condition

A filter condition is a single predicate, or a logical combination of other conditions.

Syntax	Description
Predicate	Single predicate expression
Condition `AND` | `OR` *Condition*	Conjunction or disjunction of conditions
`NOT` *Condition*	Negation of a condition
`(` | `[` *Condition* `]` | `)`	Bracketing with `(` or `[` controls evaluation order

11.3.3 Predicate

Predicates are boolean-valued expressions which specify relationships between values.

Syntax	Description
Expression `=` | `<>` | `<` | `<=` | `>` | `>=` *Expression*	Comparison operations
Expression [`NOT`] `BETWEEN` *Expression* `AND` *Expression*	Tests whether a value lies in or outside a range (inclusive)
Expression [`NOT`] `LIKE` | `ILIKE` *like-pattern*	Simple pattern matching. *like-pattern* uses the `%` character as a wild-card for any number of characters. `ILIKE` does case-insensitive matching.
Expression [`NOT`] `IN (` *Expression* { `,` *Expression* } `)`	Tests whether an expression value is (not) in a set of values
Expression `IN (` *Literal* { `,` *Literal* } `)`	Tests whether a feature ID value is in a given set. ID values are integers or string literals
Expression `IS` [`NOT`] `NULL`	Tests whether a value is (non-)null
Attribute `EXISTS` | `DOES-NOT-EXIST`	Tests whether a featuretype does (not) have a given attribute
`INCLUDE` | `EXCLUDE`	Always include (exclude) features to which this filter is applied

Temporal Predicate

Temporal predicates specify the relationship of a time-valued expression to a time or time period.

Syntax	Description
Expression `BEFORE` *Time*	Tests whether a time value is before a point in time
Expression `BEFORE OR DURING` *Time Period*	Tests whether a time value is before or during a time period
Expression `DURING` *Time Period*	Tests whether a time value is during a time period
Expression `DURING OR AFTER` *Time Period*	Tests whether a time value is during or after a time period
Expression `AFTER` *Time*	Tests whether a time value is after a point in time

Spatial Predicate

Spatial predicates specify the relationship between geometric values. Topological spatial predicates (`INTERSECTS`, `DISJOINT`, `CONTAINS`, `WITHIN`, `TOUCHES CROSSES`, `OVERLAPS` and `RELATE`) are defined in terms of the DE-9IM model described in the OGC Simple Features for SQL specification.

Syntax	Description
`INTERSECTS(`*Expression* `,` *Expression* `)`	Tests whether two geometries intersect. The converse of `DISJOINT`
`DISJOINT(`*Expression* `,` *Expression* `)`	Tests whether two geometries are disjoint. The converse of `INTERSECTS`
`CONTAINS(`*Expression* `,` *Expression* `)`	Tests whether the first geometry topologically contains the second. The converse of `WITHIN`
`WITHIN(`*Expression* `,` *Expression* `)`	Tests whether the first geometry is topologically within the second. The converse of `CONTAINS`
`TOUCHES(`*Expression* `,` *Expression* `)`	Tests whether two geometries touch. Geometries touch if they have at least one point in common, but their interiors do not intersect.
`CROSSES(`*Expression* `,` *Expression* `)`	Tests whether two geometries cross. Geometries cross if they have some but not all interior points in common
`OVERLAPS(`*Expression* `,` *Expression* `)`	Tests whether two geometries overlap. Geometries overlap if they have the same dimension, have at least one point each not shared by the other, and the intersection of the interiors of the two geometries has the same dimension as the geometries themselves
`EQUALS(`*Expression* `,` *Expression* `)`	Tests whether two geometries are topologically equal
`RELATE(` *Expression* `,` *Expression* `,` *pattern* `)`	Tests whether geometries have the spatial relationship specified by a DE-9IM matrix *pattern*. A DE-9IM pattern is a string of length 9 specified using the characters `*TF012`. Example: `'1*T***T**'`
`DWITHIN(` *Expression* `,` *Expression* `,` *distance* `,` *units* `)`	Tests whether the distance between two geometries is no more than the specified distance. *distance* is an unsigned numeric value for the distance tolerance. *units* is one of `feet`, `meters`, `statute miles`, `nautical miles`, `kilometers`
`BEYOND(` *Expression* `,` *Expression* `,` *distance* `,` *units* `)`	Similar to `DWITHIN`, but tests whether the distance between two geometries is greater than the given distance.
`BBOX (` *Expression* `,` *Number* `,` *Number* `,` *Number* `,` *Number* [`,` *CRS*] `)`	Tests whether a geometry intersects a bounding box specified by its minimum and maximum X and Y values. The optional *CRS* is a string containing an SRS code (For example, `'EPSG:1234'`. The default is to use the CRS of the queried layer)
`BBOX (` *Expression* `,` *Expression* | *Geometry* `)`	Tests whether a geometry intersects a bounding box specified by a geometric value computed by a function or provided by a geometry literal.

11.3.4 Expression

An expression specifies a attribute, literal, or computed value. The type of the value is determined by the nature of the expression. The standard PEMDAS order of evaluation is used.

Syntax	Description
Attribute	Name of a feature attribute
Literal	Literal value
Expression + | − | * | / *Expression*	Arithmetic operations
function ([*Expression* { , *Expression* }])	Value computed by evaluation of a *filter function* with zero or more arguments.
(| [*Expression*] |)	Bracketing with (or [controls evaluation order

11.3.5 Attribute

An attribute name denotes the value of a feature attribute.

- Simple attribute names are sequences of letters and numbers,
- Attribute names quoted with double-quotes may be any sequence of characters.

11.3.6 Literal

Literals specify constant values of various types.

Type	Description
Number	Integer or floating-point number. Scientific notation is supported.
Boolean	`TRUE` or `FALSE`
String	String literal delimited by single quotes. To include a single quote in the string use two single-quotes: `''`
Geometry	Geometry in WKT format. WKT is defined in the OGC Simple Features for SQL specification. All standard geometry types are supported: `POINT`, `LINESTRING`, `POLYGON`, `MULTIPOINT`, `MULTILINESTRING`, `MULTIPOLYGON`, `GEOMETRYCOLLECTION`. A custom type of Envelope is also supported with syntax `ENVELOPE` (*x1 x2 y1 y2*).
Time	A UTC date/time value in the format `yyyy-mm-hhThh:mm:ss`. The seconds value may have a decimal fraction. The time zone may be specified as `Z` or `+/-hh:mm`. Example: `2006-11-30T00:30:00Z`
Duration	A time duration specified as `P` [y `Y` m `M` d `D`] `T` [h `H` m `M` s `S`]. The duration can be specified to any desired precision by including only the required year, month, day, hour, minute and second components. Examples: `P1Y2M`, `P4Y2M20D`, `P4Y2M1DT20H3M36S`

Time Period

Specifies a period of time, in several different formats.

Syntax	Description
Time / *Time*	Period specified by a start and end time
Duration / *Time*	Period specified by a duration before a given time
Time / *Duration*	Period specified by a duration after a given time

11.4 Filter functions

The OGC Filter Encoding specification provides a generic concept of a *filter function*. A filter function is a named function with any number of arguments, which can be used in a filter expression to perform specific calculations. This provides much richer expressiveness for defining filters. Filter functions can be used in both the XML Filter Encoding language and the textual ECQL language, using the syntax appropriate to the language.

GeoServer provides many different kinds of filter functions, covering a wide range of functionality including mathematics, string formatting, and geometric operations. A complete list is provided in the *Filter Function Reference*.

Note: The Filter Encoding specification provides a standard syntax for filter functions, but does not mandate a specific set of functions. Servers are free to provide whatever functions they want, so some function expressions may work only in specific software.

11.4.1 Examples

The following examples show how filter functions are used. The first shows enhanced WFS filtering using the `geometryType` function. The second shows how to use functions in SLD to get improved label rendering.

WFS filtering

Let's assume we have a feature type whose geometry field, `geom`, can contain any kind of geometry. For a certain application we need to extract only the features whose geometry is a simple point or a multipoint. This can be done using a GeoServer-specific filter function named `geometryType`. Here is the WFS request including the filter function:

```
<wfs:GetFeature service="WFS" version="1.0.0"
  outputFormat="GML2"
  xmlns:wfs="http://www.opengis.net/wfs"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.opengis.net/wfs
                      http://schemas.opengis.net/wfs/1.0.0/WFS-basic.xsd">
  <wfs:Query typeName="sf:archsites">
    <ogc:Filter>
       <ogc:PropertyIsEqualTo>
          <ogc:Function name="geometryType">
             <ogc:PropertyName>geom</ogc:PropertyName>
          </ogc:Function>
          <ogc:Literal>Point</ogc:Literal>
       </ogc:PropertyIsEqualTo>
    </ogc:Filter>
    </wfs:Query>
</wfs:GetFeature>
```

SLD formatting

We want to display elevation labels in a contour map. The elevations are stored as floating point values, so the raw numeric values may display with unwanted decimal places (such as "150.0" or "149.999999").

We want to ensure the numbers are rounded appropriately (i.e. to display "150"). To achieve this the `numberFormat` filter function can be used in the SLD label content expression:

```
...
<TextSymbolizer>
  <Label>
    <ogc:Function name="numberFormat">
      <ogc:Literal>##</ogc:Literal>
      <ogc:PropertyName>ELEVATION</ogc:PropertyName>
    </ogc:Function>
  </Label>
  ...
</TextSymbolizer>
...
```

11.4.2 Performance implications

Using filter functions in SLD symbolizer expressions does not have significant overhead, unless the function is performing very heavy computation.

However, using functions in WFS filtering or SLD rule expressions may cause performance issues in certain cases. This is usually because specific filter functions are not recognized by a native data store filter encoder, and thus GeoServer must execute the functions in memory instead.

For example, given a filter like `BBOX(geom,-10,30,20,45) and geometryType(geom) = 'Point'` most data stores will split the filter into two separate parts. The bounding box filter will be encoded as a primary filter and executed in SQL, while the `geometryType` function will be executed in memory on the results coming from the primary filter.

11.5 Filter Function Reference

This reference describes all filter functions that can be used in WFS/WMS filtering or in SLD expressions.

The list of functions available on a Geoserver instance can be determined by browsing to http://localhost:8080/geoserver/wfs?request=GetCapabilities and searching for `ogc:FunctionNames` in the returned XML. If a function is described in the Capabilities document but is not in this reference, then it might mean that the function cannot be used for filtering, or that it is new and has not been documented. Ask for details on the user mailing list.

Unless otherwise specified, none of the filter functions in this reference are understood natively by the data stores, and thus expressions using them will be evaluated in-memory.

11.5.1 Function argument type reference

Type	Description
Double	Floating point number, 8 bytes, IEEE 754. Ranges from 4.94065645841246544e-324d to 1.79769313486231570e+308d
Float	Floating point number, 4 bytes, IEEE 754. Ranges from 1.40129846432481707e-45 to 3.40282346638528860e+38. Smaller range and less accurate than Double.
Integer	Integer number, ranging from -2,147,483,648 to 2,147,483,647
Long	Integer number, ranging from -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807
Number	A numeric value of any type
Object	A value of any type
String	A sequence of characters
Timestamp	Date and time information

11.5.2 Comparison functions

Name	Arguments	Description
between	`num`:Number, `low`:Number, `high`:Number	returns true if `low` <= `num` <= `high`
equalTo	`a`:Object, `b`:Object	Can be used to compare for equality two numbers, two strings, two dates, and so on
greaterEqualThan	`x`:Object, `y`:Object	Returns true if `x` >= `y`. Parameters can be either numbers or strings (in the second case lexicographic ordering is used)
greaterThan	`x`:Object, `y`:Object	Returns true if `x` > `y`. Parameters can be either numbers or strings (in the second case lexicographic ordering is used)
in2, in3, in4, in5, in6, in7, in8, in9, in10	`candidate`:Object, `v1`:Object, ..., `v9`:Object	Returns true if `candidate` is equal to one of the `v1`, ..., `v9` values. Use the function name matching the number of arguments specified.
isLike	`string`:String, `pattern`:String	Returns true if the string matches the specified pattern. For the full syntax of the pattern specification see the Java Pattern class javadocs
isNull	`obj`:Object	Returns true the passed parameter is `null`, false otherwise
lessThan	`x`:Object, `y`:Object	Returns true if `x` < `y`. Parameters can be either numbers or strings (in the second case lexicographic ordering is used
lessEqualThan	`x`:Object, `y`:Object	Returns true if `x` <= `y`. Parameters can be either numbers or strings (in the second case lexicographic ordering is used
not	`bool`:Boolean	Returns the negation of `bool`
notEqual	`x`:Object, `y`:Object	Returns true if `x` and `y` are equal, false otherwise

11.5.3 Control functions

Name	Arguments	Description
if_then_else	`condition`:Boolean, `x`:Object, `y`: Object	Returns `x` if the condition is true, `y` otherwise

11.5.4 Environment function

This function returns the value of environment variables defined in various contexts. Contexts which define environment variables include *SLD rendering* and the *WMS Animator*.

Name	Arguments	Description
env	`variable`:String	Returns the value of the environment variable `variable`.

11.5.5 Feature functions

Name	Arguments	Description
id	`feature`:Feature	returns the identifier of the feature
PropertyExists	`f`:Feature, `propertyName`:String	Returns `true` if `f` has a property named `propertyName`
property	`f`:Feature, `propertyName`:String	Returns the value of the property `propertyName`. Allows property names to be computed or specified by *Variable substitution in SLD*.

11.5.6 Spatial Relationship functions

For more information about the precise meaning of the spatial relationships consult the OGC Simple Feature Specification for SQL

Name	Arguments	Description
contains	`a`:Geometry, `b`:Geometry	Returns true if the geometry `a` contains `b`
crosses	`a`:Geometry, `b`:Geometry	Returns true if `a` crosses `b`
disjoint	`a`:Geometry, `b`:Geometry	Returns true if the two geometries are disjoint, false otherwise
equalsExact	`a`:Geometry, `b`:Geometry	Returns true if the two geometries are exactly equal, same coordinates in the same order
equalsExactTolerance	`a`:Geometry, `b`:Geometry, `tol`:Double	Returns true if the two geometries are exactly equal, same coordinates in the same order, allowing for a `tol` distance in the corresponding points
intersects	`a`:Geometry, `b`:Geometry	Returns true if `a` intersects `b`
isWithinDistance	`a`: Geometry, `b`:Geometry, `distance`: Double	Returns true if the distance between `a` and `b` is less than `distance` (measured as an euclidean distance)
overlaps	`a`: Geometry, `b`:Geometry	Returns true `a` overlaps with `b`
relate	`a`: Geometry, `b`:Geometry	Returns the DE-9IM intersection matrix for `a` and `b`
relatePattern	`a`: Geometry, `b`:Geometry, `pattern`:String	Returns true if the DE-9IM intersection matrix for `a` and `b` matches the specified pattern
touches	`a`: Geometry, `b`: Geometry	Returns true if `a` touches `b` according to the SQL simple feature specification rules
within	`a`: Geometry, `b`:Geometry	Returns true is fully contained inside `b`

11.5.7 Geometric functions

Name	Arguments	Description
area	geometry:Geometry	The area of the specified geometry
boundary	geometry:Geometry	Returns the boundary of a geomet
boundaryDimension	geometry:Geometry	Returns the number of dimension
buffer	geometry:Geometry, distance:Double	Returns the buffered area around
bufferWithSegments	geometry:Geometry, distance:Double, segments:Integer	Returns the buffered area around
centroid	geometry:Geometry	Returns the centroid of the geome
convexHull	geometry:Geometry	Returns the convex hull of the spe
difference	a:Geometry, b:Geometry	Returns all the points that sit in a
dimension	a:Geometry	Returns the dimension of the spec
distance	a:Geometry, b:Geometry	Returns the euclidean distance bet
endAngle	line:LineString	Returns the angle of the end segm
endPoint	line:LineString	Returns the end point of the linest
envelope	geometry:geometry	Returns the polygon representing
exteriorRing	poly:Polygon	Returns the exterior ring of the sp
geometryType	geometry:Geometry	Returns the type of the geometry a
geomFromWKT	wkt:String	Returns the Geometry represente
geomLength	geometry:Geometry	Returns the length/perimeter of th
getGeometryN	collection:GeometryCollection, n:Integer	Returns the n-th geometry inside t
getX	p:Point	Returns the x ordinate of p
getY	p:Point	Returns the y ordinate of p
getZ	p:Point	Returns the z ordinate of p
interiorPoint	geometry:Geometry	Returns a point that is either interi
interiorRingN	polyg:Polygon, n:Integer	Returns the n-th interior ring of th
intersection	a:Geometry, b:Geometry	Returns the intersection between a
isClosed	line: LineString	Returns true if line forms a close
isEmpty	geometry:Geometry	Returns true if the geometry does
isometric	geometry:Geometry, extrusion:Double	Returns a MultiPolygon containin
isRing	line:LineString	Returns true if the line is actually
isSimple	line:LineString	Returns true if the geometry self i
isValid	geometry: Geometry	Returns true if the geometry is top
numGeometries	collection: GeometryCollection	Returns the number of geometries
numInteriorRing	poly: Polygon	Returns the number of interior rin
numPoint	geometry: Geometry	Returns the number of points (ver
offset	geometry: Geometry, offsetX:Double, offsetY:Double	Offsets all points in a geometry by
pointN	geometry: Geometry, n:Integer	Returns the n-th point inside the s
startAngle	line: LineString	Returns the angle of the starting se
startPoint	line: LineString	Returns the starting point of the in
symDifference	a: Geometry, b:Geometry	Returns the symmetrical differenc
toWKT	geometry: Geometry	Returns the WKT representation o
union	a: Geometry, b:Geometry	Returns the union of a and b (the
vertices	geom: Geometry	Returns a multi-point made with a

11.5.8 Math functions

Name	Arguments	Description
abs	`value`:Integer	The absolute value of the specified Integer `value`
abs_2	`value`:Long	The absolute value of the specified Long `value`
abs_3	`value`:Float	The absolute value of the specified Float `value`
abs_4	`value`:Double	The absolute value of the specified Double `value`
acos	`angle`:Double	Returns the arc cosine of an `angle` in radians, in the range of 0.0 through `PI`
asin	`angle`:Double	Returns the arc sine of an `angle` in radians, in the range of `-PI / 2` through `PI / 2`
atan	`angle`:Double	Returns the arc tangent of an angle in radians, in the range of `-PI/2` through `PI/2`
atan2	`x`:Double, `y`:Double	Converts a rectangular coordinate `(x, y)` to polar **(r, theta)** and returns **theta**.
ceil	`x`: Double	Returns the smallest (closest to negative infinity) double value that is greater than or equal to `x` and is equal to a mathematical integer.
cos	`angle`: Double	Returns the cosine of an `angle` expressed in radians
double2bool	`x`: Double	Returns `true` if `x` is zero, `false` otherwise
exp	`x`: Double	Returns Euler's number **e** raised to the power of `x`
floor	`x`: Double	Returns the largest (closest to positive infinity) value that is less than or equal to `x` and is equal to a mathematical integer
IEEERemainder	`x`: Double, `y`:Double	Computes the remainder of `x` divided by `y` as prescribed by the IEEE 754 standard
int2bbool	`x`: Integer	Returns true if `x` is zero, false otherwise
int2ddouble	`x`: Integer	Converts `x` to a Double
log	`x`: Integer	Returns the natural logarithm (base `e`) of `x`
max, max_3, max_4	`x1`: Double, `x2`:Double, `x3`:Double, `x4`:Double	Returns the maximum between `x1`, ..., `x4`
min, min_3, min_4	`x1`: Double, `x2`:Double, `x3`:Double, `x4`:Double	Returns the minimum between `x1`, ..., `x4`
pi	None	Returns an approximation of `pi`, the ratio of the circumference of a circle to its diameter
pow	`base`:Double, `exponent`:Double	Returns the value of `base` raised to the power of `exponent`
random	None	Returns a Double value with a positive sign, greater than or equal to `0.0` and less than `1.0`. Returned values are chosen pseudo-randomly with (approximately) uniform distribution from that range.
rint	`x`:Double	Returns the Double value that is closest in value to the argument and is equal to a mathematical integer. If two double values that are mathematical integers are equally close, the result is the integer value that is even.
round_2	`x`:Double	Same as `round`, but returns a Long
round	`x`:Double	Returns the closest Integer to `x`. The result is rounded to an integer by adding 1/2, taking the floor of the result, and casting the result to type Integer. In other words, the result is equal to the value of the expression `(int)floor(a + 0.5)`
roundDouble	`x`:Double	Returns the closest Long to `x`
tan	`angle`:Double	Returns the trigonometric tangent of `angle`
toDegrees	`angle`:Double	Converts an angle expressed in radians into degrees
toRadians	`angle`:Double	Converts an angle expressed in radians into degrees

11.5.9 String functions

String functions generally will accept any type of value for `String` arguments. Non-string values will be converted into a string representation automatically.

Name	Arguments	Description
Concatenate	`s1`:String, `s2`:String, ...	Concatenates any number of strings. Non-string arguments are allowed.
strCapitalize	`sentence`:String	Fully capitalizes the sentence. For example, "HoW aRe YOU?" will be turned into "How Are You?"
strConcat	`a`:String, `b`:String	Concatenates the two strings into one
strEndsWith	`string`:String, `suffix`:String	Returns true if `string` ends with `suffix`
strEqualsIgnoreCase	`a`:String, `b`:String	Returns true if the two strings are equal ignoring case considerations
strIndexOf	`string`:String, `substring`:String	Returns the index within this string of the first occurrence of the specified substring, or `-1` if not found
strLastIndexOf	`string`:String, `substring`:String	Returns the index within this string of the last occurrence of the specified substring, or `-1` if not found
strLength	`string`:String	Returns the string length
strMatches	`string`:String, `pattern`:String	Returns true if the string matches the specified regular expression. For the full syntax of the pattern specification see the Java Pattern class javadocs
strReplace	`string`:String, `pattern`:String, `replacement`:String, `global`: boolean	Returns the string with the pattern replaced with the given replacement text. If the `global` argument is `true` then all occurrences of the pattern will be replaced, otherwise only the first. For the full syntax of the pattern specification see the Java Pattern class javadocs
strStartsWith	`string`:String, `prefix`:String	Returns true if `string` starts with `prefix`
strSubstring	`string`:String, `begin`:Integer, `end`:Integer	Returns a new string that is a substring of this string. The substring begins at the specified `begin` and extends to the character at index `endIndex - 1` (indexes are zero-based).
strSubstringStart	`string`:String, `begin`:Integer	Returns a new string that is a substring of this string. The substring begins at the specified `begin` and extends to the last character of the string
strToLowerCase	`string`:String	Returns the lower case version of the string
strToUpperCase	`string`:String	Returns the upper case version of the string
strTrim	`string`:String	Returns a copy of the string, with leading and trailing white space omitted

11.5.10 Parsing and formatting functions

Name	Arguments	Description
dateFormat	`date`:Timestamp, `format`:String	Formats the specified date according to the provided format. The format syntax can be found in the Java SimpleDateFormat javadocs
dateParse	`dateString`:String, `format`:String	Parses a date from a `dateString` formatted according to the `format` specification. The format syntax can be found in the Java SimpleDateFormat javadocs
numberFormat	`number`:Double, `format`:String	Formats the number according to the specified `format`. The format syntax can be found in the Java DecimalFormat javadocs
parseBoolean	`boolean`:String	Parses a string into a boolean. The empty string, `f`, `0.0` and `0` are considered false, everything else is considered true.
parseDouble	`number`:String	Parses a string into a double. The number can be expressed in normal or scientific form.
parseInt	`number`:String	Parses a string into an integer.
parseLong	`number`:String	Parses a string into a long integer

11.5.11 Transformation functions

Transformation functions transform values from one data space into another. These functions provide a concise way to compute styling parameters from feature attribute values. See also *Styling using Transformation Functions*.

Name	Arguments	Description
Recode	`lookupValue`:Object, `data`:Object, `value`:Object, ...	Transforms a `lookupValue` from a set of discrete data values into another set of values. Any number of `data`/`value` pairs may be specified.
Categorize	`lookupValue`:Object, `value`:Object, `threshold`:Object, ... `value`:Object, `belongsTo` : String	Transforms a continuous-valued attribute value into a set of discrete values. `lookupValue` and `value` must be an orderable type (typically numeric). The initial `value` is required. Any number of additional `threshold`/`value` pairs may be specified. `belongsTo` is optional, with the value `succeeding` or `preceding`. It defines which interval to use when the lookup value equals a threshold value.
Interpolate	`lookupValue`:Numeric, `data`:Numeric, `value`:Numeric *or* #RRGGBB, ... `mode`:String, `method`:String	Transforms a continuous-valued attribute value into another continuous range of values. Any number of `data`/`value` pairs may be specified. `mode` is optional, with the value `linear`, `cosine` or `cubic`. It defines the interpolation algorithm to use. `method` is optional, with the value `numeric` or `color`. It defines whether the target values are numeric or RGB color specifications.

CHAPTER 12

Styling

This section discusses the styling of geospatial data served through GeoServer.

12.1 Introduction to SLD

Geospatial data has no intrinsic visual component. In order to see data, it must be styled. Styling specifies color, thickness, and other visible attributes used to render data on a map.

In GeoServer, styling is accomplished using a markup language called Styled Layer Descriptor, or SLD for short. SLD is an XML-based markup language and is very powerful, although somewhat complex. This page gives an introduction to the capabilities of SLD and how it works within GeoServer.

Note: Since GeoServer uses SLD exclusively for styling, the terms "SLD" and "style" will often be used interchangeably.

12.1.1 SLD Concepts

In GeoServer styling is most often specified using XML **SLD style documents**. Style documents are associated with GeoServer **layers** (**featuretypes**) to specify how they should be **rendered**. A style document specifies a single **named layer** and a **user style** for it. The layer and style can have metadata elements such as a **name** identifying them, a **title** for displaying them, and an **abstract** describing them in detail. Within the top-level style are one or more **feature type styles**, which act as "virtual layers" to provide control over rendering order (allowing styling effects such as cased lines for roads). Each feature type style contains one or more **rules**, which control how styling is applied based on feature attributes and zoom level. Rules select applicable features by using **filters**, which are logical conditions containing **predicates**, **expressions** and **filter functions**. To specify the details of styling for individual features, rules contain any number of **symbolizers**. Symbolizers specify styling for **points**, **lines** and **polygons**, as well as **rasters** and **text labels**.

For more information refer to the *SLD Reference*.

12.1.2 Types of styling

Vector data that GeoServer can serve consists of three classes of shapes: **Points, lines, and polygons**. Lines (one dimensional shapes) are the simplest, as they have only the edge to style (also known as "stroke"). Polygons, two dimensional shapes, have an edge and an inside (also known as a "fill"), both of which can be styled differently. Points, even though they lack dimension, have both an edge and a fill (not to mention a size) that can be styled. For fills, color can be specified; for strokes, color and thickness can be specified.

GeoServer also serves raster data. This can be styled with a wide variety of control over color palette, opacity, contrast and other parameters.

More advanced styling is possible as well. Points can be specified with well-known shapes like circles, squares, stars, and even custom graphics or text. Lines can be styled with a dash styles and hashes. Polygons can be filled with a custom tiled graphics. Styling can be based on attributes in the data, so that certain features are styled differently. Text labels on features are possible as well. Styling can also be determined by zoom level, so that features are displayed in a way appropriate to their apparent size. The possibilities are vast.

12.1.3 A basic style example

A good way to learn about SLD is to study styling examples. The following is a simple SLD that can be applied to a layer that contains points, to style them as red circles with a size of 6 pixels. (This is the first example in the *Points* section of the *SLD Cookbook*.)

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <StyledLayerDescriptor version="1.0.0"
3      xsi:schemaLocation="http://www.opengis.net/sld StyledLayerDescriptor.xsd"
4      xmlns="http://www.opengis.net/sld"
5      xmlns:ogc="http://www.opengis.net/ogc"
6      xmlns:xlink="http://www.w3.org/1999/xlink"
7      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
8    <NamedLayer>
9      <Name>Simple point</Name>
10     <UserStyle>
11       <Title>GeoServer SLD Cook Book: Simple point</Title>
12       <FeatureTypeStyle>
13         <Rule>
14           <PointSymbolizer>
15             <Graphic>
16               <Mark>
17                 <WellKnownName>circle</WellKnownName>
18                 <Fill>
19                   <CssParameter name="fill">#FF0000</CssParameter>
20                 </Fill>
21               </Mark>
22               <Size>6</Size>
23             </Graphic>
24           </PointSymbolizer>
25         </Rule>
26       </FeatureTypeStyle>
27     </UserStyle>
28   </NamedLayer>
29 </StyledLayerDescriptor>
```

Although the example looks long, only a few lines are really important to understand. **Line 14** states that a "PointSymbolizer" is to be used to style data as points. **Lines 15-17** state that points are to be styled using a graphic shape specified by a "well known name", in this case a circle. SLD provides names for many shapes such as "square", "star", "triangle", etc. **Lines 18-20** specify the shape should be filled with a color of `#FF0000` (red). This is an RGB color code, written in hexadecimal, in the form of #RRGGBB. Finally, **line 22** specifies that the size of the shape is 6 pixels in width. The rest of the structure contains metadata about the style, such as a name identifying the style and a title for use in legends.

Note: In SLD documents some tags have prefixes, such as `ogc:`. This is because they are defined in **XML namespaces**. The top-level `StyledLayerDescriptor` tag (**lines 2-7**) specifies two XML namespaces, one called `xmlns`, and one called `xmlns:ogc`. The first namespace is the default for the document, so tags

belonging to it do not need a prefix. Tags belonging to the second require the prefix `ogc:`. In fact, the namespace prefixes can be any identifier. The first namespace could be called `xmlns:sld` (as it often is) and then all the tags in this example would require an `sld:` prefix. The key point is that tags need to have the prefix for the namespace they belong to.

See the *SLD Cookbook* for more examples of styling with SLD.

12.2 Working with SLD

This section describes how to create, view and troubleshoot SLD styling in GeoServer.

12.2.1 Creating

GeoServer comes with some basic styles defined in its catalog. Any number of new styles can be added to the catalog. Styles can also be specified **externally** to the server, either to define a complete map, or to extend the server style catalog using **library mode**.

Catalog Styles

Styles in the catalog can be viewed, edited and validated via the *Styles* menu of the *Web Administration Interface*. They may also be created and accessed via the REST *Styles* API.

Catalog styles consist of a *StyledLayerDescriptor* document containing a single `<NamedLayer>` element, which contains a single `<UserStyle>` element to specify the styling. The layer name is ignored, since the style may be applied to many different layers.

Every layer (featuretype) registered with GeoServer must have at least one catalog style associated with it, which is the default style for rendering the layer. Any number of additional styles can be associated with a layer. This allows layers to have appropriate styles advertised in the WMS `GetCapabilities` document. A layer's styles can be changed using the *Layers* page of the *Web Administration Interface*.

Note: When adding a layer and a style for it to GeoServer at the same time, the style should be added first, so that the new layer can be associated with the style immediately.

External Styles

Styling can be defined externally to the server in a number of ways:

- An internet-accessible SLD document can be provided via the `SLD=url` parameter in a WMS *GetMap* GET request
- An SLD document can be provided directly in a WMS *GetMap* GET request using the `SLD_BODY=style` parameter. The SLD XML must be URL-encoded.
- A *StyledLayerDescriptor* element can be included in a WMS `GetMap` POST request XML document.

In all of these cases, if the WMS `layers` parameter is not supplied then the map content is defined completely by the layers and styles present in the external SLD. If the `layers` parameter is present, then styling operates in *Library Mode*.

External styles can define new layers of styled data, by using the SLD *InlineFeature* element to provide feature data. This can be used to implement dynamic feature highlighting, for example.

External styling may be generated dynamically by client applications, This provides a powerful way for clients to control styling effects.

Library Mode

In **library mode** externally-defined styles are treated as a *style library*, which acts as an extension to the server style catalog. Library mode occurs when map layers and styles are specified using the `layers` and `styles` WMS parameters, and additional styling is supplied externally using one of the methods described in the previous section. The styles in the external style document take precedence over the catalog styles during rendering.

Style lookup in library mode operates as follows:

- For each layer in the `layers` list, the applied style is either a named style specified in the `styles` list (if present), or the layer default style
- For a **named** style, if the eternal style document has a `<NamedLayer>...<UserStyle>` with matching layer name and style name, then it is used. Otherwise, the style name is searched for in the catalog. If it is not found there, an error occurs.
- For a **default** style, the external style document is searched to find a `<NamedLayer>` element with the layer name. If it contains a `<UserStyle>` with the `<IsDefault>` element having the value `1` then that style is used. Otherwise, the default server style for the layer (which must exist) is used.

Generally it is simpler and more performant to use styles from the server catalog. However, library mode can be useful if it is required to style a map containing many layers and where only a few of them need to have their styling defined externally.

12.2.2 Viewing

Once a style has been associated with a layer, the resulting rendering of the layer data can be viewed by using the *Layer Preview*. The most convenient output format to use is the built-in OpenLayers viewer. Styles can be modified while the view is open, and their effect is visible as soon as the map view is panned or zoomed. Alternate styles can be viewed by specifying them in the `styles` WMS request parameter.

To view the effect of compositing multiple styled layers, several approaches are available:

- Create a **layer group** for the desired layers using the *Layer Groups* page, and preview it. Non-default styles can be specified for layers if required.
- Submit a WMS *GetMap* GET request specifying multiple layers in the `layers` parameter, and the corresponding styles in the `styles` parameter (if non-default styles are required).
- Submit a WMS `GetMap` POST request containing a *StyledLayerDescriptor* element specifying server layers, optional layers of inline data, and either named catalog styles or user-defined styling for each layer.

12.2.3 Troubleshooting

SLD is a type of programming language, not unlike creating a web page or building a script. As such, problems may arise that may require troubleshooting.

Syntax Errors

To minimize syntax errors when creating the SLD, it is recommended to use a text editor that is designed to work with XML (such as the *Style Editor* provided in the GeoServer UI). XML editors can make finding syntax errors easier by providing syntax highlighting and (sometimes) built-in error checking.

The GeoServer *Style Editor* allows validating a document against the SLD XML schema. This is not mandatory, but is recommended to do before saving styles.

Semantic Errors

Semantic errors cannot be caught by SLD validation, but show up when a style is applied during map rendering. Most of the time this will result in a map displaying no features (a blank map), but some errors will prevent the map from rendering at all.

The easiest way to fix semantic errors in an SLD is to try to isolate the error. If the SLD is long with many rules and filters, try temporarily removing some of them to see if the errors go away.

In some cases the server will produce a WMS Exception document which may help to identify the error. It is also worth checking the server log to see if any error messages have been recorded.

12.3 SLD Cookbook

The SLD Cookbook is a collection of SLD "recipes" for creating various types of map styles. Wherever possible, each example is designed to show off a single SLD feature so that code can be copied from the examples and adapted when creating SLDs of your own. While not an exhaustive reference like the *SLD Reference* or the OGC SLD 1.0 specification the SLD Cookbook is designed to be a practical reference, showing common style templates that are easy to understand.

The SLD Cookbook is divided into four sections: the first three for each of the vector types (points, lines, and polygons) and the fourth section for rasters. Each example in every section contains a screenshot showing actual GeoServer WMS output, a snippet of the SLD code for reference, and a link to download the full SLD.

Each section uses data created especially for the SLD Cookbook, with shapefiles for vector data and GeoTIFFs for raster data. The projection for data is EPSG:4326. All files can be easily loaded into GeoServer in order to recreate the examples.

Data Type	Shapefile
Point	`sld_cookbook_point.zip`
Line	`sld_cookbook_line.zip`
Polygon	`sld_cookbook_polygon.zip`
Raster	`sld_cookbook_raster.zip`

12.3.1 Points

While points are seemingly the simplest type of shape, possessing only position and no other dimensions, there are many different ways that a point can be styled in SLD.

Warning: The code examples shown on this page are **not the full SLD code**, as they omit the SLD header and footer information for the sake of brevity. Please use the links to download the full SLD for each example.

Example points layer

The `points layer` used for the examples below contains name and population information for the major cities of a fictional country. For reference, the attribute table for the points in this layer is included below.

fid (Feature ID)	**name** (City name)	**pop** (Population)
point.1	Borfin	157860
point.2	Supox City	578231
point.3	Ruckis	98159
point.4	Thisland	34879
point.5	Synopolis	24567
point.6	San Glissando	76024
point.7	Detrania	205609

`Download the points shapefile`

Simple point

This example specifies points be styled as red circles with a diameter of 6 pixels.

Figure 12.1: *Simple point*

Code

`View and download the full "Simple point" SLD`

```
<FeatureTypeStyle>
  <Rule>
    <PointSymbolizer>
      <Graphic>
        <Mark>
          <WellKnownName>circle</WellKnownName>
          <Fill>
            <CssParameter name="fill">#FF0000</CssParameter>
          </Fill>
        </Mark>
        <Size>6</Size>
      </Graphic>
    </PointSymbolizer>
```

```
14      </Rule>
15    </FeatureTypeStyle>
```

Details

There is one `<Rule>` in one `<FeatureTypeStyle>` for this SLD, which is the simplest possible situation. (All subsequent examples will contain one `<Rule>` and one `<FeatureTypeStyle>` unless otherwise specified.) Styling points is accomplished via the `<PointSymbolizer>` (**lines 3-13**). **Line 6** specifies the shape of the symbol to be a circle, with **line 8** determining the fill color to be red (`#FF0000`). **Line 11** sets the size (diameter) of the graphic to be 6 pixels.

Simple point with stroke

This example adds a stroke (or border) around the *Simple point*, with the stroke colored black and given a thickness of 2 pixels.

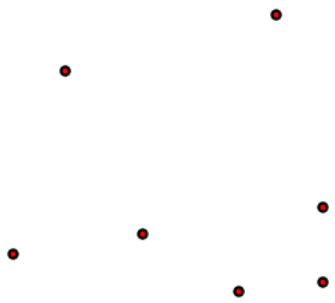


Figure 12.2: *Simple point with stroke*

Code

View and download the full "Simple point with stroke" SLD

```
1     <FeatureTypeStyle>
2       <Rule>
3         <PointSymbolizer>
4           <Graphic>
5             <Mark>
6               <WellKnownName>circle</WellKnownName>
7               <Fill>
8                 <CssParameter name="fill">#FF0000</CssParameter>
9               </Fill>
10              <Stroke>
11                <CssParameter name="stroke">#000000</CssParameter>
12                <CssParameter name="stroke-width">2</CssParameter>
13              </Stroke>
14            </Mark>
15            <Size>6</Size>
16          </Graphic>
```

```
      </PointSymbolizer>
    </Rule>
  </FeatureTypeStyle>
```

Details

This example is similar to the *Simple point* example. **Lines 10-13** specify the stroke, with **line 11** setting the color to black (`#000000`) and **line 12** setting the width to 2 pixels.

Rotated square

This example creates a square instead of a circle, colors it green, sizes it to 12 pixels, and rotates it by 45 degrees.

Figure 12.3: *Rotated square*

Code

View and download the full "Rotated square" SLD

```
  <FeatureTypeStyle>
    <Rule>
      <PointSymbolizer>
        <Graphic>
          <Mark>
            <WellKnownName>square</WellKnownName>
            <Fill>
              <CssParameter name="fill">#009900</CssParameter>
            </Fill>
          </Mark>
          <Size>12</Size>
          <Rotation>45</Rotation>
        </Graphic>
      </PointSymbolizer>
    </Rule>
  </FeatureTypeStyle>
```

Details

In this example, **line 6** sets the shape to be a square, with **line 8** setting the color to a dark green (`#009900`). **Line 11** sets the size of the square to be 12 pixels, and **line 12** set the rotation is to 45 degrees.

Transparent triangle

This example draws a triangle, creates a black stroke identical to the *Simple point with stroke* example, and sets the fill of the triangle to 20% opacity (mostly transparent).

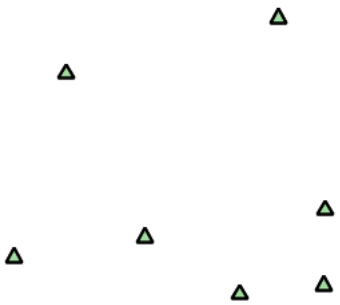


Figure 12.4: *Transparent triangle*

Code

View and download the full "Transparent triangle" SLD

```
1    <FeatureTypeStyle>
2      <Rule>
3        <PointSymbolizer>
4          <Graphic>
5            <Mark>
6              <WellKnownName>triangle</WellKnownName>
7              <Fill>
8                <CssParameter name="fill">#009900</CssParameter>
9                <CssParameter name="fill-opacity">0.2</CssParameter>
10             </Fill>
11             <Stroke>
12               <CssParameter name="stroke">#000000</CssParameter>
13               <CssParameter name="stroke-width">2</CssParameter>
14             </Stroke>
15           </Mark>
16           <Size>12</Size>
17         </Graphic>
18       </PointSymbolizer>
19     </Rule>
20   </FeatureTypeStyle>
```

Details

In this example, **line 6** once again sets the shape, in this case to a triangle. **Line 8** sets the fill color to a dark green (`#009900`) and **line 9** sets the opacity to 0.2 (20% opaque). An opacity value of 1 means that the shape is drawn 100% opaque, while an opacity value of 0 means that the shape is drawn 0% opaque, or completely transparent. The value of 0.2 (20% opaque) means that the fill of the points partially takes on the color and style of whatever is drawn beneath it. In this example, since the background is white, the dark green looks lighter. Were the points imposed on a dark background, the resulting color would be darker. **Lines 12-13** set the stroke color to black (`#000000`) and width to 2 pixels. Finally, **line 16** sets the size of the point to be 12 pixels in diameter.

Point as graphic

This example styles each point as a graphic instead of as a simple shape.

Figure 12.5: *Point as graphic*

Code

View and download the full "Point as graphic" SLD

```
1    <FeatureTypeStyle>
2      <Rule>
3        <PointSymbolizer>
4          <Graphic>
5            <ExternalGraphic>
6              <OnlineResource
7                xlink:type="simple"
8                xlink:href="smileyface.png" />
9              <Format>image/png</Format>
10           </ExternalGraphic>
11           <Size>32</Size>
12         </Graphic>
13       </PointSymbolizer>
14     </Rule>
15   </FeatureTypeStyle>
```

Details

This style uses a graphic instead of a simple shape to render the points. In SLD, this is known as an `<ExternalGraphic>`, to distinguish it from the commonly-used shapes such as squares and circles that are "internal" to the renderer. **Lines 5-10** specify the details of this graphic. **Line 8** sets the path and file name of the graphic, while **line 9** indicates the format (MIME type) of the graphic (image/png). In this example, the graphic is contained in the same directory as the SLD, so no path information is necessary in **line 8**, although a full URL could be used if desired. **Line 11** determines the size of the displayed graphic; this can be set independently of the dimensions of the graphic itself, although in this case they are the same (32 pixels). Should a graphic be rectangular, the `<Size>` value will apply to the *height* of the graphic only, with the width scaled proportionally.

Figure 12.6: *Graphic used for points*

Point with default label

This example shows a text label on the *Simple point* that displays the "name" attribute of the point. This is how a label will be displayed in the absence of any other customization.

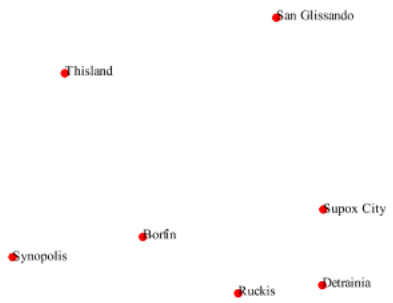


Figure 12.7: *Point with default label*

Code

View and download the full "Point with default label" SLD

```
1    <FeatureTypeStyle>
2      <Rule>
3        <PointSymbolizer>
4          <Graphic>
5            <Mark>
6              <WellKnownName>circle</WellKnownName>
7              <Fill>
8                <CssParameter name="fill">#FF0000</CssParameter>
9              </Fill>
10           </Mark>
```

```
11            <Size>6</Size>
12          </Graphic>
13        </PointSymbolizer>
14        <TextSymbolizer>
15          <Label>
16            <ogc:PropertyName>name</ogc:PropertyName>
17          </Label>
18          <Fill>
19            <CssParameter name="fill">#000000</CssParameter>
20          </Fill>
21        </TextSymbolizer>
22      </Rule>
23    </FeatureTypeStyle>
```

Details

Lines 3-13, which contain the `<PointSymbolizer>`, are identical to the *Simple point* example above. The label is set in the `<TextSymbolizer>` on **lines 14-27**. **Lines 15-17** determine what text to display in the label, which in this case is the value of the "name" attribute. (Refer to the attribute table in the *Example points layer* section if necessary.) **Line 19** sets the text color. All other details about the label are set to the renderer default, which here is Times New Roman font, font color black, and font size of 10 pixels. The bottom left of the label is aligned with the center of the point.

Point with styled label

This example improves the label style from the *Point with default label* example by centering the label above the point and providing a different font name and size.

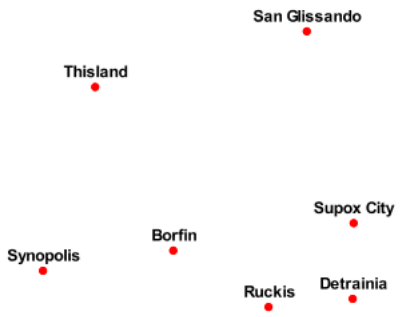


Figure 12.8: *Point with styled label*

Code

View and download the full "Point with styled label" SLD

```
1     <FeatureTypeStyle>
2       <Rule>
3         <PointSymbolizer>
4           <Graphic>
```

```
5             <Mark>
6               <WellKnownName>circle</WellKnownName>
7               <Fill>
8                 <CssParameter name="fill">#FF0000</CssParameter>
9               </Fill>
10            </Mark>
11            <Size>6</Size>
12          </Graphic>
13        </PointSymbolizer>
14        <TextSymbolizer>
15          <Label>
16            <ogc:PropertyName>name</ogc:PropertyName>
17          </Label>
18          <Font>
19            <CssParameter name="font-family">Arial</CssParameter>
20            <CssParameter name="font-size">12</CssParameter>
21            <CssParameter name="font-style">normal</CssParameter>
22            <CssParameter name="font-weight">bold</CssParameter>
23          </Font>
24          <LabelPlacement>
25            <PointPlacement>
26              <AnchorPoint>
27                <AnchorPointX>0.5</AnchorPointX>
28                <AnchorPointY>0.0</AnchorPointY>
29              </AnchorPoint>
30              <Displacement>
31                <DisplacementX>0</DisplacementX>
32                <DisplacementY>5</DisplacementY>
33              </Displacement>
34            </PointPlacement>
35          </LabelPlacement>
36          <Fill>
37            <CssParameter name="fill">#000000</CssParameter>
38          </Fill>
39        </TextSymbolizer>
40      </Rule>
41    </FeatureTypeStyle>
```

Details

In this example, **lines 3-13** are identical to the *Simple point* example above. The `<TextSymbolizer>` on lines 14-39 contains many more details about the label styling than the previous example, *Point with default label*. **Lines 15-17** once again specify the "name" attribute as text to display. **Lines 18-23** set the font information: **line 19** sets the font family to be "Arial", **line 20** sets the font size to 12, **line 21** sets the font style to "normal" (as opposed to "italic" or "oblique"), and **line 22** sets the font weight to "bold" (as opposed to "normal"). **Lines 24-35** (`<LabelPlacement>`) determine the placement of the label relative to the point. The `<AnchorPoint>` (**lines 26-29**) sets the point of intersection between the label and point, which here (**line 27-28**) sets the point to be centered (0.5) horizontally axis and bottom aligned (0.0) vertically with the label. There is also `<Displacement>` (**lines 30-33**), which sets the offset of the label relative to the line, which in this case is 0 pixels horizontally (**line 31**) and 5 pixels vertically (**line 32**). Finally, **line 37** sets the font color of the label to black (`#000000`).

The result is a centered bold label placed slightly above each point.

Point with rotated label

This example builds on the previous example, *Point with styled label*, by rotating the label by 45 degrees, positioning the labels farther away from the points, and changing the color of the label to purple.

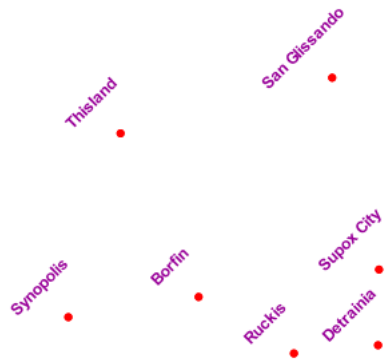


Figure 12.9: *Point with rotated label*

Code

View and download the full "Point with rotated label" SLD

```
<FeatureTypeStyle>
  <Rule>
    <PointSymbolizer>
      <Graphic>
        <Mark>
          <WellKnownName>circle</WellKnownName>
          <Fill>
            <CssParameter name="fill">#FF0000</CssParameter>
          </Fill>
        </Mark>
        <Size>6</Size>
      </Graphic>
    </PointSymbolizer>
    <TextSymbolizer>
      <Label>
        <ogc:PropertyName>name</ogc:PropertyName>
      </Label>
      <Font>
        <CssParameter name="font-family">Arial</CssParameter>
        <CssParameter name="font-size">12</CssParameter>
        <CssParameter name="font-style">normal</CssParameter>
        <CssParameter name="font-weight">bold</CssParameter>
      </Font>
      <LabelPlacement>
        <PointPlacement>
          <AnchorPoint>
            <AnchorPointX>0.5</AnchorPointX>
            <AnchorPointY>0.0</AnchorPointY>
          </AnchorPoint>
          <Displacement>
```

```
31              <DisplacementX>0</DisplacementX>
32              <DisplacementY>25</DisplacementY>
33            </Displacement>
34            <Rotation>-45</Rotation>
35          </PointPlacement>
36        </LabelPlacement>
37        <Fill>
38          <CssParameter name="fill">#990099</CssParameter>
39        </Fill>
40      </TextSymbolizer>
41    </Rule>
42  </FeatureTypeStyle>
```

Details

This example is similar to the *Point with styled label*, but there are three important differences. **Line 32** specifies 25 pixels of vertical displacement. **Line 34** specifies a rotation of "-45" or 45 degrees counter-clockwise. (Rotation values increase clockwise, which is why the value is negative.) Finally, **line 38** sets the font color to be a shade of purple (`#99099`).

Note that the displacement takes effect before the rotation during rendering, so in this example, the 25 pixel vertical displacement is itself rotated 45 degrees.

Attribute-based point

This example alters the size of the symbol based on the value of the population ("pop") attribute.

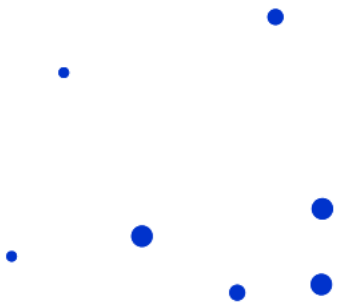


Figure 12.10: *Attribute-based point*

Code

View and download the full "Attribute-based point" SLD

```
1   <FeatureTypeStyle>
2     <Rule>
3       <Name>SmallPop</Name>
4       <Title>1 to 50000</Title>
5       <ogc:Filter>
```

```
      <ogc:PropertyIsLessThan>
        <ogc:PropertyName>pop</ogc:PropertyName>
        <ogc:Literal>50000</ogc:Literal>
      </ogc:PropertyIsLessThan>
    </ogc:Filter>
    <PointSymbolizer>
      <Graphic>
        <Mark>
          <WellKnownName>circle</WellKnownName>
          <Fill>
            <CssParameter name="fill">#0033CC</CssParameter>
          </Fill>
        </Mark>
        <Size>8</Size>
      </Graphic>
    </PointSymbolizer>
  </Rule>
  <Rule>
    <Name>MediumPop</Name>
    <Title>50000 to 100000</Title>
    <ogc:Filter>
      <ogc:And>
        <ogc:PropertyIsGreaterThanOrEqualTo>
          <ogc:PropertyName>pop</ogc:PropertyName>
          <ogc:Literal>50000</ogc:Literal>
        </ogc:PropertyIsGreaterThanOrEqualTo>
        <ogc:PropertyIsLessThan>
          <ogc:PropertyName>pop</ogc:PropertyName>
          <ogc:Literal>100000</ogc:Literal>
        </ogc:PropertyIsLessThan>
      </ogc:And>
    </ogc:Filter>
    <PointSymbolizer>
      <Graphic>
        <Mark>
          <WellKnownName>circle</WellKnownName>
          <Fill>
            <CssParameter name="fill">#0033CC</CssParameter>
          </Fill>
        </Mark>
        <Size>12</Size>
      </Graphic>
    </PointSymbolizer>
  </Rule>
  <Rule>
    <Name>LargePop</Name>
    <Title>Greater than 100000</Title>
    <ogc:Filter>
      <ogc:PropertyIsGreaterThanOrEqualTo>
        <ogc:PropertyName>pop</ogc:PropertyName>
        <ogc:Literal>100000</ogc:Literal>
      </ogc:PropertyIsGreaterThanOrEqualTo>
    </ogc:Filter>
    <PointSymbolizer>
      <Graphic>
        <Mark>
          <WellKnownName>circle</WellKnownName>
          <Fill>
```

```
64                <CssParameter name="fill">#0033CC</CssParameter>
65              </Fill>
66            </Mark>
67            <Size>16</Size>
68          </Graphic>
69        </PointSymbolizer>
70      </Rule>
71    </FeatureTypeStyle>
```

Details

Note: Refer to the *Example points layer* to see the attributes for this data. This example has eschewed labels in order to simplify the style, but you can refer to the example *Point with styled label* to see which attributes correspond to which points.

This style contains three rules. Each `<Rule>` varies the style based on the value of the population ("pop") attribute for each point, with smaller values yielding a smaller circle, and larger values yielding a larger circle.

The three rules are designed as follows:

Rule order	**Rule name**	**Population** ("pop")	**Size**
1	SmallPop	Less than 50,000	8
2	MediumPop	50,000 to 100,000	12
3	LargePop	Greater than 100,000	16

The order of the rules does not matter in this case, since each shape is only rendered by a single rule.

The first rule, on **lines 2-22**, specifies the styling of those points whose population attribute is less than 50,000. **Lines 5-10** set this filter, with **lines 6-9** setting the "less than" filter, **line 7** denoting the attribute ("pop"), and **line 8** the value of 50,000. The symbol is a circle (**line 14**), the color is dark blue (`#0033CC`, on **line 16**), and the size is 8 pixels in diameter (**line 19**).

The second rule, on **lines 23-49**, specifies a style for points whose population attribute is greater than or equal to 50,000 and less than 100,000. The population filter is set on **lines 26-37**. This filter is longer than in the first rule because two criteria need to be specified instead of one: a "greater than or equal to" and a "less than" filter. Notice the `And` on **line 27** and **line 36**. This mandates that both filters need to be true for the rule to be applicable. The size of the graphic is set to 12 pixels on **line 46**. All other styling directives are identical to the first rule.

The third rule, on **lines 50-70**, specifies a style for points whose population attribute is greater than or equal to 100,000. The population filter is set on **lines 53-58**, and the only other difference is the size of the circle, which in this rule (**line 67**) is 16 pixels.

The result of this style is that cities with larger populations have larger points.

Zoom-based point

This example alters the style of the points at different zoom levels.

Code

```
View and download the full "Zoom-based point" SLD
```

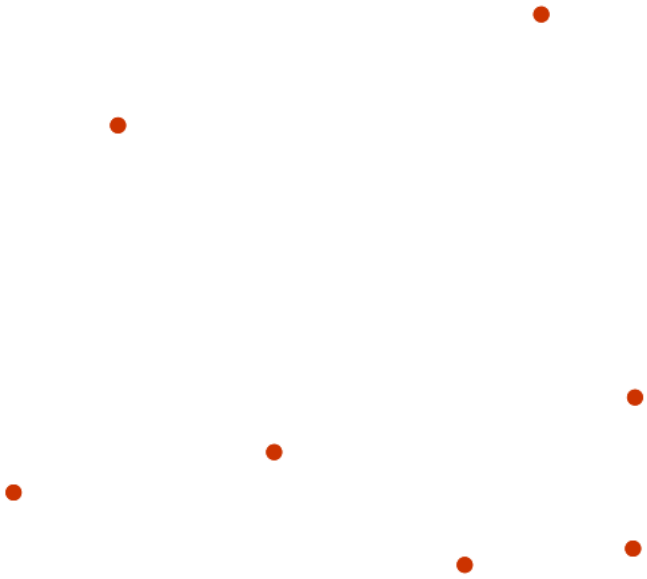


Figure 12.11: *Zoom-based point: Zoomed in*

Figure 12.12: *Zoom-based point: Partially zoomed*

Figure 12.13: *Zoom-based point: Zoomed out*

```
<FeatureTypeStyle>
  <Rule>
    <Name>Large</Name>
    <MaxScaleDenominator>160000000</MaxScaleDenominator>
    <PointSymbolizer>
      <Graphic>
        <Mark>
          <WellKnownName>circle</WellKnownName>
          <Fill>
            <CssParameter name="fill">#CC3300</CssParameter>
          </Fill>
        </Mark>
        <Size>12</Size>
      </Graphic>
    </PointSymbolizer>
  </Rule>
  <Rule>
    <Name>Medium</Name>
    <MinScaleDenominator>160000000</MinScaleDenominator>
    <MaxScaleDenominator>320000000</MaxScaleDenominator>
    <PointSymbolizer>
      <Graphic>
        <Mark>
          <WellKnownName>circle</WellKnownName>
          <Fill>
            <CssParameter name="fill">#CC3300</CssParameter>
          </Fill>
        </Mark>
        <Size>8</Size>
      </Graphic>
    </PointSymbolizer>
  </Rule>
  <Rule>
    <Name>Small</Name>
    <MinScaleDenominator>320000000</MinScaleDenominator>
    <PointSymbolizer>
      <Graphic>
        <Mark>
          <WellKnownName>circle</WellKnownName>
          <Fill>
```

```
41                <CssParameter name="fill">#CC3300</CssParameter>
42              </Fill>
43            </Mark>
44            <Size>4</Size>
45          </Graphic>
46        </PointSymbolizer>
47      </Rule>
48    </FeatureTypeStyle>
```

Details

It is often desirable to make shapes larger at higher zoom levels when creating a natural-looking map. This example styles the points to vary in size based on the zoom level (or more accurately, scale denominator). Scale denominators refer to the scale of the map. A scale denominator of 10,000 means the map has a scale of 1:10,000 in the units of the map projection.

Note: Determining the appropriate scale denominators (zoom levels) to use is beyond the scope of this example.

This style contains three rules. The three rules are designed as follows:

Rule order	Rule name	Scale denominator	Point size
1	Large	1:160,000,000 or less	12
2	Medium	1:160,000,000 to 1:320,000,000	8
3	Small	Greater than 1:320,000,000	4

The order of these rules does not matter since the scales denominated in each rule do not overlap.

The first rule (**lines 2-16**) is for the smallest scale denominator, corresponding to when the view is "zoomed in". The scale rule is set on **line 4**, so that the rule will apply to any map with a scale denominator of 160,000,000 or less. The rule draws a circle (**line 8**), colored red (`#CC3300` on **line 10**) with a size of 12 pixels (**line 13**).

The second rule (**lines 17-32**) is the intermediate scale denominator, corresponding to when the view is "partially zoomed". The scale rules are set on **lines 19-20**, so that the rule will apply to any map with a scale denominator between 160,000,000 and 320,000,000. (The `<MinScaleDenominator>` is inclusive and the `<MaxScaleDenominator>` is exclusive, so a zoom level of exactly 320,000,000 would *not* apply here.) Aside from the scale, the only difference between this rule and the first is the size of the symbol, which is set to 8 pixels on **line 29**.

The third rule (**lines 33-47**) is the largest scale denominator, corresponding to when the map is "zoomed out". The scale rule is set on **line 35**, so that the rule will apply to any map with a scale denominator of 320,000,000 or more. Again, the only other difference between this rule and the others is the size of the symbol, which is set to 4 pixels on **line 44**.

The result of this style is that points are drawn larger as one zooms in and smaller as one zooms out.

12.3.2 Lines

While lines can also seem to be simple shapes, having length but no width, there are many options and tricks for making lines display nicely.

Warning: The code examples shown on this page are **not the full SLD code**, as they omit the SLD header and footer information for the sake of brevity. Please use the links to download the full SLD for each example.

Example lines layer

The `lines layer` used in the examples below contains road information for a fictional country. For reference, the attribute table for the points in this layer is included below.

fid (Feature ID)	**name** (Road name)	**type** (Road class)
line.1	Latway	highway
line.2	Crescent Avenue	secondary
line.3	Forest Avenue	secondary
line.4	Longway	highway
line.5	Saxer Avenue	secondary
line.6	Ridge Avenue	secondary
line.7	Holly Lane	local-road
line.8	Mulberry Street	local-road
line.9	Nathan Lane	local-road
line.10	Central Street	local-road
line.11	Lois Lane	local-road
line.12	Rocky Road	local-road
line.13	Fleet Street	local-road
line.14	Diane Court	local-road
line.15	Cedar Trail	local-road
line.16	Victory Road	local-road
line.17	Highland Road	local-road
line.18	Easy Street	local-road
line.19	Hill Street	local-road
line.20	Country Road	local-road
line.21	Main Street	local-road
line.22	Jani Lane	local-road
line.23	Shinbone Alley	local-road
line.24	State Street	local-road
line.25	River Road	local-road

`Download the lines shapefile`

Simple line

This example specifies lines be colored black with a thickness of 3 pixels.

Code

`View and download the full "Simple line" SLD`

```
1    <FeatureTypeStyle>
2      <Rule>
3        <LineSymbolizer>
4          <Stroke>
5            <CssParameter name="stroke">#000000</CssParameter>
6            <CssParameter name="stroke-width">3</CssParameter>
```

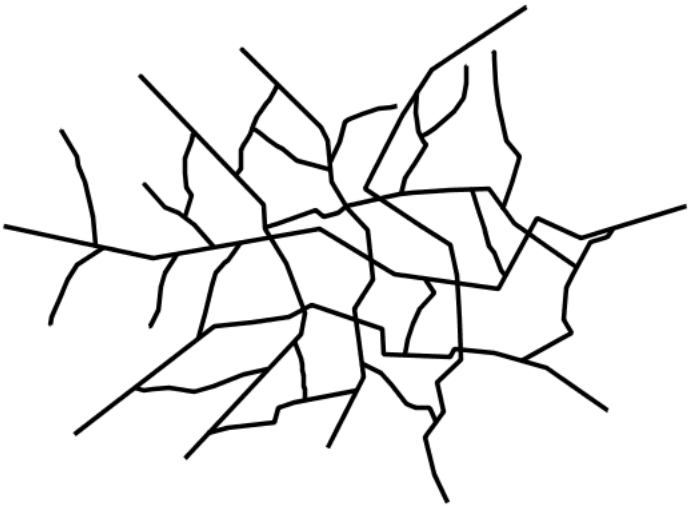


Figure 12.14: *Simple line*

```
7           </Stroke>
8         </LineSymbolizer>
9       </Rule>
10    </FeatureTypeStyle>
```

Details

There is one `<Rule>` in one `<FeatureTypeStyle>` for this SLD, which is the simplest possible situation. (All subsequent examples will contain one `<Rule>` and one `<FeatureTypeStyle>` unless otherwise specified.) Styling lines is accomplished via the `<LineSymbolizer>` (**lines 3-8**). **Line 5** specifies the color of the line to be black (`#000000`), while **line 6** specifies the width of the lines to be 3 pixels.

Line with border

This example shows how to draw lines with borders (sometimes called "cased lines"). In this case the lines are drawn with a 3 pixel blue center and a 1 pixel wide gray border.

Code

View and download the full "Line with border" SLD

```
1     <FeatureTypeStyle>
2        <Rule>
3         <LineSymbolizer>
4           <Stroke>
5             <CssParameter name="stroke">#333333</CssParameter>
6             <CssParameter name="stroke-width">5</CssParameter>
7             <CssParameter name="stroke-linecap">round</CssParameter>
8           </Stroke>
```

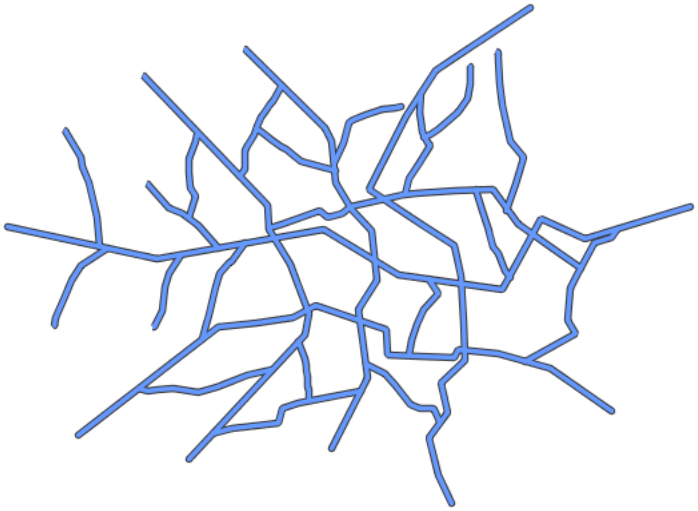


Figure 12.15: *Line with border*

```
9           </LineSymbolizer>
10        </Rule>
11      </FeatureTypeStyle>
12      <FeatureTypeStyle>
13         <Rule>
14          <LineSymbolizer>
15          <Stroke>
16              <CssParameter name="stroke">#6699FF</CssParameter>
17              <CssParameter name="stroke-width">3</CssParameter>
18              <CssParameter name="stroke-linecap">round</CssParameter>
19            </Stroke>
20          </LineSymbolizer>
21         </Rule>
22      </FeatureTypeStyle>
```

Details

Lines in SLD have no notion of a "fill", only "stroke". Thus, unlike points or polygons, it is not possible to style the "edge" of the line geometry. It is, however, possible to achieve this effect by drawing each line twice: once with a certain width and again with a slightly smaller width. This gives the illusion of fill and stroke by obscuring the larger lines everywhere except along the edges of the smaller lines.

Since every line is drawn twice, the order of the rendering is *very* important. GeoServer renders `<FeatureTypeStyle>`s in the order that they are presented in the SLD. In this style, the gray border lines are drawn first via the first `<FeatureTypeStyle>`, followed by the blue center lines in a second `<FeatureTypeStyle>`. This ensures that the blue lines are not obscured by the gray lines, and also ensures proper rendering at intersections, so that the blue lines "connect".

In this example, **lines 1-11** comprise the first `<FeatureTypeStyle>`, which is the outer line (or "stroke"). **Line 5** specifies the color of the line to be dark gray (`#333333`), **line 6** specifies the width of this line to be 5 pixels, and in **line 7** a `stroke-linecap` parameter of `round` renders the ends of the line as rounded instead of flat. (When working with bordered lines using a round line cap ensures that the border connects

properly at the ends of the lines.)

Lines 12-22 comprise the second `<FeatureTypeStyle>`, which is the the inner line (or "fill"). **Line 16** specifies the color of the line to be a medium blue (`#6699FF`), **line 17** specifies the width of this line to be 3 pixels, and **line 18** again renders the edges of the line to be rounded instead of flat.

The result is a 3 pixel blue line with a 1 pixel gray border, since the 5 pixel gray line will display 1 pixel on each side of the 3 pixel blue line.

Dashed line

This example alters the *Simple line* to create a dashed line consisting of 5 pixels of drawn line alternating with 2 pixels of blank space.

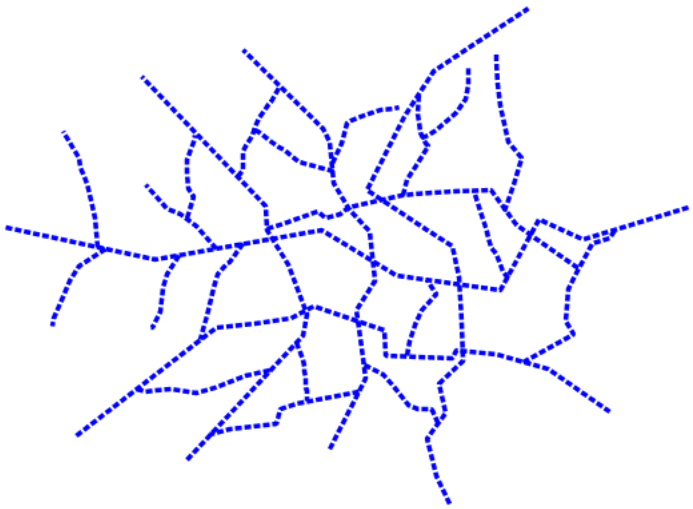


Figure 12.16: *Dashed line*

Code

View and download the full "Dashed line" SLD

```
1    <FeatureTypeStyle>
2      <Rule>
3        <LineSymbolizer>
4          <Stroke>
5            <CssParameter name="stroke">#0000FF</CssParameter>
6            <CssParameter name="stroke-width">3</CssParameter>
7            <CssParameter name="stroke-dasharray">5 2</CssParameter>
8          </Stroke>
9        </LineSymbolizer>
10     </Rule>
11   </FeatureTypeStyle>
```

Details

In this example, **line 5** sets the color of the lines to be blue (`#0000FF`) and **line 6** sets the width of the lines to be 3 pixels. **Line 7** determines the composition of the line dashes. The value of `5 2` creates a repeating pattern of 5 pixels of drawn line, followed by 2 pixels of omitted line.

Railroad (hatching)

This example uses hatching to create a railroad style. Both the line and the hatches are black, with a 2 pixel thickness for the main line and a 1 pixel width for the perpendicular hatches.

Note: This example leverages an SLD extension in GeoServer. Hatching is not part of the standard SLD 1.0 specification.

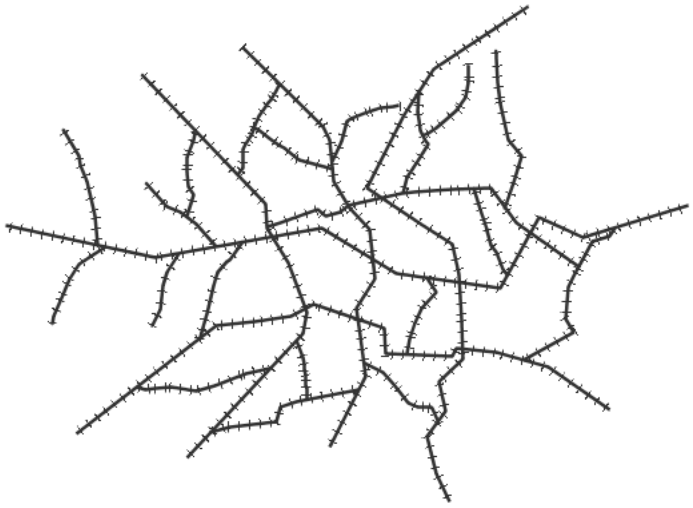


Figure 12.17: *Railroad (hatching)*

Code

View and download the full "Railroad (hatching)" SLD

```
1    <FeatureTypeStyle>
2      <Rule>
3        <LineSymbolizer>
4          <Stroke>
5            <CssParameter name="stroke">#333333</CssParameter>
6            <CssParameter name="stroke-width">3</CssParameter>
7          </Stroke>
8        </LineSymbolizer>
9        <LineSymbolizer>
10         <Stroke>
11           <GraphicStroke>
12             <Graphic>
```

```
13                <Mark>
14                  <WellKnownName>shape://vertline</WellKnownName>
15                  <Stroke>
16                    <CssParameter name="stroke">#333333</CssParameter>
17                    <CssParameter name="stroke-width">1</CssParameter>
18                  </Stroke>
19                </Mark>
20                <Size>12</Size>
21              </Graphic>
22            </GraphicStroke>
23          </Stroke>
24        </LineSymbolizer>
25      </Rule>
26    </FeatureTypeStyle>
```

Details

In this example there are two `<LineSymbolizer>`s. The first symbolizer, on **lines 3-8**, draws a standard line, with **line 5** drawing the lines as dark gray (`#333333`) and **line 6** setting the width of the lines to be 2 pixels.

The hatching is invoked in the second symbolizer, on **lines 9-24**. **Line 14** specifies that the symbolizer draw a vertical line hatch (`shape://vertline`) perpendicular to the line geometry. **Lines 16-17** set the hatch color to dark gray (`#333333`) and width to 1 pixel. Finally, **line 20** specifies both the length of the hatch and the distance between each hatch to both be 12 pixels.

Spaced graphic symbols

This example uses a graphic stroke along with dash arrays to create a "dot and space" line type. Adding the dash array specification allows to control the amount of space between one symbol and the next one. Without using the dash array the lines would be densely populated with dots, each one touching the previous one.

Note: This example may not work in other systems using SLD, since they may not support combining the use of `stroke-dasharray` and `GraphicStroke`. While the SLD is spec-compliant, the SLD specification does not state what this combination is supposed to produce.

Code

View and download the full "Spaced symbols" SLD

```
1     <FeatureTypeStyle>
2       <Rule>
3         <LineSymbolizer>
4           <Stroke>
5             <GraphicStroke>
6               <Graphic>
7                 <Mark>
8                   <WellKnownName>circle</WellKnownName>
9                   <Fill>
10                    <CssParameter name="fill">#666666</CssParameter>
11                  </Fill>
12                  <Stroke>
```

Figure 12.18: *Spaced symbols along a line*

```
13                    <CssParameter name="stroke">#333333</CssParameter>
14                    <CssParameter name="stroke-width">1</CssParameter>
15                  </Stroke>
16                </Mark>
17                <Size>4</Size>
18              </Graphic>
19            </GraphicStroke>
20            <CssParameter name="stroke-dasharray">4 6</CssParameter>
21          </Stroke>
22        </LineSymbolizer>
23      </Rule>
24    </FeatureTypeStyle>
```

Details

This example, like others before, uses a `GraphicStroke` to place a graphic symbol along a line. The symbol, defined at **lines 7-16** is a 4 pixel gray circle with a dark gray outline. The spacing between symbols is controlled with the `stroke-dasharray` at **line 20**, which specifies 4 pixels of pen-down (just enough to draw the circle) and 6 pixels of pen-up, to provide the spacing.

Alternating symbols with dash offsets

This example shows how to create a complex line style which alternates a dashed line and a graphic symbol. The code builds on features shown in the previous examples:

- `stroke-dasharray` controls pen-down/pen-up behavior to generate dashed lines
- `GraphicStroke` places symbols along a line
- combining the two allows control of symbol spacing

This also shows the usage of a *dash offset*, which controls where rendering starts in the dash array. For example, with a dash array of `5 10` and a dash offset of `7` the renderer starts drawing the pattern 7 pixels from the beginning. It skips the 5 pixels pen-down section and 2 pixels of the pen-up section, then draws the remaining 8 pixels of pen-up, then 5 down, 10 up, and so on.

The example shows how to use these features to create two synchronized sequences of dash arrays, one drawing line segments and the other symbols.

Note: This example may not work in other systems using SLD, since they may not support combining the use of `stroke-dasharray` and `GraphicStroke`. While the SLD is spec-compliant, the SLD specification does not state what this combination is supposed to produce.

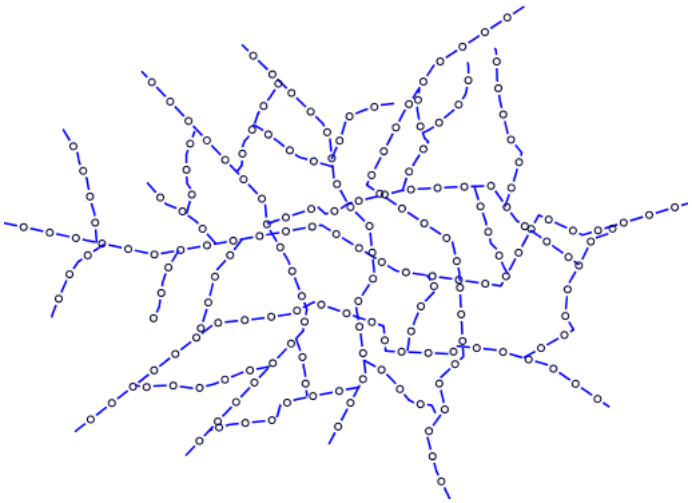


Figure 12.19: *Alternating dash and symbol*

Code

View and download the full "Spaced symbols" SLD

```
   <FeatureTypeStyle>
     <Rule>
       <LineSymbolizer>
         <Stroke>
           <CssParameter name="stroke">#0000FF</CssParameter>
           <CssParameter name="stroke-width">1</CssParameter>
           <CssParameter name="stroke-dasharray">10 10</CssParameter>
         </Stroke>
       </LineSymbolizer>
       <LineSymbolizer>
         <Stroke>
           <GraphicStroke>
             <Graphic>
               <Mark>
                 <WellKnownName>circle</WellKnownName>
                 <Stroke>
                   <CssParameter name="stroke">#000033</CssParameter>
                   <CssParameter name="stroke-width">1</CssParameter>
                 </Stroke>
               </Mark>
               <Size>5</Size>
             </Graphic>
           </GraphicStroke>
           <CssParameter name="stroke-dasharray">5 15</CssParameter>
           <CssParameter name="stroke-dashoffset">7.5</CssParameter>
```

```
26          </Stroke>
27        </LineSymbolizer>
28      </Rule>
29    </FeatureTypeStyle>
```

Details

In this example two `LineSymbolizer`s use `stroke-dasharray` and different symbology to produce a sequence of alternating dashes and symbols. The first symbolizer (**lines 3-9**) is a simple dashed line alternating 10 pixels of pen-down with 10 pixels of pen-up. The second symbolizer (**lines 10-27**) alternates a 5 pixel empty circle with 15 pixels of white space. The circle symbol is produced by a `Mark` element, with its symbology specified by `stroke` parameters (**lines 17-18**). The spacing between symbols is controlled with the `stroke-dasharray` (**line 24**), which specifies 5 pixels of pen-down (just enough to draw the circle) and 15 pixels of pen-up. In order to have the two sequences positioned correctly the second one uses a `stroke-dashoffset` of 7.5 (**line 25**). This makes the sequence start with 12.5 pixels of white space, then a circle (which is then centered between the two line segments of the other pattern), then 15 pixels of white space, and so on.

Line with default label

This example shows a text label on the simple line. This is how a label will be displayed in the absence of any other customization.

Figure 12.20: *Line with default label*

Code

`View and download the full "Line with default label" SLD`

```
<FeatureTypeStyle>
  <Rule>
    <LineSymbolizer>
      <Stroke>
        <CssParameter name="stroke">#FF0000</CssParameter>
      </Stroke>
    </LineSymbolizer>
    <TextSymbolizer>
      <Label>
        <ogc:PropertyName>name</ogc:PropertyName>
      </Label>
      <LabelPlacement>
        <LinePlacement />
      </LabelPlacement>
      <Fill>
        <CssParameter name="fill">#000000</CssParameter>
      </Fill>
    </TextSymbolizer>
  </Rule>
</FeatureTypeStyle>
```

Details

In this example, there is one rule with a `<LineSymbolizer>` and a `<TextSymbolizer>`. The `<LineSymbolizer>` (**lines 3-7**) draws red lines (`#FF0000`). Since no width is specified, the default is set to 1 pixel. The `<TextSymbolizer>` (**lines 8-15**) determines the labeling of the lines. **Lines 9-11** specify that the text of the label will be determined by the value of the "name" attribute for each line. (Refer to the attribute table in the *Example lines layer* section if necessary.) **Line 13** sets the text color to black. All other details about the label are set to the renderer default, which here is Times New Roman font, font color black, and font size of 10 pixels.

Label following line

This example renders the text label to follow the contour of the lines.

Note: Labels following lines is an SLD extension specific to GeoServer. It is not part of the SLD 1.0 specification.

Code

`View and download the full "Label following line" SLD`

```
<FeatureTypeStyle>
  <Rule>
    <LineSymbolizer>
      <Stroke>
        <CssParameter name="stroke">#FF0000</CssParameter>
      </Stroke>
    </LineSymbolizer>
    <TextSymbolizer>
      <Label>
        <ogc:PropertyName>name</ogc:PropertyName>
      </Label>
```

Figure 12.21: *Label following line*

```
12          <LabelPlacement>
13            <LinePlacement />
14          </LabelPlacement>
15          <Fill>
16            <CssParameter name="fill">#000000</CssParameter>
17          </Fill>
18          <VendorOption name="followLine">true</VendorOption>
19        </TextSymbolizer>
20      </Rule>
21    </FeatureTypeStyle>
```

Details

As the *Alternating symbols with dash offsets* example showed, the default label behavior isn't optimal. The label is displayed at a tangent to the line itself, leading to uncertainty as to which label corresponds to which line.

This example is similar to the *Alternating symbols with dash offsets* example with the exception of **lines 12-18**. **Line 18** sets the option to have the label follow the line, while **lines 12-14** specify that the label is placed along a line. If `<LinePlacement />` is not specified in an SLD, then `<PointPlacement />` is assumed, which isn't compatible with line-specific rendering options.

Note: Not all labels are shown due to label conflict resolution. See the next section on *Optimized label placement* for an example of how to maximize label display.

Optimized label placement

This example optimizes label placement for lines such that the maximum number of labels are displayed.

Note: This example uses options that are specific to GeoServer and are not part of the SLD 1.0 specification.

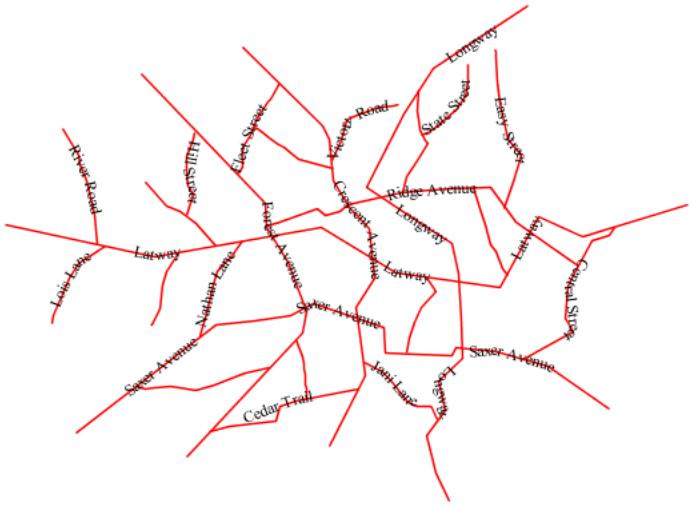


Figure 12.22: *Optimized label*

Code

View and download the full "Optimized label" SLD

```
  <FeatureTypeStyle>
    <Rule>
      <LineSymbolizer>
        <Stroke>
          <CssParameter name="stroke">#FF0000</CssParameter>
        </Stroke>
      </LineSymbolizer>
      <TextSymbolizer>
        <Label>
          <ogc:PropertyName>name</ogc:PropertyName>
        </Label>
        <LabelPlacement>
           <LinePlacement />
        </LabelPlacement>
        <Fill>
          <CssParameter name="fill">#000000</CssParameter>
        </Fill>
        <VendorOption name="followLine">true</VendorOption>
        <VendorOption name="maxAngleDelta">90</VendorOption>
        <VendorOption name="maxDisplacement">400</VendorOption>
        <VendorOption name="repeat">150</VendorOption>
      </TextSymbolizer>
    </Rule>
  </FeatureTypeStyle>
```

Details

GeoServer uses "conflict resolution" to ensure that labels aren't drawn on top of other labels, obscuring them both. This accounts for the reason why many lines don't have labels in the previous example, *Label following line*. While this setting can be toggled, it is usually a good idea to leave it on and use other label placement options to ensure that labels are drawn as often as desired and in the correct places. This example does just that.

This example is similar to the previous example, *Label following line*. The only differences are contained in **lines 18-21**. **Line 19** sets the maximum angle that the label will follow. This sets the label to never bend more than 90 degrees to prevent the label from becoming illegible due to a pronounced curve or angle. **Line 20** sets the maximum displacement of the label to be 400 pixels. In order to resolve conflicts with overlapping labels, GeoServer will attempt to move the labels such that they are no longer overlapping. This value sets how far the label can be moved relative to its original placement. Finally, **line 21** sets the labels to be repeated every 150 pixels. A feature will typically receive only one label, but this can cause confusion for long lines. Setting the label to repeat ensures that the line is always labeled locally.

Optimized and styled label

This example improves the style of the labels from the *Optimized label placement* example.

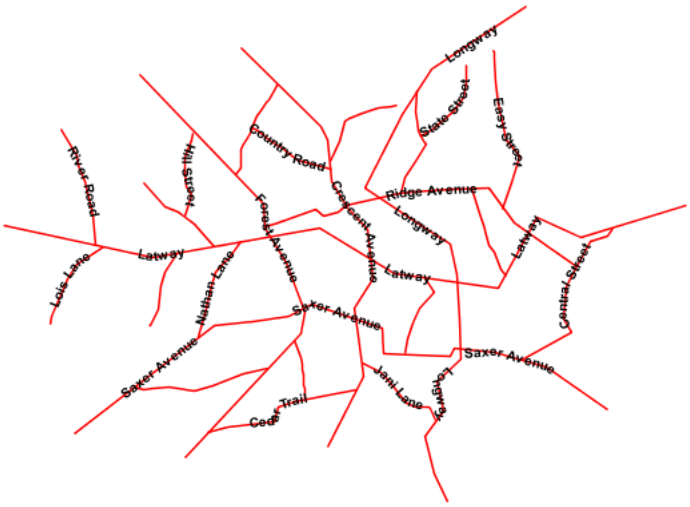


Figure 12.23: *Optimized and styled label*

Code

View and download the full "Optimized and styled label" SLD

```
1     <FeatureTypeStyle>
2       <Rule>
3         <LineSymbolizer>
4           <Stroke>
5             <CssParameter name="stroke">#FF0000</CssParameter>
```

```
      </Stroke>
    </LineSymbolizer>
    <TextSymbolizer>
      <Label>
        <ogc:PropertyName>name</ogc:PropertyName>
      </Label>
      <LabelPlacement>
        <LinePlacement />
      </LabelPlacement>
      <Fill>
        <CssParameter name="fill">#000000</CssParameter>
      </Fill>
      <Font>
        <CssParameter name="font-family">Arial</CssParameter>
        <CssParameter name="font-size">10</CssParameter>
        <CssParameter name="font-style">normal</CssParameter>
        <CssParameter name="font-weight">bold</CssParameter>
      </Font>
      <VendorOption name="followLine">true</VendorOption>
      <VendorOption name="maxAngleDelta">90</VendorOption>
      <VendorOption name="maxDisplacement">400</VendorOption>
      <VendorOption name="repeat">150</VendorOption>
    </TextSymbolizer>
  </Rule>
</FeatureTypeStyle>
```

Details

This example is similar to the *Optimized label placement*. The only difference is in the font information, which is contained in **lines 18-23**. **Line 19** sets the font family to be "Arial", **line 20** sets the font size to 10, **line 21** sets the font style to "normal" (as opposed to "italic" or "oblique"), and **line 22** sets the font weight to "bold" (as opposed to "normal").

Attribute-based line

This example styles the lines differently based on the "type" (Road class) attribute.

Code

View and download the full "Attribute-based line" SLD

```
<FeatureTypeStyle>
  <Rule>
    <Name>local-road</Name>
    <ogc:Filter>
      <ogc:PropertyIsEqualTo>
        <ogc:PropertyName>type</ogc:PropertyName>
        <ogc:Literal>local-road</ogc:Literal>
      </ogc:PropertyIsEqualTo>
    </ogc:Filter>
    <LineSymbolizer>
      <Stroke>
        <CssParameter name="stroke">#009933</CssParameter>
        <CssParameter name="stroke-width">2</CssParameter>
```

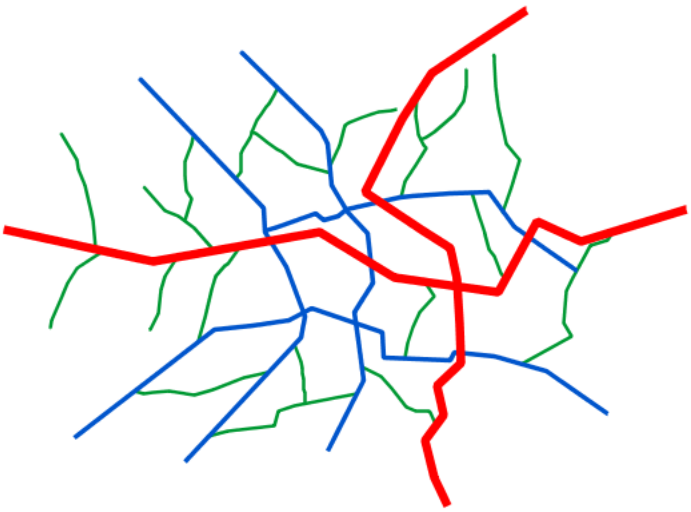


Figure 12.24: *Attribute-based line*

```
14          </Stroke>
15        </LineSymbolizer>
16      </Rule>
17    </FeatureTypeStyle>
18    <FeatureTypeStyle>
19      <Rule>
20        <Name>secondary</Name>
21        <ogc:Filter>
22          <ogc:PropertyIsEqualTo>
23            <ogc:PropertyName>type</ogc:PropertyName>
24            <ogc:Literal>secondary</ogc:Literal>
25          </ogc:PropertyIsEqualTo>
26        </ogc:Filter>
27        <LineSymbolizer>
28          <Stroke>
29            <CssParameter name="stroke">#0055CC</CssParameter>
30            <CssParameter name="stroke-width">3</CssParameter>
31          </Stroke>
32        </LineSymbolizer>
33      </Rule>
34    </FeatureTypeStyle>
35    <FeatureTypeStyle>
36      <Rule>
37      <Name>highway</Name>
38        <ogc:Filter>
39          <ogc:PropertyIsEqualTo>
40            <ogc:PropertyName>type</ogc:PropertyName>
41            <ogc:Literal>highway</ogc:Literal>
42          </ogc:PropertyIsEqualTo>
43        </ogc:Filter>
44        <LineSymbolizer>
45          <Stroke>
46            <CssParameter name="stroke">#FF0000</CssParameter>
47            <CssParameter name="stroke-width">6</CssParameter>
```

```
48          </Stroke>
49        </LineSymbolizer>
50      </Rule>
51    </FeatureTypeStyle>
```

Details

Note: Refer to the *Example lines layer* to see the attributes for the layer. This example has eschewed labels in order to simplify the style, but you can refer to the example *Optimized and styled label* to see which attributes correspond to which points.

There are three types of road classes in our fictional country, ranging from back roads to high-speed freeways: "highway", "secondary", and "local-road". In order to handle each case separately, there is more than one `<FeatureTypeStyle>`, each containing a single rule. This ensures that each road type is rendered in order, as each `<FeatureTypeStyle>` is drawn based on the order in which it appears in the SLD.

The three rules are designed as follows:

Rule order	Rule name / type	Color	Size
1	local-road	`#009933` (green)	2
2	secondary	`#0055CC` (blue)	3
3	highway	`#FF0000` (red)	6

Lines 2-16 comprise the first `<Rule>`. **Lines 4-9** set the filter for this rule, such that the "type" attribute has a value of "local-road". If this condition is true for a particular line, the rule is rendered according to the `<LineSymbolizer>` which is on **lines 10-15**. **Lines 12-13** set the color of the line to be a dark green (`#009933`) and the width to be 2 pixels.

Lines 19-33 comprise the second `<Rule>`. **Lines 21-26** set the filter for this rule, such that the "type" attribute has a value of "secondary". If this condition is true for a particular line, the rule is rendered according to the `<LineSymbolizer>` which is on **lines 27-32**. **Lines 29-30** set the color of the line to be a dark blue (`#0055CC`) and the width to be 3 pixels, making the lines slightly thicker than the "local-road" lines and also a different color.

Lines 36-50 comprise the third and final `<Rule>`. **Lines 38-43** set the filter for this rule, such that the "type" attribute has a value of "primary". If this condition is true for a particular line, the rule is rendered according to the `<LineSymbolizer>` which is on **lines 44-49**. **Lines 46-47** set the color of the line to be a bright red (`#FF0000`) and the width to be 6 pixels, so that these lines are rendered on top of and thicker than the other two road classes. In this way, the "primary" roads are given priority in the map rendering.

Zoom-based line

This example alters the *Simple line* style at different zoom levels.

Code

View and download the full "Zoom-based line" SLD

```
1     <FeatureTypeStyle>
2       <Rule>
3         <Name>Large</Name>
4         <MaxScaleDenominator>180000000</MaxScaleDenominator>
```

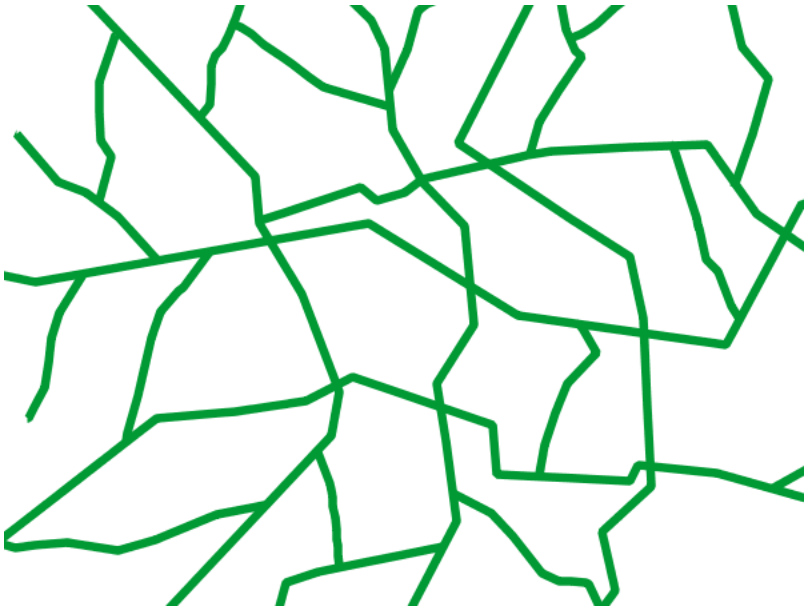


Figure 12.25: *Zoom-based line: Zoomed in*

Figure 12.26: *Zoom-based line: Partially zoomed*

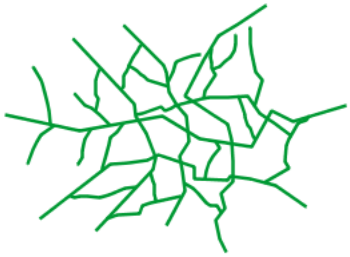


Figure 12.27: *Zoom-based line: Zoomed out*

```
        <LineSymbolizer>
          <Stroke>
            <CssParameter name="stroke">#009933</CssParameter>
            <CssParameter name="stroke-width">6</CssParameter>
          </Stroke>
        </LineSymbolizer>
      </Rule>
      <Rule>
        <Name>Medium</Name>
        <MinScaleDenominator>180000000</MinScaleDenominator>
        <MaxScaleDenominator>360000000</MaxScaleDenominator>
        <LineSymbolizer>
          <Stroke>
            <CssParameter name="stroke">#009933</CssParameter>
            <CssParameter name="stroke-width">4</CssParameter>
          </Stroke>
        </LineSymbolizer>
      </Rule>
      <Rule>
        <Name>Small</Name>
        <MinScaleDenominator>360000000</MinScaleDenominator>
        <LineSymbolizer>
          <Stroke>
            <CssParameter name="stroke">#009933</CssParameter>
            <CssParameter name="stroke-width">2</CssParameter>
          </Stroke>
        </LineSymbolizer>
      </Rule>
    </FeatureTypeStyle>
```

Details

It is often desirable to make shapes larger at higher zoom levels when creating a natural-looking map. This example varies the thickness of the lines according to the zoom level (or more accurately, scale denominator). Scale denominators refer to the scale of the map. A scale denominator of 10,000 means the map has a scale of 1:10,000 in the units of the map projection.

Note: Determining the appropriate scale denominators (zoom levels) to use is beyond the scope of this example.

This style contains three rules. The three rules are designed as follows:

Rule order	Rule name	Scale denominator	Line width
1	Large	1:180,000,000 or less	6
2	Medium	1:180,000,000 to 1:360,000,000	4
3	Small	Greater than 1:360,000,000	2

The order of these rules does not matter since the scales denominated in each rule do not overlap.

The first rule (**lines 2-11**) is the smallest scale denominator, corresponding to when the view is "zoomed in". The scale rule is set on **line 4**, so that the rule will apply to any map with a scale denominator of 180,000,000 or less. **Line 7-8** draws the line to be dark green (`#009933`) with a width of 6 pixels.

The second rule (**lines 12-22**) is the intermediate scale denominator, corresponding to when the view is "partially zoomed". **Lines 14-15** set the scale such that the rule will apply to any map with scale denominators between 180,000,000 and 360,000,000. (The `<MinScaleDenominator>` is inclusive and the `<MaxScaleDenominator>` is exclusive, so a zoom level of exactly 360,000,000 would *not* apply here.) Aside from the scale, the only difference between this rule and the previous is the width of the lines, which is set to 4 pixels on **line 19**.

The third rule (**lines 23-32**) is the largest scale denominator, corresponding to when the map is "zoomed out". The scale rule is set on **line 25**, so that the rule will apply to any map with a scale denominator of 360,000,000 or greater. Again, the only other difference between this rule and the others is the width of the lines, which is set to 2 pixels on **line 29**.

The result of this style is that lines are drawn with larger widths as one zooms in and smaller widths as one zooms out.

12.3.3 Polygons

Polygons are two dimensional shapes that contain both an outer edge (or "stroke") and an inside (or "fill"). A polygon can be thought of as an irregularly-shaped point and is styled in similar ways to points.

> **Warning:** The code examples shown on this page are **not the full SLD code**, as they omit the SLD header and footer information for the sake of brevity. Please use the links to download the full SLD for each example.

Example polygons layer

The `polygons layer` used below contains county information for a fictional country. For reference, the attribute table for the polygons is included below.

fid (Feature ID)	**name** (County name)	**pop** (Population)
polygon.1	Irony County	412234
polygon.2	Tracker County	235421
polygon.3	Dracula County	135022
polygon.4	Poly County	1567879
polygon.5	Bearing County	201989
polygon.6	Monte Cristo County	152734
polygon.7	Massive County	67123
polygon.8	Rhombus County	198029

`Download the polygons shapefile`

Simple polygon

This example shows a polygon filled in blue.

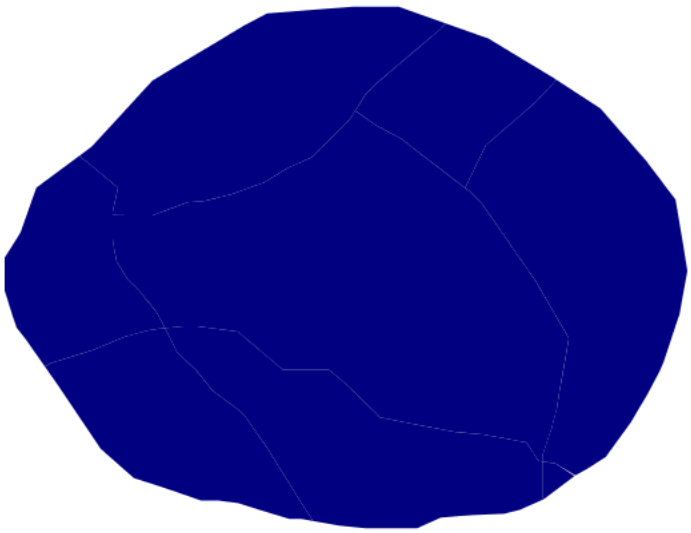


Figure 12.28: *Simple polygon*

Code

`View and download the full "Simple polygon" SLD`

```
1    <FeatureTypeStyle>
2      <Rule>
3        <PolygonSymbolizer>
4          <Fill>
5            <CssParameter name="fill">#000080</CssParameter>
6          </Fill>
7        </PolygonSymbolizer>
8      </Rule>
9    </FeatureTypeStyle>
```

Details

There is one `<Rule>` in one `<FeatureTypeStyle>` for this style, which is the simplest possible situation. (All subsequent examples will share this characteristic unless otherwise specified.) Styling polygons is accomplished via the `<PolygonSymbolizer>` (**lines 3-7**). **Line 5** specifies dark blue (`#000080`) as the polygon's fill color.

Note: The light-colored borders around the polygons in the figure are artifacts of the renderer caused by the polygons being adjacent. There is no border in this style.

Simple polygon with stroke

This example adds a 2 pixel white stroke to the *Simple polygon* example.

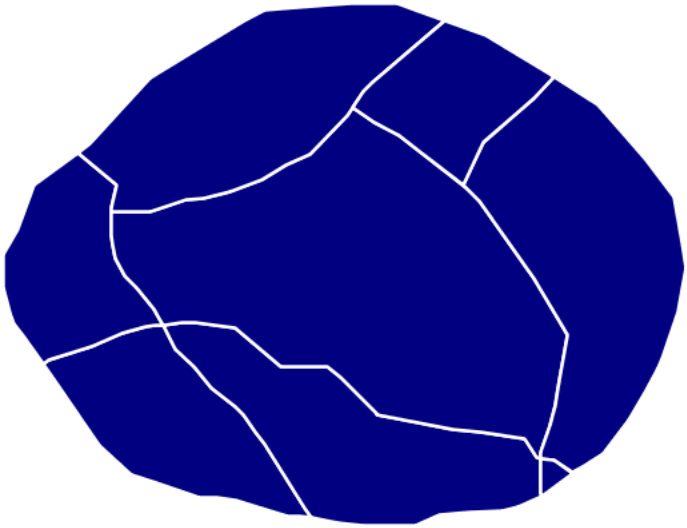


Figure 12.29: *Simple polygon with stroke*

Code

View and download the full "Simple polygon with stroke" SLD

```
1    <FeatureTypeStyle>
2      <Rule>
3        <PolygonSymbolizer>
4          <Fill>
5            <CssParameter name="fill">#000080</CssParameter>
6          </Fill>
7          <Stroke>
8            <CssParameter name="stroke">#FFFFFF</CssParameter>
9            <CssParameter name="stroke-width">2</CssParameter>
10         </Stroke>
11       </PolygonSymbolizer>
```

```
12      </Rule>
13    </FeatureTypeStyle>
```

Details

This example is similar to the *Simple polygon* example above, with the addition of the `<Stroke>` tag (**lines 7-10**). **Line 8** sets the color of stroke to white (`#FFFFFF`) and **line 9** sets the width of the stroke to 2 pixels.

Transparent polygon

This example builds on the *Simple polygon with stroke* example and makes the fill partially transparent by setting the opacity to 50%.

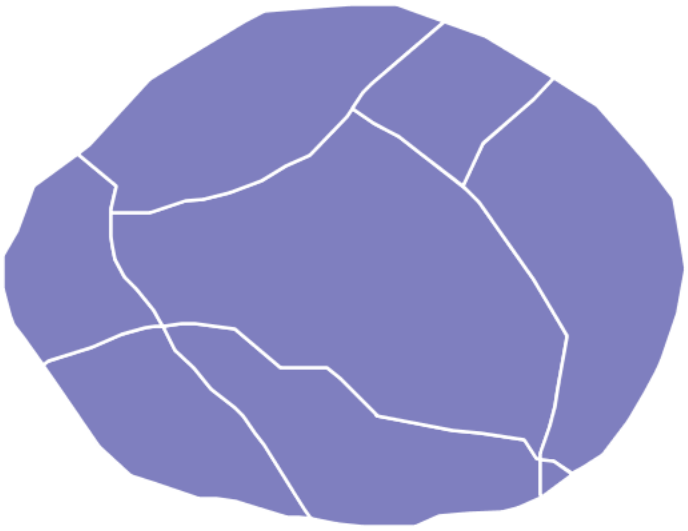


Figure 12.30: *Transparent polygon*

Code

View and download the full "Transparent polygon" SLD

```
1     <FeatureTypeStyle>
2       <Rule>
3         <PolygonSymbolizer>
4           <Fill>
5             <CssParameter name="fill">#000080</CssParameter>
6             <CssParameter name="fill-opacity">0.5</CssParameter>
7           </Fill>
8           <Stroke>
9             <CssParameter name="stroke">#FFFFFF</CssParameter>
10            <CssParameter name="stroke-width">2</CssParameter>
11          </Stroke>
12        </PolygonSymbolizer>
```

```
13      </Rule>
14    </FeatureTypeStyle>
```

Details

This example is similar to the *Simple polygon with stroke* example, save for defining the fill's opacity in **line 6**. The value of 0.5 results in partially transparent fill that is 50% opaque. An opacity value of 1 would draw the fill as 100% opaque, while an opacity value of 0 would result in a completely transparent (0% opaque) fill. In this example, since the background is white, the dark blue looks lighter. Were the points imposed on a dark background, the resulting color would be darker.

Graphic fill

This example fills the polygons with a tiled graphic.

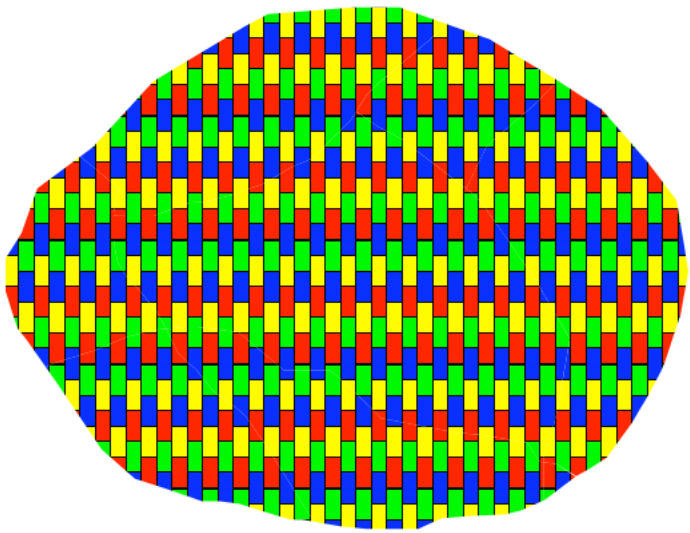


Figure 12.31: *Graphic fill*

Code

View and download the full "Graphic fill" SLD

```
1     <FeatureTypeStyle>
2       <Rule>
3         <PolygonSymbolizer>
4           <Fill>
5             <GraphicFill>
6               <Graphic>
7                 <ExternalGraphic>
8                   <OnlineResource
9                     xlink:type="simple"
10                    xlink:href="colorblocks.png" />
```

```
11                <Format>image/png</Format>
12              </ExternalGraphic>
13            <Size>93</Size>
14            </Graphic>
15          </GraphicFill>
16        </Fill>
17      </PolygonSymbolizer>
18    </Rule>
19  </FeatureTypeStyle>
```

Details

This style fills the polygon with a tiled graphic. This is known as an `<ExternalGraphic>` in SLD, to distinguish it from commonly-used shapes such as squares and circles that are "internal" to the renderer. **Lines 7-12** specify details for the graphic, with **line 10** setting the path and file name of the graphic and **line 11** indicating the file format (MIME type) of the graphic (`image/png`). Although a full URL could be specified if desired, no path information is necessary in **line 11** because this graphic is contained in the same directory as the SLD. **Line 13** determines the height of the displayed graphic in pixels; if the value differs from the height of the graphic then it will be scaled accordingly while preserving the aspect ratio.

Figure 12.32: *Graphic used for fill*

Hatching fill

This example fills the polygons with a hatching pattern.

Note: This example leverages an SLD extension in GeoServer. Hatching is not part of the standard SLD 1.0 specification.

Code

View and download the full "Hatching fill" SLD

```
1   <FeatureTypeStyle>
2     <Rule>
3       <PolygonSymbolizer>
4         <Fill>
5           <GraphicFill>
6             <Graphic>
7               <Mark>
8                 <WellKnownName>shape://times</WellKnownName>
9                 <Stroke>
10                  <CssParameter name="stroke">#990099</CssParameter>
11                  <CssParameter name="stroke-width">1</CssParameter>
12                </Stroke>
13              </Mark>
```

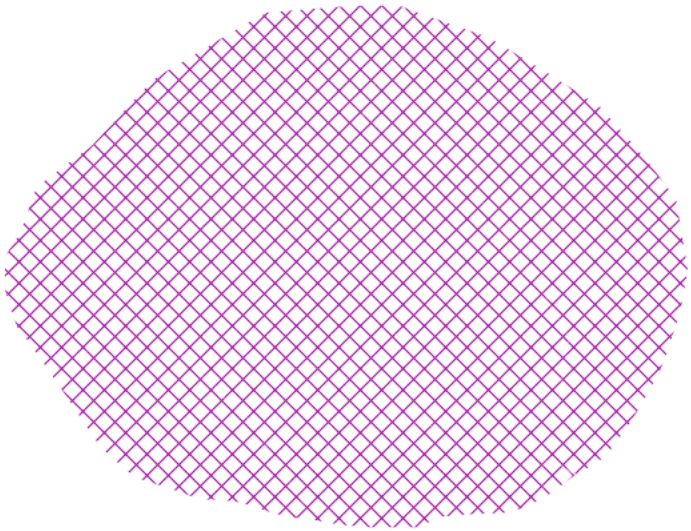


Figure 12.33: *Hatching fill*

```
14                <Size>16</Size>
15              </Graphic>
16            </GraphicFill>
17          </Fill>
18        </PolygonSymbolizer>
19      </Rule>
20    </FeatureTypeStyle>
```

Details

In this example, there is a `<GraphicFill>` tag as in the *Graphic fill* example, but a `<Mark>` (**lines 7-13**) is used instead of an `<ExternalGraphic>`. **Line 8** specifies a "times" symbol (an "x") be tiled throughout the polygon. **Line 10** sets the color to purple (`#990099`), **line 11** sets the width of the hatches to 1 pixel, and **line 14** sets the size of the tile to 16 pixels. Because hatch tiles are always square, the `<Size>` sets both the width and the height.

Polygon with default label

This example shows a text label on the polygon. In the absence of any other customization, this is how a label will be displayed.

Code

View and download the full "Polygon with default label" SLD

```
1     <FeatureTypeStyle>
2       <Rule>
3         <PolygonSymbolizer>
4           <Fill>
```

Figure 12.34: *Polygon with default label*

```
5           <CssParameter name="fill">#40FF40</CssParameter>
6         </Fill>
7         <Stroke>
8           <CssParameter name="stroke">#FFFFFF</CssParameter>
9           <CssParameter name="stroke-width">2</CssParameter>
10        </Stroke>
11      </PolygonSymbolizer>
12      <TextSymbolizer>
13        <Label>
14          <ogc:PropertyName>name</ogc:PropertyName>
15        </Label>
16      </TextSymbolizer>
17    </Rule>
18  </FeatureTypeStyle>
```

Details

In this example there is a `<PolygonSymbolizer>` and a `<TextSymbolizer>`. **Lines 3-11** comprise the `<PolygonSymbolizer>`. The fill of the polygon is set on **line 5** to a light green (`#40FF40`) while the stroke of the polygon is set on **lines 8-9** to white (`#FFFFFF`) with a thickness of 2 pixels. The label is set in the `<TextSymbolizer>` on **lines 12-16**, with **line 14** determining what text to display, in this case the value of the "name" attribute. (Refer to the attribute table in the *Example polygons layer* section if necessary.) All other details about the label are set to the renderer default, which here is Times New Roman font, font color black, and font size of 10 pixels.

Label halo

This example alters the look of the *Polygon with default label* by adding a white halo to the label.

Figure 12.35: *Label halo*

Code

View and download the full "Label halo" SLD

```
<FeatureTypeStyle>
  <Rule>
    <PolygonSymbolizer>
      <Fill>
        <CssParameter name="fill">#40FF40</CssParameter>
      </Fill>
      <Stroke>
        <CssParameter name="stroke">#FFFFFF</CssParameter>
        <CssParameter name="stroke-width">2</CssParameter>
      </Stroke>
    </PolygonSymbolizer>
    <TextSymbolizer>
      <Label>
        <ogc:PropertyName>name</ogc:PropertyName>
      </Label>
      <Halo>
        <Radius>3</Radius>
        <Fill>
          <CssParameter name="fill">#FFFFFF</CssParameter>
        </Fill>
      </Halo>
    </TextSymbolizer>
  </Rule>
</FeatureTypeStyle>
```

Details

This example is similar to the *Polygon with default label*, with the addition of a halo around the labels on **lines 16-21**. A halo creates a color buffer around the label to improve label legibility. **Line 17** sets the radius of the halo, extending the halo 3 pixels around the edge of the label, and **line 19** sets the color of the halo to white (`#FFFFFF`). Since halos are most useful when set to a sharp contrast relative to the text color, this example uses a white halo around black text to ensure optimum readability.

Polygon with styled label

This example improves the label style from the *Polygon with default label* example by centering the label on the polygon, specifying a different font name and size, and setting additional label placement optimizations.

Note: The label placement optimizations discussed below (the `<VendorOption>` tags) are SLD extensions that are custom to GeoServer. They are not part of the SLD 1.0 specification.

Figure 12.36: *Polygon with styled label*

Code

View and download the full "Polygon with styled label" SLD

```
1     <FeatureTypeStyle>
2       <Rule>
3         <PolygonSymbolizer>
4           <Fill>
5             <CssParameter name="fill">#40FF40</CssParameter>
6           </Fill>
7           <Stroke>
8             <CssParameter name="stroke">#FFFFFF</CssParameter>
9             <CssParameter name="stroke-width">2</CssParameter>
```

```
10          </Stroke>
11        </PolygonSymbolizer>
12        <TextSymbolizer>
13          <Label>
14            <ogc:PropertyName>name</ogc:PropertyName>
15          </Label>
16          <Font>
17            <CssParameter name="font-family">Arial</CssParameter>
18            <CssParameter name="font-size">11</CssParameter>
19            <CssParameter name="font-style">normal</CssParameter>
20            <CssParameter name="font-weight">bold</CssParameter>
21          </Font>
22          <LabelPlacement>
23            <PointPlacement>
24              <AnchorPoint>
25                <AnchorPointX>0.5</AnchorPointX>
26                <AnchorPointY>0.5</AnchorPointY>
27              </AnchorPoint>
28            </PointPlacement>
29          </LabelPlacement>
30          <Fill>
31            <CssParameter name="fill">#000000</CssParameter>
32          </Fill>
33          <VendorOption name="autoWrap">60</VendorOption>
34          <VendorOption name="maxDisplacement">150</VendorOption>
35        </TextSymbolizer>
36      </Rule>
37    </FeatureTypeStyle>
```

Details

This example is similar to the *Polygon with default label* example, with additional styling options within the `<TextSymbolizer>` on lines **12-35**. **Lines 16-21** set the font styling. **Line 17** sets the font family to be Arial, **line 18** sets the font size to 11 pixels, **line 19** sets the font style to "normal" (as opposed to "italic" or "oblique"), and **line 20** sets the font weight to "bold" (as opposed to "normal").

The `<LabelPlacement>` tag on **lines 22-29** affects where the label is placed relative to the centroid of the polygon. **Line 21** centers the label by positioning it 50% (or 0.5) of the way horizontally along the centroid of the polygon. **Line 22** centers the label vertically in exactly the same way.

Finally, there are two added touches for label placement optimization: **line 33** ensures that long labels are split across multiple lines by setting line wrapping on the labels to 60 pixels, and **line 34** allows the label to be displaced by up to 150 pixels. This ensures that labels are compacted and less likely to spill over polygon boundaries. Notice little Massive County in the corner, whose label is now displayed."

Attribute-based polygon

This example styles the polygons differently based on the "pop" (Population) attribute.

Code

`View and download the full "Attribute-based polygon" SLD`

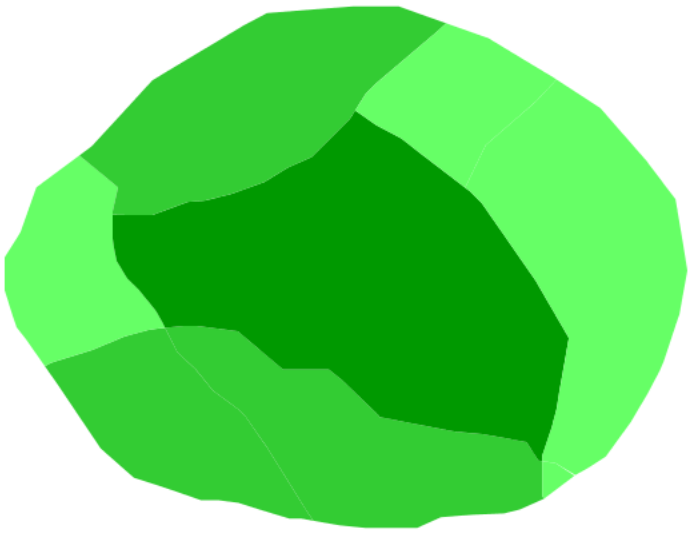


Figure 12.37: *Attribute-based polygon*

```
<FeatureTypeStyle>
  <Rule>
    <Name>SmallPop</Name>
    <Title>Less Than 200,000</Title>
    <ogc:Filter>
      <ogc:PropertyIsLessThan>
        <ogc:PropertyName>pop</ogc:PropertyName>
        <ogc:Literal>200000</ogc:Literal>
      </ogc:PropertyIsLessThan>
    </ogc:Filter>
    <PolygonSymbolizer>
      <Fill>
        <CssParameter name="fill">#66FF66</CssParameter>
      </Fill>
    </PolygonSymbolizer>
  </Rule>
  <Rule>
    <Name>MediumPop</Name>
    <Title>200,000 to 500,000</Title>
    <ogc:Filter>
      <ogc:And>
        <ogc:PropertyIsGreaterThanOrEqualTo>
          <ogc:PropertyName>pop</ogc:PropertyName>
          <ogc:Literal>200000</ogc:Literal>
        </ogc:PropertyIsGreaterThanOrEqualTo>
        <ogc:PropertyIsLessThan>
          <ogc:PropertyName>pop</ogc:PropertyName>
          <ogc:Literal>500000</ogc:Literal>
        </ogc:PropertyIsLessThan>
      </ogc:And>
    </ogc:Filter>
    <PolygonSymbolizer>
      <Fill>
        <CssParameter name="fill">#33CC33</CssParameter>
```

```
35          </Fill>
36        </PolygonSymbolizer>
37      </Rule>
38      <Rule>
39        <Name>LargePop</Name>
40        <Title>Greater Than 500,000</Title>
41        <ogc:Filter>
42          <ogc:PropertyIsGreaterThan>
43            <ogc:PropertyName>pop</ogc:PropertyName>
44            <ogc:Literal>500000</ogc:Literal>
45          </ogc:PropertyIsGreaterThan>
46        </ogc:Filter>
47        <PolygonSymbolizer>
48          <Fill>
49            <CssParameter name="fill">#009900</CssParameter>
50          </Fill>
51        </PolygonSymbolizer>
52      </Rule>
53    </FeatureTypeStyle>
```

Details

Note: Refer to the *Example polygons layer* to see the attributes for the layer. This example has eschewed labels in order to simplify the style, but you can refer to the example *Polygon with styled label* to see which attributes correspond to which polygons.

Each polygon in our fictional country has a population that is represented by the population ("pop") attribute. This style contains three rules that alter the fill based on the value of "pop" attribute, with smaller values yielding a lighter color and larger values yielding a darker color.

The three rules are designed as follows:

Rule order	**Rule name**	**Population** ("pop")	**Color**
1	SmallPop	Less than 200,000	`#66FF66`
2	MediumPop	200,000 to 500,000	`#33CC33`
3	LargePop	Greater than 500,000	`#009900`

The order of the rules does not matter in this case, since each shape is only rendered by a single rule.

The first rule, on **lines 2-16**, specifies the styling of polygons whose population attribute is less than 200,000. **Lines 5-10** set this filter, with **lines 6-9** setting the "less than" filter, **line 7** denoting the attribute ("pop"), and **line 8** the value of 200,000. The color of the polygon fill is set to a light green (`#66FF66`) on **line 13**.

The second rule, on **lines 17-37**, is similar, specifying a style for polygons whose population attribute is greater than or equal to 200,000 but less than 500,000. The filter is set on **lines 20-31**. This filter is longer than in the first rule because two criteria need to be specified instead of one: a "greater than or equal to" and a "less than" filter. Notice the `And` on **line 21** and **line 30**. This mandates that both filters need to be true for the rule to be applicable. The color of the polygon fill is set to a medium green on (`#33CC33`) on **line 34**.

The third rule, on **lines 38-52**, specifies a style for polygons whose population attribute is greater than or equal to 500,000. The filter is set on **lines 41-46**. The color of the polygon fill is the only other difference in this rule, which is set to a dark green (`#009900`) on **line 49**.

Zoom-based polygon

This example alters the style of the polygon at different zoom levels.

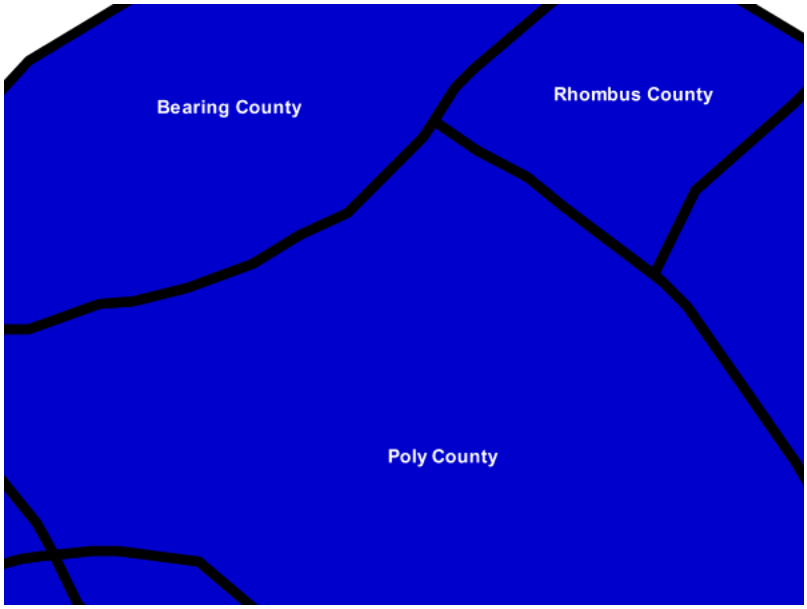


Figure 12.38: *Zoom-based polygon: Zoomed in*

Code

View and download the full "Zoom-based polygon" SLD

```
   <FeatureTypeStyle>
     <Rule>
       <Name>Large</Name>
       <MaxScaleDenominator>100000000</MaxScaleDenominator>
       <PolygonSymbolizer>
         <Fill>
           <CssParameter name="fill">#0000CC</CssParameter>
         </Fill>
         <Stroke>
           <CssParameter name="stroke">#000000</CssParameter>
           <CssParameter name="stroke-width">7</CssParameter>
         </Stroke>
       </PolygonSymbolizer>
       <TextSymbolizer>
         <Label>
           <ogc:PropertyName>name</ogc:PropertyName>
         </Label>
         <Font>
           <CssParameter name="font-family">Arial</CssParameter>
           <CssParameter name="font-size">14</CssParameter>
           <CssParameter name="font-style">normal</CssParameter>
           <CssParameter name="font-weight">bold</CssParameter>
         </Font>
         <LabelPlacement>
```

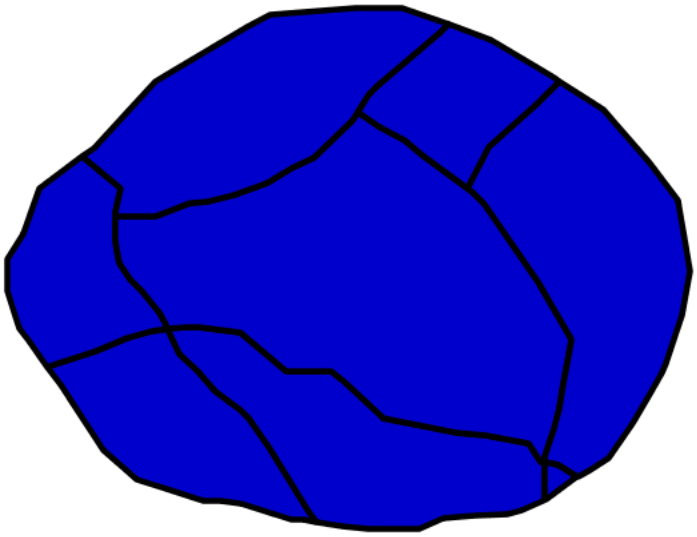


Figure 12.39: *Zoom-based polygon: Partially zoomed*

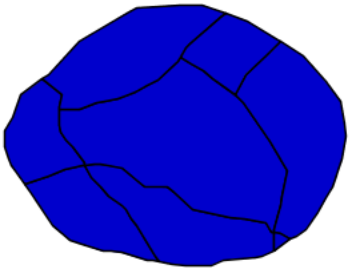


Figure 12.40: *Zoom-based polygon: Zoomed out*

```
            <PointPlacement>
              <AnchorPoint>
                <AnchorPointX>0.5</AnchorPointX>
                <AnchorPointY>0.5</AnchorPointY>
              </AnchorPoint>
            </PointPlacement>
          </LabelPlacement>
          <Fill>
            <CssParameter name="fill">#FFFFFF</CssParameter>
          </Fill>
        </TextSymbolizer>
      </Rule>
      <Rule>
        <Name>Medium</Name>
        <MinScaleDenominator>100000000</MinScaleDenominator>
        <MaxScaleDenominator>200000000</MaxScaleDenominator>
        <PolygonSymbolizer>
          <Fill>
            <CssParameter name="fill">#0000CC</CssParameter>
          </Fill>
          <Stroke>
            <CssParameter name="stroke">#000000</CssParameter>
            <CssParameter name="stroke-width">4</CssParameter>
          </Stroke>
        </PolygonSymbolizer>
      </Rule>
      <Rule>
        <Name>Small</Name>
        <MinScaleDenominator>200000000</MinScaleDenominator>
        <PolygonSymbolizer>
          <Fill>
            <CssParameter name="fill">#0000CC</CssParameter>
          </Fill>
          <Stroke>
            <CssParameter name="stroke">#000000</CssParameter>
            <CssParameter name="stroke-width">1</CssParameter>
          </Stroke>
        </PolygonSymbolizer>
      </Rule>
    </FeatureTypeStyle>
```

Details

It is often desirable to make shapes larger at higher zoom levels when creating a natural-looking map. This example varies the thickness of the lines according to the zoom level. Polygons already do this by nature of being two dimensional, but another way to adjust styling of polygons based on zoom level is to adjust the thickness of the stroke (to be larger as the map is zoomed in) or to limit labels to only certain zoom levels. This is ensures that the size and quantity of strokes and labels remains legible and doesn't overshadow the polygons themselves.

Zoom levels (or more accurately, scale denominators) refer to the scale of the map. A scale denominator of 10,000 means the map has a scale of 1:10,000 in the units of the map projection.

Note: Determining the appropriate scale denominators (zoom levels) to use is beyond the scope of this example.

This style contains three rules, defined as follows:

Rule order	Rule name	Scale denominator	Stroke width	Label display?
1	Large	1:100,000,000 or less	7	Yes
2	Medium	1:100,000,000 to 1:200,000,000	4	No
3	Small	Greater than 1:200,000,000	2	No

The first rule, on **lines 2-36**, is for the smallest scale denominator, corresponding to when the view is "zoomed in". The scale rule is set on **line 40** such that the rule will apply only where the scale denominator is 100,000,000 or less. **Line 7** defines the fill as blue (`#0000CC`). Note that the fill is kept constant across all rules regardless of the scale denominator. As in the *Polygon with default label* or *Polygon with styled label* examples, the rule also contains a `<TextSymbolizer>` at **lines 14-35** for drawing a text label on top of the polygon. **Lines 19-22** set the font information to be Arial, 14 pixels, and bold with no italics. The label is centered both horizontally and vertically along the centroid of the polygon on by setting `<AnchorPointX>` and `<AnchorPointY>` to both be 0.5 (or 50%) on **lines 27-28**. Finally, the color of the font is set to white (`#FFFFFF`) in **line 33**.

The second rule, on **lines 37-50**, is for the intermediate scale denominators, corresponding to when the view is "partially zoomed". The scale rules on **lines 39-40** set the rule such that it will apply to any map with a scale denominator between 100,000,000 and 200,000,000. (The `<MinScaleDenominator>` is inclusive and the `<MaxScaleDenominator>` is exclusive, so a zoom level of exactly 200,000,000 would *not* apply here.) Aside from the scale, there are two differences between this rule and the first: the width of the stroke is set to 4 pixels on **line 47** and a `<TextSymbolizer>` is not present so that no labels will be displayed.

The third rule, on **lines 51-63**, is for the largest scale denominator, corresponding to when the map is "zoomed out". The scale rule is set on **line 53** such that the rule will apply to any map with a scale denominator of 200,000,000 or greater. Again, the only differences between this rule and the others are the width of the lines, which is set to 1 pixel on **line 60**, and the absence of a `<TextSymbolizer>` so that no labels will be displayed.

The resulting style produces a polygon stroke that gets larger as one zooms in and labels that only display when zoomed in to a sufficient level.

12.3.4 Rasters

Rasters are geographic data displayed in a grid. They are similar to image files such as PNG files, except that instead of each point containing visual information, each point contains geographic information in numerical form. Rasters can be thought of as a georeferenced table of numerical values.

One example of a raster is a Digital Elevation Model (DEM) layer, which has elevation data encoded numerically at each georeferenced data point.

> **Warning:** The code examples shown on this page are **not the full SLD code**, as they omit the SLD header and footer information for the sake of brevity. Please use the links to download the full SLD for each example.

Example raster

The `raster layer` that is used in the examples below contains elevation data for a fictional world. The data is stored in EPSG:4326 (longitude/latitude) and has a data range from 70 to 256. If rendered in grayscale, where minimum values are colored black and maximum values are colored white, the raster would look like this:

`Download the raster shapefile`

Figure 12.41: *Raster file as rendered in grayscale*

Two-color gradient

This example shows a two-color style with green at lower elevations and brown at higher elevations.

Figure 12.42: *Two-color gradient*

Code

View and download the full "Two-color gradient" SLD

```
1    <FeatureTypeStyle>
2      <Rule>
3        <RasterSymbolizer>
4          <ColorMap>
5            <ColorMapEntry color="#008000" quantity="70" />
6            <ColorMapEntry color="#663333" quantity="256" />
7          </ColorMap>
8        </RasterSymbolizer>
9      </Rule>
10   </FeatureTypeStyle>
```

Details

There is one `<Rule>` in one `<FeatureTypeStyle>` for this example, which is the simplest possible situation. All subsequent examples will share this characteristic. Styling of rasters is done via the `<RasterSymbolizer>` tag (**lines 3-8**).

This example creates a smooth gradient between two colors corresponding to two elevation values. The gradient is created via the `<ColorMap>` on **lines 4-7**. Each entry in the `<ColorMap>` represents one entry or anchor in the gradient. **Line 5** sets the lower value of 70 via the `quantity` parameter, which is styled a dark green (`#008000`). **Line 6** sets the upper value of 256 via the `quantity` parameter again, which is styled a dark brown (`#663333`). All data values in between these two quantities will be linearly interpolated: a value of 163 (the midpoint between 70 and 256) will be colored as the midpoint between the two colors (in this case approximately `#335717`, a muddy green).

Transparent gradient

This example creates the same two-color gradient as in the *Two-color gradient* as in the example above but makes the entire layer mostly transparent by setting a 30% opacity.

Figure 12.43: *Transparent gradient*

Code

View and download the full "Transparent gradient" SLD

```
1    <FeatureTypeStyle>
2      <Rule>
3        <RasterSymbolizer>
4          <Opacity>0.3</Opacity>
5          <ColorMap>
6            <ColorMapEntry color="#008000" quantity="70" />
7            <ColorMapEntry color="#663333" quantity="256" />
8          </ColorMap>
9        </RasterSymbolizer>
10     </Rule>
11   </FeatureTypeStyle>
```

Details

This example is similar to the *Two-color gradient* example save for the addition of **line 4**, which sets the opacity of the layer to 0.3 (or 30% opaque). An opacity value of 1 means that the shape is drawn 100% opaque, while an opacity value of 0 means that the shape is rendered as completely transparent. The value of 0.3 means that the the raster partially takes on the color and style of whatever is drawn beneath it. Since the background is white in this example, the colors generated from the `<ColorMap>` look lighter, but were the raster imposed on a dark background the resulting colors would be darker.

Brightness and contrast

This example normalizes the color output and then increases the brightness by a factor of 2.

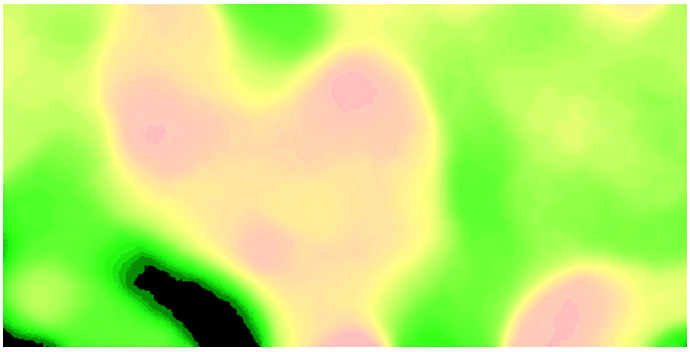


Figure 12.44: *Brightness and contrast*

Code

View and download the full "Brightness and contrast" SLD

```
1    <FeatureTypeStyle>
2      <Rule>
3        <RasterSymbolizer>
4          <ContrastEnhancement>
5            <Normalize />
6            <GammaValue>0.5</GammaValue>
7          </ContrastEnhancement>
8          <ColorMap>
9            <ColorMapEntry color="#008000" quantity="70" />
10           <ColorMapEntry color="#663333" quantity="256" />
11         </ColorMap>
12       </RasterSymbolizer>
13     </Rule>
14   </FeatureTypeStyle>
```

Details

This example is similar to the *Two-color gradient*, save for the addition of the `<ContrastEnhancement>` tag on **lines 4-7**. **Line 5** normalizes the output by increasing the contrast to its maximum extent. **Line 6** then adjusts the brightness by a factor of 0.5. Since values less than 1 make the output brighter, a value of 0.5 makes the output twice as bright.

As with previous examples, **lines 8-11** determine the `<ColorMap>`, with **line 9** setting the lower bound (70) to be colored dark green (`#008000`) and **line 10** setting the upper bound (256) to be colored dark brown (`#663333`).

Three-color gradient

This example creates a three-color gradient in primary colors. In addition, the gradient doesn't span the entire range of data values, leading some data not to be rendered at all.

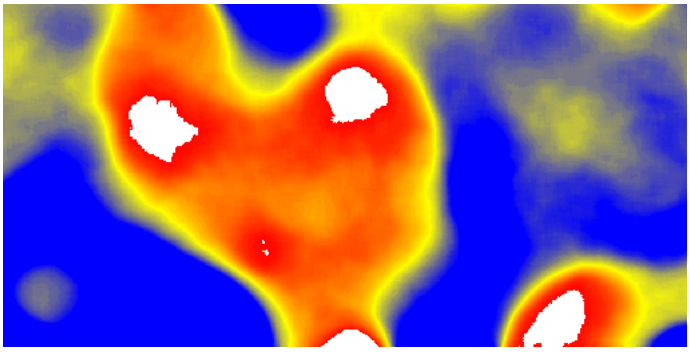


Figure 12.45: *Three-color gradient*

Code

View and download the full "Three-color gradient" SLD

```
1    <FeatureTypeStyle>
2      <Rule>
3        <RasterSymbolizer>
4          <ColorMap>
5            <ColorMapEntry color="#0000FF" quantity="150" />
6            <ColorMapEntry color="#FFFF00" quantity="200" />
7            <ColorMapEntry color="#FF0000" quantity="250" />
8          </ColorMap>
9        </RasterSymbolizer>
10     </Rule>
11   </FeatureTypeStyle>
```

Details

This example creates a three-color gradient based on a `<ColorMap>` with three entries on **lines 4-8**: **line 5** specifies the lower bound (150) be styled in blue (`#0000FF`), **line 6** specifies an intermediate point (200) be styled in yellow (`#FFFF00`), and **line 7** specifies the upper bound (250) be styled in red (`#FF0000`).

Since our data values run between 70 and 256, some data points are not accounted for in this style. Those values below the lowest entry in the color map (the range from 70 to 149) are styled the same color as the lower bound, in this case blue. On the other hand, values above the upper bound in the color map (the range from 251 to 256) are not rendered at all.

Alpha channel

This example creates an "alpha channel" effect such that higher values are increasingly transparent.

Code

View and download the full "Alpha channel" SLD

```
1    <FeatureTypeStyle>
2      <Rule>
3        <RasterSymbolizer>
4          <ColorMap>
```

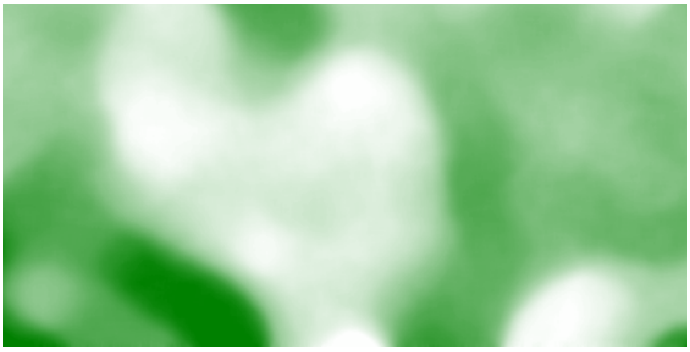


Figure 12.46: *Alpha channel*

```
5           <ColorMapEntry color="#008000" quantity="70" />
6           <ColorMapEntry color="#008000" quantity="256" opacity="0"/>
7         </ColorMap>
8       </RasterSymbolizer>
9     </Rule>
10  </FeatureTypeStyle>
```

Details

An alpha channel is another way of referring to variable transparency. Much like how a gradient maps values to colors, each entry in a `<ColorMap>` can have a value for opacity (with the default being 1.0 or completely opaque).

In this example, there is a `<ColorMap>` with two entries: **line 5** specifies the lower bound of 70 be colored dark green (`#008000`), while **line 6** specifies the upper bound of 256 also be colored dark green but with an opacity value of 0. This means that values of 256 will be rendered at 0% opacity (entirely transparent). Just like the gradient color, the opacity is also linearly interpolated such that a value of 163 (the midpoint between 70 and 256) is rendered at 50% opacity.

Discrete colors

This example shows a gradient that is not linearly interpolated but instead has values mapped precisely to one of three specific colors.

Note: This example leverages an SLD extension in GeoServer. Discrete colors are not part of the standard SLD 1.0 specification.

Code

View and download the full "Discrete colors" SLD

```
1   <FeatureTypeStyle>
2     <Rule>
3       <RasterSymbolizer>
4         <ColorMap type="intervals">
5           <ColorMapEntry color="#008000" quantity="150" />
6           <ColorMapEntry color="#663333" quantity="256" />
```

Figure 12.47: *Discrete colors*

```
7           </ColorMap>
8         </RasterSymbolizer>
9       </Rule>
10    </FeatureTypeStyle>
```

Details

Sometimes color bands in discrete steps are more appropriate than a color gradient. The `type="intervals"` parameter added to the `<ColorMap>` on **line 4** sets the display to output discrete colors instead of a gradient. The values in each entry correspond to the upper bound for the color band such that colors are mapped to values less than the value of one entry but greater than or equal to the next lower entry. For example, **line 5** colors all values less than 150 to dark green (`#008000`) and line 6 colors all values less than 256 but greater than or equal to 150 to dark brown (`#663333`).

Many color gradient

This example shows a gradient interpolated across eight different colors.

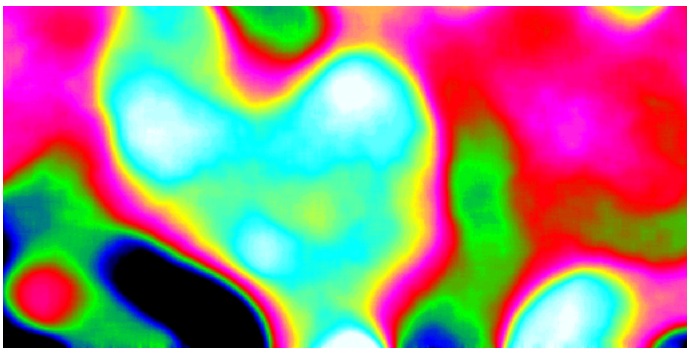


Figure 12.48: *Many color gradient*

Code

`View and download the full "Many color gradient" SLD`

```
1   <FeatureTypeStyle>
2     <Rule>
3       <RasterSymbolizer>
4         <ColorMap>
5           <ColorMapEntry color="#000000" quantity="95" />
6           <ColorMapEntry color="#0000FF" quantity="110" />
7           <ColorMapEntry color="#00FF00" quantity="135" />
8           <ColorMapEntry color="#FF0000" quantity="160" />
9           <ColorMapEntry color="#FF00FF" quantity="185" />
10          <ColorMapEntry color="#FFFF00" quantity="210" />
11          <ColorMapEntry color="#00FFFF" quantity="235" />
12          <ColorMapEntry color="#FFFFFF" quantity="256" />
13        </ColorMap>
14      </RasterSymbolizer>
15    </Rule>
16  </FeatureTypeStyle>
```

Details

A `<ColorMap>` can include up to 255 `<ColorMapEntry>` elements. This example has eight entries (**lines 4-13**):

Entry number	Value	Color	RGB code
1	95	Black	`#000000`
2	110	Blue	`#0000FF`
3	135	Green	`#00FF00`
4	160	Red	`#FF0000`
5	185	Purple	`#FF00FF`
6	210	Yellow	`#FFFF00`
7	235	Cyan	`#00FFFF`
8	256	White	`#FFFFFF`

12.4 SLD Reference

The OGC **Styled Layer Descriptor (SLD)** standard defines a language for expressing styling of geospatial data. GeoServer uses SLD as its primary styling language.

SLD 1.0.0 is defined in the following specification:

- OGC Styled Layer Descriptor Implementation Specification, Version 1.0.0

Subsequently the functionality of SLD has been split into two specifications:

- OGC Symbology Encoding Implementation Specification, Version 1.1.0
- OGC Styled Layer Descriptor profile of the Web Map Service Implementation Specification, Version 1.1.0

GeoServer implements the SLD 1.0.0 standard, as well as some parts of the SE 1.1.0 and WMS-SLD 1.1.0 standards.

Elements of SLD

The following sections describe the SLD elements implemented in GeoServer.

The root element for an SLD is `<StyledLayerDescriptor>`. It contains a **Layers** and **Styles** elements which describe how a map is to be composed and styled.

12.4.1 StyledLayerDescriptor

The root element for an SLD is `<StyledLayerDescriptor>`. It contains a sequence of *Layers* defining the styled map content.

The `<StyledLayerDescriptor>` element contains the following elements:

Tag	Required?	Description
`<NamedLayer>`	0..N	A reference to a named layer in the server catalog
`<UserLayer>`	0..N	A layer defined in the style itself

12.4.2 Layers

An SLD document contains a sequence of layer definitions indicating the layers to be styled. Each layer definition is either a **NamedLayer** reference or a supplied **UserLayer**.

NamedLayer

A **NamedLayer** specifies an existing layer to be styled, and the styling to apply to it. The styling may be any combination of catalog styles and explicitly-defined styles. If no style is specified, the default style for the layer is used.

The `<NamedLayer>` element contains the following elements:

Tag	Required?	Description
`<Name>`	Yes	The name of the layer to be styled. (Ignored in catalog styles.)
`<Description>`	No	The description for the layer.
`<NamedStyle>`	0..N	The name of a catalog style to apply to the layer.
`<UserStyle>`	0..N	The definition of a style to apply to the layer. See *Styles*

UserLayer

A **UserLayer** defines a new layer to be styled, and the styling to apply to it. The data for the layer is provided directly in the layer definition using the `<InlineFeature>` element. Since the layer is not known to the server, the styling must be explicitly specified as well.

The `<UserLayer>` element contains the following elements:

Tag	Required?	Description
`<Name>`	No	The name for the layer being defined
`<Description>`	No	The description for the layer
`<InlineFeature>`	No	One or more feature collections providing the layer data, specified using GML.
`<UserStyle>`	1..N	The definition of the style(s) to use for the layer. See *Styles*

A common use is to define a geometry to be rendered to indicate an Area Of Interest.

InlineFeature

An **InlineFeature** element contains data defining a layer to be styled. The element contains one or more `<FeatureCollection>` elements defining the data. Each Feature Collection can contain any number of `<featureMember>` elements, each containing a feature specified using GML markup. The features can

contain any type of geometry (point, line or polygon, and collections of these). They may also contain scalar-valued attributes, which can be useful for labelling.

Example

The following style specifies a named layer using the default style, and a user-defined layer with inline data and styling. It displays the US States layer, with a labelled red box surrounding the Pacific NW.

```
<sld:StyledLayerDescriptor xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.opengis.net/sld StyledLayerDescriptor.xsd"
   xmlns:sld="http://www.opengis.net/sld" version="1.0.0">
   <sld:NamedLayer>
      <sld:Name>usa:states</sld:Name>
   </sld:NamedLayer>
   <sld:UserLayer>
      <sld:Name>Inline</sld:Name>
      <sld:InlineFeature>
         <sld:FeatureCollection>
            <sld:featureMember>
              <feature>
                <geometryProperty>
                  <gml:Polygon>
                     <gml:outerBoundaryIs>
                        <gml:LinearRing>
                           <gml:coordinates>
          -127.0,51.0 -110.0,51.0 -110.0,41.0 -127.0,41.0 -127.0,51.0
                           </gml:coordinates>
                        </gml:LinearRing>
                     </gml:outerBoundaryIs>
                  </gml:Polygon>
                </geometryProperty>
                <title>Pacific NW </title>
              </feature>
            </sld:featureMember>
         </sld:FeatureCollection>
      </sld:InlineFeature>
      <sld:UserStyle>
         <sld:FeatureTypeStyle>
            <sld:Rule>
                  <sld:PolygonSymbolizer>
                <Stroke>
                  <CssParameter name="stroke">#FF0000</CssParameter>
                  <CssParameter name="stroke-width">2</CssParameter>
                </Stroke>
              </sld:PolygonSymbolizer>
              <sld:TextSymbolizer>
                <sld:Label>
                  <ogc:PropertyName>title</ogc:PropertyName>
                </sld:Label>
                <sld:Fill>
                  <sld:CssParameter name="fill">#FF0000</sld:CssParameter>
                </sld:Fill>
              </sld:TextSymbolizer>
            </sld:Rule>
         </sld:FeatureTypeStyle>
      </sld:UserStyle>
   </sld:UserLayer>
```

```
</sld:StyledLayerDescriptor>
```

12.4.3 Styles

The style elements specify the styling to be applied to a layer.

UserStyle

The **UserStyle** element defines styling for a layer.

The `<UserStyle>` element contains the following elements:

Tag	Required?	Description
`<Name>`	No	The name of the style, used to reference it externally. (Ignored for catalog styles.)
`<Title>`	No	The title of the style.
`<Abstract>`	No	The description for the style.
`<IsDefault>`	No	Whether the style is the default one for a named layer. Used in SLD **Library Mode**. Values are `1` or `0` (default).
`<FeatureTypeStyle>`	1..N	Defines the symbology for rendering a single feature type.

FeatureTypeStyle

The **FeatureTypeStyle** element specifies the styling that is applied to a single feature type of a layer. It contains a list of rules which determine the symbology to be applied to each feature of a layer.

The `<FeatureTypeStyle>` element contains the following elements:

Tag	Required?	Description
`<Name>`	No	Not used at present
`<Title>`	No	The title for the style.
`<Abstract>`	No	The description for the style.
`<FeatureTypeName>`	No	Identifies the feature type the style is to be applied to. Omitted if the style applies to all features in a layer.
`<Rule>`	1..N	A styling rule to be evaluated. See *Rules*

Usually a layer contains only a single feature type, so the `<FeatureTypeName>` is omitted.

Any number of `<FeatureTypeStyle>` elements can be specified in a style. In GeoServer each one is rendered into a separate image buffer. After all features are rendered the buffers are composited to form the final layer image. The compositing is done in the order the FeatureTypeStyles are given in the SLD, with the first one on the bottom (the "Painter's Model"). This effectively creates "virtual layers", which can be used to achieve styling effects such as cased lines.

Styles contain **Rules** and **Filters** to determine sets of features to be styled with specific symbology. Rules may also specify the scale range in which the feature styling is visible.

12.4.4 Rules

Styling **rules** define the portrayal of features. A rule combines a *filter* with any number of symbolizers. Features for which the filter condition evaluates as true are rendered using the the symbolizers in the rule.

Syntax

The `<Rule>` element contains the following elements:

Tag	Required?	Description
`<Name>`	No	Specifies a name for the rule.
`<Title>`	No	Specifies a title for the rule. The title is used in display lists and legends.
`<Abstract>`	No	Specifies an abstract describing the rule.
`<Filter>`	No	Specifies a filter controlling when the rule is applied. See *Filters*
`<MinScaleDenominator>`	No	Specifies the minimum scale denominator (inclusive) for the scale range in which this rule applies. If present, the rule applies at the given scale and all smaller scales.
`<MaxScaleDenominator>`	No	Specifies the maximum scale denominator (exclusive) for the scale range in which this rule applies. If present, the rule applies at scales larger than the given scale.
`<PointSymbolizer>`	0..N	Specifies styling as points. See *PointSymbolizer*
`<LineSymbolizer>`	0..N	Specifies styling as lines. See *LineSymbolizer*
`<PolygonSymbolizer>`	0..N	Specifies styling as polygons. See *PolygonSymbolizer*
`<TextSymbolizer>`	0..N	Specifies styling for text labels. See *TextSymbolizer*
`<RasterSymbolizer>`	0..N	Specifies styling for raster data. See *RasterSymbolizer*

Scale Selection

Rules support **scale selection** to allow specifying the scale range in which a rule may be applied (assuming the filter condition is satisfied as well, if present). Scale selection allows for varying portrayal of features at different map scales. In particular, at smaller scales it is common to use simpler styling for features, or even prevent the display of some features altogether.

Scale ranges are specified by using **scale denominators**. These values correspond directly to the ground distance covered by a map, but are inversely related to the common "large" and "small" terminology for map scale. In other words:

- **large scale** maps cover *less* area and have a *smaller* scale denominator
- **small scale** maps cover *more* area and have a *larger* scale denominator

Two optional elements specify the scale range for a rule:

Tag	Required?	Description
`<MinScaleDenominator>`	No	Specifies the minimum scale denominator (inclusive) for the scale range in which this rule applies. If present, the rule applies at the given scale and all smaller scales.
`<MaxScaleDenominator>`	No	Specifies the maximum scale denominator (exclusive) for the scale range in which this rule applies. If present, the rule applies at scales larger than the given scale.

Note: The current scale can also be obtained via the `wms_scale_denominator` *SLD environment variable*. This allows including scale dependency in *Filter Expressions*.

The following example shows the use of scale selection in a pair of rules. The rules specify that:

- at scales **above** 1:20,000 (*larger* scales, with scale denominators *smaller* than 20,000) features are symbolized with 10-pixel red squares,

- at scales **at or below** 1:20,000 (*smaller* scales, with scale denominators *larger* than 20,000) features are symbolized with 4-pixel blue triangles.

```
<Rule>
   <MaxScaleDenominator>20000</MaxScaleDenominator>
   <PointSymbolizer>
     <Graphic>
       <Mark>
         <WellKnownName>square</WellKnownName>
         <Fill><CssParameter name="fill">#FF0000</CssParameter>
       </Mark>
       <Size>10</Size>
     </Graphic>
   </PointSymbolizer>
</Rule>
<Rule>
   <MinScaleDenominator>20000</MinScaleDenominator>
   <PointSymbolizer>
     <Graphic>
       <Mark>
         <WellKnownName>triangle</WellKnownName>
         <Fill><CssParameter name="fill">#0000FF</CssParameter>
       </Mark>
       <Size>4</Size>
     </Graphic>
   </PointSymbolizer>
</Rule>
```

Evaluation Order

Within an SLD document each `<FeatureTypeStyle>` can contain many rules. Multiple-rule SLDs are the basis for thematic styling. In GeoServer each `<FeatureTypeStyle>` is evaluated once for each feature processed. The rules within it are evaluated in the order they occur. A rule is applied when its filter condition (if any) is true for a feature and the rule is enabled at the current map scale. The rule is applied by rendering the feature using each symbolizer within the rule, in the order in which they occur. The rendering is performed into the image buffer for the parent `<FeatureTypeStyle>`. Thus symbolizers earlier in a `FeatureTypeStyle` and `Rule` are rendered *before* symbolizers occuring later in the document (this is the "Painter's Model" method of rendering).

Examples

The following rule applies only to features which have a `POPULATION` attribute greater than 100,000, and symbolizes the features as red points.

```
<Rule>
   <ogc:Filter>
     <ogc:PropertyIsGreaterThan>
       <ogc:PropertyName>POPULATION</ogc:PropertyName>
       <ogc:Literal>100000</ogc:Literal>
     </ogc:PropertyIsGreaterThan>
   </ogc:Filter>
   <PointSymbolizer>
     <Graphic>
       <Mark>
         <Fill><CssParameter name="fill">#FF0000</CssParameter>
       </Mark>
```

```
      </Graphic>
    </PointSymbolizer>
</Rule>
```

An additional rule can be added which applies to features whose `POPULATION` attribute is less than 100,000, and symbolizes them as green points.

```
<Rule>
  <ogc:Filter>
    <ogc:PropertyIsLessThan>
      <ogc:PropertyName>POPULATION</ogc:PropertyName>
      <ogc:Literal>100000</ogc:Literal>
    </ogc:PropertyIsLessThan>
  </ogc:Filter>
  <PointSymbolizer>
    <Graphic>
      <Mark>
        <Fill><CssParameter name="fill">#0000FF</CssParameter>
      </Mark>
    </Graphic>
  </PointSymbolizer>
</Rule>
```

12.4.5 Filters

A *filter* is the mechanism in SLD for specifying conditions. They are similar in functionality to the SQL "WHERE" clause. Filters are used within *Rules* to determine which styles should be applied to which features in a data set. The filter language used by SLD follows the OGC Filter Encoding standard. It is described in detail in the *Filter Encoding Reference*.

A filter condition is specified by using a **comparison operator** or a **spatial operator**, or two or more of these combined by **logical operators**. The operators are usually used to compare properties of the features being filtered to other properties or to literal data.

Comparison operators

Comparison operators are used to specify conditions on the non-spatial attributes of a feature. The following **binary comparison operators** are available:

- `<PropertyIsEqualTo>`
- `<PropertyIsNotEqualTo>`
- `<PropertyIsLessThan>`
- `<PropertyIsLessThanOrEqualTo>`
- `<PropertyIsGreaterThan>`
- `<PropertyIsGreaterThanOrEqualTo>`

These operators contain two *filter expressions* to be compared. The first operand is often a `<PropertyName>`, but both operands may be any expression, function or literal value.

Binary comparison operators may include a `matchCase` attribute with the value `true` or `false`. If this attribute is `true` (which is the default), string comparisons are case-sensitive. If the attribute is specified and has the value `false` strings comparisons do not check case.

Other available **value comparison operators** are:

- <PropertyIsLike>
- <PropertyIsNull>
- <PropertyIsBetween>

<PropertyIsLike> matches a string property value against a text **pattern**. It contains a <PropertyName> element containing the name of the property containing the string to be matched and a <Literal> element containing the pattern. The pattern is specified by a sequence of regular characters and three special pattern characters. The pattern characters are defined by the following required attributes of the <PropertyIsLike> element:

- wildCard specifies a pattern character which matches any sequence of zero or more characters
- singleChar specifies a pattern character which matches any single character
- escapeChar specifies an escape character which can be used to escape these pattern characters

<PropertyIsNull> tests whether a property value is null. It contains a single <PropertyName> element containing the name of the property containing the value to be tested.

<PropertyIsBetween> tests whether an expression value lies within a range. It contains a *filter expression* providing the value to test, followed by the elements <LowerBoundary> and <UpperBoundary>, each containing a *filter expression*.

Examples

- The following filter selects features whose NAME attribute has the value of "New York":

```
<PropertyIsEqualTo>
   <PropertyName>NAME</PropertyName>
   <Literal>New York</Literal>
</PropertyIsEqualTo>
```

- The following filter selects features whose geometry area is greater than 1,000,000:

```
<PropertyIsGreaterThan>
   <ogc:Function name="area">
     <PropertyName>GEOMETRY</PropertyName>
   </ogc:Function>
   <Literal>1000000</Literal>
</PropertyIsEqualTo>
```

Spatial operators

Spatial operators are used to specify conditions on the geometric attributes of a feature. The following spatial operators are available:

Topological Operators

These operators test topological spatial relationships using the standard OGC Simple Features predicates:

- <Intersects>
- <Equals>
- <Disjoint>
- <Touches>
- <Within>

- <Overlaps>
- <Crosses>
- <Intersects>
- <Contains>

The content for these operators is a <PropertyName> element for a geometry-valued property and a GML geometry literal.

Distance Operators

These operators compute distance relationships between geometries:

- <DWithin>
- <Beyond>

The content for these elements is a <PropertyName> element for a geometry-valued property, a GML geometry literal, and a <Distance> element containing the value for the distance tolerance. The <Distance> element may include an optional units attribute.

Bounding Box Operator

This operator tests whether a feature geometry attribute intersects a given bounding box:

- <BBOX>

The content is an optional <PropertyName> element, and a GML envelope literal. If the PropertyName is omitted the default geometry attribute is assumed.

Examples

- The following filter selects features with a geometry that intersects the point (1,1):

```
<Intersects>
   <PropertyName>GEOMETRY</PropertyName>
   <Literal>
      <gml:Point>
         <gml:coordinates>1 1</gml:coordinates>
      </gml:Point>
   </Literal>
</Intersects>
```

- The following filter selects features with a geometry that intersects the box [-10,0 : 10,10]:

```
<ogc:BBOX>
  <ogc:PropertyName>GEOMETRY</ogc:PropertyName>
  <gml:Box srsName="urn:x-ogc:def:crs:EPSG:4326">
    <gml:coord>
      <gml:X>-10</gml:X> <gml:Y>0</gml:Y>
    </gml:coord>
    <gml:coord>
      <gml:X>10</gml:X> <gml:Y>10</gml:Y>
    </gml:coord>
  </gml:Box>
</ogc:BBOX>
```

Logical operators

Logical operators are used to create logical combinations of other filter operators. They may be nested to any depth. The following logical operators are available:

- `<And>`
- `<Or>`
- `<Not>`

The content for `<And>` and `<Or>` is two filter operator elements. The content for `<Not>` is a single filter operator element.

Examples

- The following filter uses `<And>` to combine a comparison operator and a spatial operator:

```
<And>
   <PropertyIsEqualTo>
      <PropertyName>NAME</PropertyName>
      <Literal>New York</Literal>
   </PropertyIsEqualTo>
   <Intersects>
      <PropertyName>GEOMETRY</PropertyName>
      <Literal>
         <gml:Point>
             <gml:coordinates>1 1</gml:coordinates>
         </gml:Point>
      </Literal>
   </Intersects>
</And>
```

Filter Expressions

Filter expressions allow performing computation on data values. The following elements can be used to form expressions.

Arithmetic Operators

These operators perform arithmetic on numeric values. Each contains two expressions as sub-elements.

- `<Add>`
- `<Sub>`
- `<Mul>`
- `<Div>`

Functions

The `<Function>` element specifies a filter function to be evaluated. The `name` attribute gives the function name. The element contains a sequence of zero or more filter expressions providing the function arguments. See the *Filter Function Reference* for details of the functions provided by GeoServer.

Feature Property Values

The `<PropertyName>` element allows referring to the value of a given feature attribute. It contains a string specifying the attribute name.

Literals

The <Literal> element allows specifying constant values of numeric, boolean, string, date or geometry type.

Rules contain **Symbolizers** to specify how features are styled. There are 5 types of symbolizers:

- PointSymbolizer, which styles features as **points**
- LineSymbolizer, which styles features as **lines**
- PolygonSymbolizer, which styles features as **polygons**
- TextSymbolizer, which styles **text labels** for features
- RasterSymbolizer, which styles **raster coverages**

Each symbolizer type has its own parameters to control styling.

12.4.6 PointSymbolizer

A **PointSymbolizer** styles features as **points**. Points are depicted as graphic symbols at a single location on the map.

Syntax

A <PointSymbolizer> contains an optional <Geometry> element, and a required <Graphic> element specifying the point symbology.

Tag	Required?	Description
<Geometry>	No	Specifies the geometry to be rendered.
<Graphic>	Yes	Specifies the styling for the point symbol.

Geometry

The <Geometry> element is optional. If present, it specifies the featuretype property from which to obtain the geometry to style using a <PropertyName> element. See also *Geometry transformations in SLD* for GeoServer extensions for specifying geometry.

Any kind of geometry may be styled with a <PointSymbolizer>. For non-point geometries, a representative point is used (such as the centroid of a line or polygon).

Graphic

Symbology is specified using a <Graphic> element. The symbol is specified by either an <ExternalGraphic> or a <Mark> element. **External Graphics** are image files (in a format such as PNG or SVG) that contain the shape and color information defining how to render a symbol. **Marks** are vector shapes whose stroke and fill are defined explicitly in the symbolizer.

There are five possible sub-elements of the <Graphic> element. One of <ExternalGraphic> or <Mark> must be specified; the others are optional.

Tag	Required?	Description
`<ExternalGraphic>`	No (when using `<Mark>`)	Specifies an external image file to use as the symbol.
`<Mark>`	No (when using `<ExternalGraphic>`)	Specifies a named shape to use as the symbol.
`<Opacity>`	No	Specifies the opacity (transparency) of the symbol. Values range from `0` (completely transparent) to `1` (completely opaque). Value may contain *expressions*. Default is `1` (opaque).
`<Size>`	No	Specifies the size of the symbol, in pixels. When used with an image file, this specifies the height of the image, with the width being scaled accordingly. if omitted the native symbol size is used. Value may contain *expressions*.
`<Rotation>`	No	Specifies the rotation of the symbol about its center point, in decimal degrees. Positive values indicate rotation in the clockwise direction, negative values indicate counter-clockwise rotation. Value may contain *expressions*. Default is `0`.

ExternalGraphic

External Graphics are image files (in formats such as PNG or SVG) that contain the shape and color information defining how to render a symbol. For GeoServer extensions for specifying external graphics, see *Graphic symbology in GeoServer*.

The `<ExternalGraphic>` element has the sub-elements:

Tag	Required?	Description
`<OnlineResource>`	Yes	The `xlink:href` attribute specifies the location of the image file. The value can be either a URL or a local pathname relative to the SLD directory. The value can contain CQL expressions delimited by `${ }`. The attribute `xlink:type="simple"` is also required. The element does not contain any content.
`<Format>`	Yes	The MIME type of the image format. Most standard web image formats are supported. Common MIME types are `image/png`, `image/jpeg`, `image/gif`, and `image/svg+xml`

Mark

Marks are predefined vector shapes identified by a well-known name. Their fill and stroke can be defined explicitly in the SLD. For GeoServer extensions for specifying mark symbols, see *Graphic symbology in GeoServer*.

The `<Mark>` element has the sub-elements:

Tag	Required?	Description
<WellKnownName>	Yes	The name of the shape. Standard SLD shapes are `circle`, `square`, `triangle`, `star`, `cross`, or `x`. Default is `square`.
<Fill>	No	Specifies how the symbol should be filled (for closed shapes). Options are to use <CssParameter name="fill"> to specify a solid fill color, or using <GraphicFill> for a tiled graphic fill. See the PolygonSymbolizer *Fill* for the full syntax.
<Stroke>	No	Specifies how the symbol linework should be drawn. Some options are using <CssParameter name="stroke"> to specify a stroke color, or using <GraphicStroke> for a repeated graphic. See the LineSymbolizer *Stroke* for the full syntax.

Example

The following symbolizer is taken from the *Points* section in the *SLD Cookbook*.

```
<PointSymbolizer>
  <Graphic>
    <Mark>
      <WellKnownName>circle</WellKnownName>
      <Fill>
        <CssParameter name="fill">#FF0000</CssParameter>
      </Fill>
    </Mark>
    <Size>6</Size>
  </Graphic>
</PointSymbolizer>
```

The symbolizer contains the required <Graphic> element. Inside this element is the <Mark> element and <Size> element, which are the minimum required element inside <Graphic> (when not using the <ExternalGraphic> element). The <Mark> element contains the <WellKnownName> element and a <Fill> element. No other element are required. In summary, this example specifies the following:

1. Features will be rendered as points
2. Points will be rendered as circles
3. Circles will be rendered with a diameter of 6 pixels and filled with the color red

The next example uses an external graphic loaded from the file system:

```
<PointSymbolizer>
  <Graphic>
    <ExternalGraphic>
      <OnlineResource xlink:type="simple"
                      xlink:href="file:///var/www/htdocs/sun.png" />
      <Format>image.png</Format>
    </ExternalGraphic>
  </Graphic>
</PointSymbolizer>
```

For `file://` URLs, the file must be readable by the user the Geoserver process is running as. You can also use `href://` URLs to reference remote graphics.

Further examples can be found in the *Points* section of the *SLD Cookbook*.

Using expressions in parameter values

Many SLD parameters allow their values to be of **mixed type**. This means that the element content can be:

- a constant value expressed as a string
- a *filter expression*
- any combination of strings and filter expressions.

Using expressions in parameter values provides the ability to determine styling dynamically on a per-feature basis, by computing parameter values from feature properties. Using computed parameters is an alternative to using rules in some situations, and may provide a more compact SLD document.

GeoServer also supports using substitution variables provided in WMS requests. This is described in *Variable substitution in SLD*.

12.4.7 LineSymbolizer

A **LineSymbolizer** styles features as **lines**. Lines are one-dimensional geometries that have both position and length. Each line is comprised of one or more **line segments**, and has either two **ends** or none (if it is closed).

Syntax

A `<LineSymbolizer>` contains an optional `<Geometry>` element, and a required `<Stroke>` element specifying the line symbology.

Tag	Required?	Description
`<Geometry>`	No	Specifies the geometry to be rendered.
`<Stroke>`	Yes	Specifies the styling for the line.

Geometry

The `<Geometry>` element is optional. If present, it specifies the featuretype property from which to obtain the geometry to style using the `PropertyName` element. See also *Geometry transformations in SLD* for GeoServer extensions for specifying geometry.

Any kind of geometry may be styled with a `<LineSymbolizer>`. Point geometries are treated as lines of zero length, with a horizontal orientation. For polygonal geometries the boundary (or boundaries) are used as the lines, each line being a closed ring with no ends.

Stroke

The `<Stroke>` element specifies the styling of a line. There are three elements that can be included inside the `<Stroke>` element.

Tag	Required?	Description
`<GraphicFill>`	No	Renders the pixels of the line with a repeated pattern.
`<GraphicStroke>`	No	Renders the line with a repeated linear graphic.
`<CssParameter>`	0..N	Determines the stroke styling parameters.

GraphicFill

The `<GraphicFill>` element specifies that the pixels of the line are to be filled with a repeating graphic image or symbol. The graphic is specified by a `<Graphic>` sub-element, which is described in the `PointSymbolizer` *Graphic* section.

GraphicStroke

The `<GraphicStroke>` element specifies the the line is to be drawn using a repeated graphic image or symbol following the line. The graphic is specified by a `<Graphic>` sub-element, which is described in the `PointSymbolizer` *Graphic* section.

The spacing of the graphic symbol can be specified using the `<Size>` element in the `<Graphic>` element, or the `<CSSParameter name="stroke-dasharray">` in the `Stroke` element.

CssParameter

The `<CssParameter>` elements describe the basic styling of the line. Any number of `<CssParameter>` elements can be specified.

The `name` **attribute** indicates what aspect of styling an element specifies, using the standard CSS/SVG styling model. The **content** of the element supplies the value of the styling parameter. The value may contain *expressions*.

The following parameters are supported:

Parameter	Required?	Description
`name="stroke"`	No	Specifies the solid color given to the line, in the form `#RRGGBB`. Default is black (`#000000`).
`name="stroke-width"`	No	Specifies the width of the line in pixels. Default is `1`.
`name="stroke-opacity"`	No	Specifies the opacity (transparency) of the line. The value is a number are between `0` (completely transparent) and `1` (completely opaque). Default is `1`.
`name="stroke-linejoin"`	No	Determines how lines are rendered at intersections of line segments. Possible values are `mitre` (sharp corner), `round` (rounded corner), and `bevel` (diagonal corner). Default is `mitre`.
`name="stroke-linecap"`	No	Determines how lines are rendered at their ends. Possible values are `butt` (sharp square edge), `round` (rounded edge), and `square` (slightly elongated square edge). Default is `butt`.
`name="stroke-dasharray"`	No	Encodes a dash pattern as a series of numbers separated by spaces. Odd-indexed numbers (first, third, etc) determine the length in pxiels to draw the line, and even-indexed numbers (second, fourth, etc) determine the length in pixels to blank out the line. Default is an unbroken line. *Starting from version 2.1* dash arrays can be combined with graphic strokes to generate complex line styles with alternating symbols or a mix of lines and symbols.
`name="stroke-dashoffset"`	No	Specifies the distance in pixels into the `dasharray` pattern at which to start drawing. Default is `0`.

Example

The following symbolizer is taken from the *Lines* section in the *SLD Cookbook*.

```
<LineSymbolizer>
  <Stroke>
    <CssParameter name="stroke">#0000FF</CssParameter>
    <CssParameter name="stroke-width">3</CssParameter>
    <CssParameter name="stroke-dasharray">5 2</CssParameter>
  </Stroke>
</LineSymbolizer>
```

The symbolizer styles a feature as a dashed blue line of width 3 pixels.

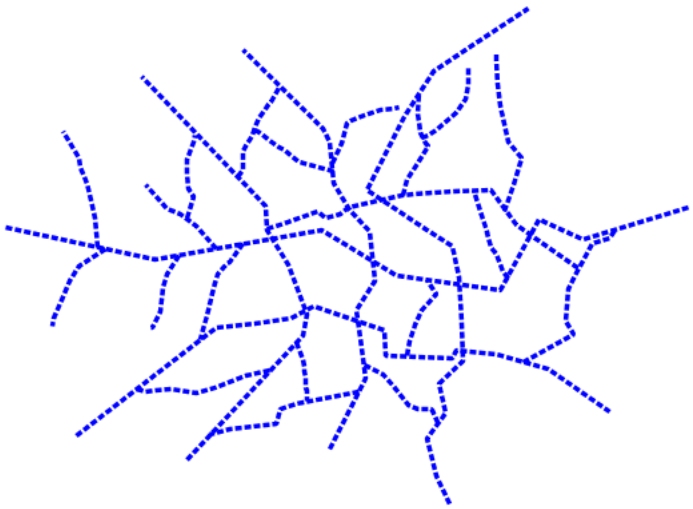


Figure 12.49: *Dashed blue line*

12.4.8 PolygonSymbolizer

A **PolygonSymbolizer** styles features as **polygons**. Polygons are two-dimensional geometries. They can be depicted with styling for their interior (fill) and their border (stroke). Polygons may contain one or more holes, which are stroked but not filled. When rendering a polygon, the fill is rendered before the border is stroked.

Syntax

A `<PolygonSymbolizer>` contains an optional `<Geometry>` element, and two elements `<Fill>` and `<Stroke>` for specifying styling:

Tag	Required?	Description
`<Geometry>`	No	Specifies the geometry to be rendered.
`<Fill>`	No	Specifies the styling for the polygon interior.
`<Stroke>`	No	Specifies the styling for the polygon border.

Geometry

The `<Geometry>` element is optional. If present, it specifies the featuretype property from which to obtain the geometry to style using the `PropertyName` element. See also *Geometry transformations in SLD* for GeoServer extensions for specifying geometry.

Any kind of geometry may be styled with a `<PolygonSymbolizer>`. Point geometries are treated as small orthonormal square polygons. Linear geometries are closed by joining their ends.

Stroke

The `<Stroke>` element specifies the styling for the **border** of a polygon. The syntax is described in the `<LineSymbolizer>` *Stroke* section.

Fill

The `<Fill>` element specifies the styling for the **interior** of a polygon. It can contain the sub-elements:

Tag	Required?	Description
`<GraphicFill>`	No	Renders the fill of the polygon with a repeated pattern.
`<CssParameter>`	0..N	Specifies parameters for filling with a solid color.

GraphicFill

The `<GraphicFill>` element contains a `<Graphic>` element, which specifies a graphic image or symbol to use for a repeated fill pattern. The syntax is described in the `PointSymbolizer` *Graphic* section.

CssParameter

The `<CssParameter>` elements describe the styling of a solid polygon fill. Any number of `<CssParameter>` elements can be specified.

The `name` **attribute** indicates what aspect of styling an element specifies, using the standard CSS/SVG styling model. The **content** of the element supplies the value of the styling parameter. The value may contain *expressions*.

The following parameters are supported:

Parameter	Required?	Description
`name="fill"`	No	Specifies the fill color, in the form `#RRGGBB`. Default is grey (`#808080`).
`name="fill-opacity"`	No	Specifies the opacity (transparency) of the fill. The value is a decimal number between `0` (completely transparent) and `1` (completely opaque). Default is `1`.

Example

The following symbolizer is taken from the *Polygons* section in the *SLD Cookbook*.

```
1         <PolygonSymbolizer>
2           <Fill>
3             <CssParameter name="fill">#000080</CssParameter>
```

```
4           </Fill>
5         </PolygonSymbolizer>
```

This symbolizer contains only a `<Fill>` element. Inside this element is a `<CssParameter>` that specifies the fill color for the polygon to be `#000080` (a muted blue).

Further examples can be found in the *Polygons* section of the *SLD Cookbook*.

12.4.9 TextSymbolizer

A **TextSymbolizer** styles features as **text labels**. Text labels are positioned eoither at points or along linear paths derived from the geometry being labelled.

Labelling is a complex operation, and effective labelling is crucial to obtaining legible and visually pleasing cartographic output. For this reason SLD provides many options to control label placement. To improve quality even more GeoServer provides additional options and parameters. The usage of the standard and extended options are described in greater detail in the following section on *Labeling*.

Syntax

A `<TextSymbolizer>` contains the following elements:

Tag	Required?	Description
`<Geometry>`	No	The geometry to be labelled.
`<Label>`	No	The text content for the label.
`<Font>`	No	The font information for the label.
`<LabelPlacement>`	No	Sets the position of the label relative to its associated geometry.
`<Halo>`	No	Creates a colored background around the label text, for improved legibility.
`<Fill>`	No	The fill style of the label text.
`<Graphic>`	No	A graphic to be displayed behind the label text. See *Graphic* for content syntax.
`<Priority>`	No	The priority of the label during conflict resolution. Content may contains *expressions*. See also *Priority Labeling*.
`<VendorOption>`	0..N	A GeoServer-specific option. See *Labeling* for descriptions of the available options. Any number of options may be specified.

Geometry

The `<Geometry>` element is optional. If present, it specifies the featuretype property from which to obtain the geometry to label, using a `<PropertyName>` element. See also *Geometry transformations in SLD* for GeoServer extensions for specifying geometry.

Any kind of geometry may be labelled with a `<TextSymbolizer>`. For non-point geometries, a representative point is used (such as the centroid of a line or polygon).

Label

The `<Label>` element specifies the text that will be rendered as the label. It allows content of mixed type, which means that the content can be a mixture of string data and *Filter Expressions*. These are concatenated to form the final label text. If a label is provided directly by a feature property, the content is a single

<PropertyName>. Multiple properties can be included in the label, and property values can be manipulated by filter expressions and functions. Additional "boilerplate" text can be provided as well. Whitespace can be preserved by surrounding it with XML <![CDATA[ ]]> delimiters.

If this element is omitted, no label is rendered.

Font

The <Font> element specifes the font to be used for the label. A set of <CssParameter> elements specify the details of the font.

The name **attribute** indicates what aspect of the font is described, using the standard CSS/SVG font model. The **content** of the element supplies the value of the font parameter. The value may contain *expressions*.

Parameter	Required?	Description
name="font-family"	No	The family name of the font to use for the label. Default is Times.
name="font-style"	No	The style of the font. Options are normal, italic, and oblique. Default is normal.
name="font-weight"	No	The weight of the font. Options are normal and bold. Default is normal.
name="font-size"	No	The size of the font in pixels. Default is 10.

LabelPlacement

The <LabelPlacement> element specifies the placement of the label relative to the geometry being labelled. There are two possible sub-elements: <PointPlacement> or <LinePlacement>. Exactly one of these must be specified.

Tag	Required?	Description
<PointPlacement>	No	Labels a geometry at a single point
<LinePlacement>	No	Labels a geometry along a linear path

PointPlacement

The <PointPlacement> element indicates the label is placed at a labelling point derived from the geometry being labelled. The position of the label relative to the labelling point may be controlled by the following sub-elements:

Tag	Required?	Description
<AnchorPoint>	No	The location within the label bounding box that is aligned with the label point. The location is specified by <AnchorPointX> and <AnchorPointY> sub-elements, with values in the range [0..1]. Values may contain *expressions*.
<Displacement>	No	Specifies that the label point should be offset from the original point. The offset is specified by <DisplacementX> and <DisplacementY> sub-elements, with values in pixels. Values may contain *expressions*. Default is (0, 0).
<Rotation>	No	The rotation of the label in clockwise degrees (negative values are counterclockwise). Value may contain *expressions*. Default is 0.

The anchor point justification, displacement offsetting, and rotation are applied in that order.

LinePlacement

The `<LinePlacement>` element indicates the label is placed along a linear path derived from the geometry being labelled. The position of the label relative to the linear path may be controlled by the following sub-element:

Tag	Required?	Description
`<PerpendicularOffset>`	No	The offset from the linear path, in pixels. Positive values offset to the left of the line, negative to the right. Value may contain *expressions*. Default is `0`.

The appearance of text along linear paths can be further controlled by the vendor options `followLine`, `maxDisplacement`, `repeat`, `labelAllGroup`, and `maxAngleDelta`. These are described in *Labeling*.

Halo

A halo creates a colored background around the label text, which improves readability in low contrast situations. Within the `<Halo>` element there are two sub-elements which control the appearance of the halo:

Tag	Required?	Description
`<Radius>`	No	The halo radius, in pixels. Value may contain *expressions*. Default is `1`.
`<Fill>`	No	The color and opacity of the halo via `CssParameter` elements for `fill` and `fill-opacity`. See *Fill* for full syntax. The parameter values may contain *expressions*. Default is a **white** fill (`#FFFFFF`) at **100%** opacity.

Fill

The `<Fill>` element specifies the fill style for the label text. The syntax is the same as that of the `PolygonSymbolizer` *Fill* element. The default fill color is **black** (`#FFFFFF`) at **100%** opacity..

Graphic

The `<Graphic>` element specifies a graphic symbol to be displayed behind the label text (if any). A classic use for this is to display "highway shields" behind road numbers provided by feature attributes. The element content has the same syntax as the `<PointSymbolizer>` *Graphic* element. Graphics can be provided by internal *mark symbols*, or by external images or SVG files. Their size and aspect ratio can be changed to match the text displayed with them by using the vendor options *graphic-resize* and *graphic-margin*.

Example

The following symbolizer is taken from the *Points* section in the *SLD Cookbook*.

```
1       <TextSymbolizer>
2         <Label>
3           <ogc:PropertyName>name</ogc:PropertyName>
4         </Label>
5         <Font>
6           <CssParameter name="font-family">Arial</CssParameter>
7           <CssParameter name="font-size">12</CssParameter>
8           <CssParameter name="font-style">normal</CssParameter>
9           <CssParameter name="font-weight">bold</CssParameter>
```

```
      </Font>
      <LabelPlacement>
        <PointPlacement>
          <AnchorPoint>
            <AnchorPointX>0.5</AnchorPointX>
            <AnchorPointY>0.0</AnchorPointY>
          </AnchorPoint>
          <Displacement>
            <DisplacementX>0</DisplacementX>
            <DisplacementY>25</DisplacementY>
          </Displacement>
          <Rotation>-45</Rotation>
        </PointPlacement>
      </LabelPlacement>
      <Fill>
        <CssParameter name="fill">#990099</CssParameter>
      </Fill>
    </TextSymbolizer>
```

The symbolizer labels features with the text from the `name` property. The font is Arial in bold at 12 pt size, filled in purple. The labels are centered on the point along their lower edge, then displaced 25 pixels upwards, and finally rotated 45 degrees counterclockwise.

The displacement takes effect before the rotation during rendering, so the 25 pixel vertical displacement is itself rotated 45 degrees.

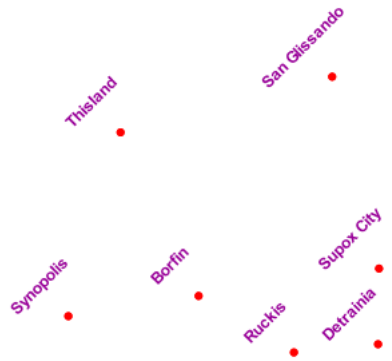


Figure 12.50: *Point with rotated label*

12.4.10 Labeling

This section discusses the details of controlling label placement via the standard SLD options. It also describes a number of GeoServer enhanced options for label placement that provide better cartographic output.

LabelPlacement

The SLD specification defines two alternative label placement strategies which can be used in the `<LabelPlacement>` element:

- `<PointPlacement>` places labels at a single point

- `<LinePlacement>` places labels along a line

PointPlacement

When `<PointPlacement>` is used the geometry is labelled at a single **label point**. For lines, this point lies at the middle of the visible portion of the line. For polygons, the point is the centroid of the visible portion of the polygon. The position of the label relative to the label point can be controlled by the following sub-elements:

Element	Description
`<AnchorPoint>`	This is relative to the LABEL. Using this you can do things such as center the label on top of the point, have the label to the left of the point, or have the label centered under the point.
`<Displacement>`	Offsets the label from the anchor point by a given number pixels in X and Y.
`<Rotation>`	Rotates the label clockwise by a given number of degrees.

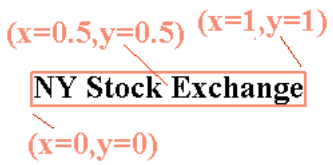
The best way to explain these options is with examples.

AnchorPoint

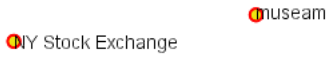
The anchor point determines where the label is placed relative to the label point.

```
<AnchorPoint>
   <AnchorPointX>
   0.5
   </AnchorPointX>
   <AnchorPointY>
   0.5
   </AnchorPointY>
</AnchorPoint>
```

The anchor point values are specified relative to the bounding box of the label. The bottom left of the box is (0, 0), the top left is (1, 1), and the middle is (0.5, 0.5). The (X,Y) location of the anchor point inside the label's bounding box is placed at the label point.



The following examples show how changing the anchor point affects the position of labels:



X=0, Y=0.5 - (default) places the label to the right of the label point

X=0.5, Y=0.5 - places the centre of the label at the label point

X=1, Y=0.5 - places the label to the left of the label point

X=0.5, Y=0 - places the label horizontally centred above the label point

Displacement

Displacement allows fine control of the placement of the label. The displacement values offset the location of the label from the anchor point by a specified number of pixels. The element syntax is:

```
<Displacement>
  <DisplacementX>
     10
  </DisplacementX>
  <DisplacementY>
      0
  </DisplacementY>
</Displacement>
```

Examples:

Displacement of X=10 pixels (compare with default anchor point of (X=0, Y=0.5) shown above)

Displacement of Y=-10 pixels (compare with anchor point (X= 0.5, Y=1.0) - not shown)

Rotation

The optional `<Rotation>` element specifies that labels should be rotated clockwise by a given number of degrees

```
<Rotation>
  45
</Rotation>
```

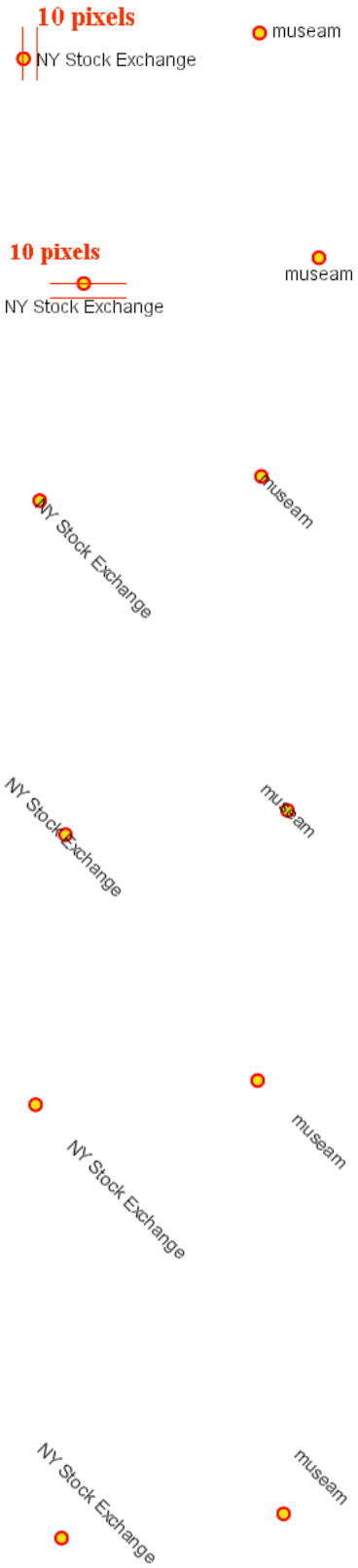
The examples below show how the rotation interacts with anchor points and displacements.

45 degree rotation

45 degree rotation with anchor point (X=0.5, Y=0.5)

45 degree rotation with 40-pixel X displacement

45 degree rotation with 40-pixel Y displacement with anchor point (X=0.5, Y=0.5)


10 pixels
NY Stock Exchange
museam
10 pixels
NY Stock Exchange
museam
NY Stock Exchange
museam
NY Stock Exchange
museam
NY Stock Exchange
museam
NY Stock Exchange
museam

LinePlacement

To label linear features (such as a road or river), the `<LinePlacement>` element can be specified. This indicates that the styler should determine the best placement and rotation for the labels along the lines.

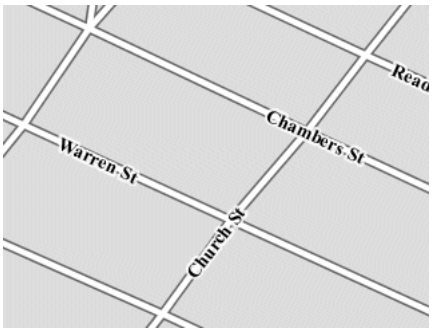
The standard SLD LinePlacement element provides one optional sub-element, `<PerpendicularOffset>`. GeoServer provides much more control over line label placement via vendor-specific options; see below for details.

PerpendicularOffset

The optional `<PerpendicularOffset>` element allows you to position a label above or below a line. (This is similiar to the `<DisplacementY>` for label points described above.) The displacement value is specified in pixels. A positive value displaces upwards, a negative value downwards.

```
<LabelPlacement>
  <LinePlacement>
    <PerpendicularOffset>
       10
    </PerpendicularOffset>
  </LinePlacement>
</LabelPlacement>
```

Examples:



PerpendicularOffset = 0 (default)

PerpendicularOffset = 10

Composing labels from multiple attributes

The `<Label>` element in *<TextSymbolizer>* allows mixed content. This means its content can be a mixture of plain text and *Filter Expressions*. The mix gets interepreted as a concatenation. You can leverage this to create complex labels out of multiple attributes.

For example, if you want both a state name and its abbreviation to appear in a label, you can do the following:

```
<Label>
  <ogc:PropertyName>STATE_NAME</ogc:PropertyName> (<ogc:PropertyName>STATE_ABBR</ogc:PropertyName>)
</Label>
```

and you'll get a label looking like `Texas (TX)`.

If you need to add extra white space or newline, you'll stumble into an XML oddity. The whitespace handling in the Label element is following a XML rule called "collapse", in which all leading and trailing whitespaces have to be removed, whilst all whitespaces (and newlines) in the middle of the xml element are collapsed into a single whitespace.

So, what if you need to insert a newline or a sequence of two or more spaces between your property names? Enter CDATA. CDATA is a special XML section that has to be returned to the interpreter as-is, without following any whitespace handling rule. So, for example, if you wanted to have the state abbreviation sitting on the next line you'd use the following:

```
<Label>
  <ogc:PropertyName>STATE_NAME</ogc:PropertyName><![CDATA[
]]>(<ogc:PropertyName>STATE_ABBR</ogc:PropertyName>)
</Label>
```

Geoserver Enhanced Options

GeoServer provides a number of label styling options as extensions to the SLD specification. Using these options gives more control over how the map looks, since the SLD standard isn't expressive enough to provide all the options one might want.

These options are specified as subelements of `<TextSymbolizer>`.

Priority Labeling

The optional `<Priority>` element allows specifying label priority. This controls how conflicts (overlaps) between labels are resolved during rendering. The element content may be an *expression* to retrieve or calculate a relative priority value for each feature in a layer. Alternatively, the content may be a constant value, to set the priority of a layer's labels relative to other layers on a rendered map.

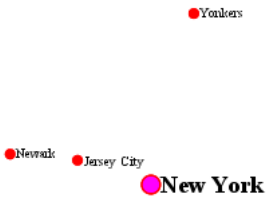
The default priority for labels is 1000.

Note: Standard SLD Conflict Resolution

If the `<Priority>` element is not present, or if a group of labels all have the same priority, then standard SLD label conflict resolution is used. Under this strategy, the label to display out of a group of conflicting labels is chosen essentially at random.

For example, take the following dataset of cities:

```
City Name    | population
-------------+------------
Yonkers      |     197,818
Jersey City  |     237,681
Newark       |     280,123
New York     |   8,107,916
```



City locations (large scale map)

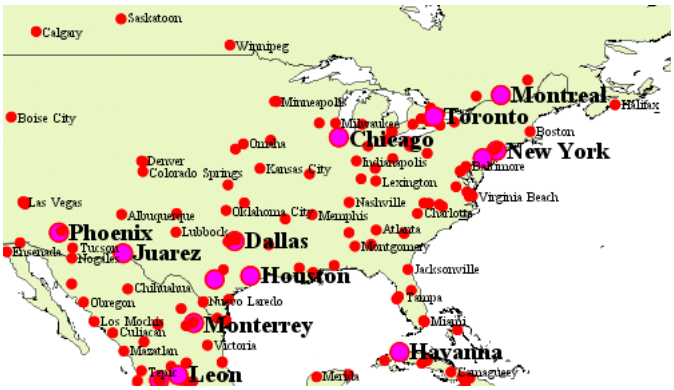
More people know where New York City is than where Jersey City is. Thus we want to give the label "New York" priority so it will be visible when in conflict with (overlapping) "Jersey City". To do this we include the following code in the `<TextSymbolizer>`:

```
<Priority>
    <PropertyName>population</PropertyName>
</Priority>
```

This ensures that at small scales New York is labeled in preference to the less populous cities nearby:

City locations (small scale map)

Without priority labeling, Jersey City could be labeled in preference to New York, making it difficult to interpret the map. At scales showing many features, priority labeling is essential to ensure that larger cities are more visible than smaller cities.



Grouping Features (group)

The `group` option allows displaying a single label for multiple features in a logical group.

```
<VendorOption name="group">yes</VendorOption>
```

Grouping works by collecting all features with the same label text, then choosing a representative geometry for the group, according to the following rules:

Geometry	Label Point
Point Set	The first point inside the view rectangle is used.
Line Set	Lines are joined together, clipped to the view rectangle, and the longest path is used.
Polygon Set	Polygons are clipped to the view rectangle, and the largest polygon is used.

If desired the labeller can be forced to label every element in a group by specifying the *labelAllGroup* option.

Warning: Be careful that the labels truly indicate features that should be grouped together. For example, grouping on city name alone might end up creating a group containing both *Paris* (France) and *Paris* (Texas).

Road data is a classic example to show why grouping is useful. It is usually desirable to display only a single label for all of "Main Street", not a label for every block of "Main Street."

When the `group` option is off (the default), grouping is not performed and every block feature is labeled (subject to label deconfliction):

When the `group` option is used, geometries with the same label are grouped together and the label position is determined from the entire group. This produces a much less cluttered map:

labelAllGroup

The `labelAllGroup` option can be used in conjunction with the `group` option (see *Grouping Features (group)*). It causes *all* of the disjoint paths in a line group to be labeled, not just the longest one.

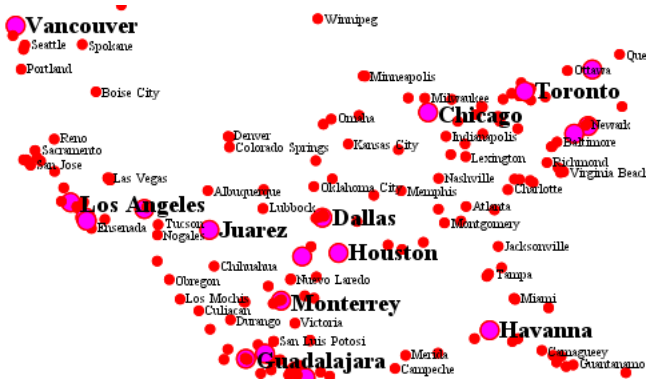
```
<VendorOption name="labelAllGroup">true</VendorOption>
```

Overlapping and Separating Labels (spaceAround)

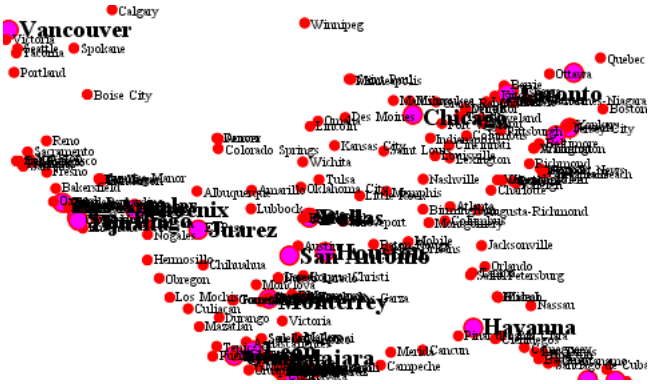
By default GeoServer will not render labels "on top of each other". By using the `spaceAround` option you can either allow labels to overlap, or add extra space around labels. The value supplied for the option is a positive or negative size, in pixels.

```
<VendorOption name="spaceAround">10</VendorOption>
```

Using the default value of 0, the bounding box of a label cannot overlap the bounding box of another label:



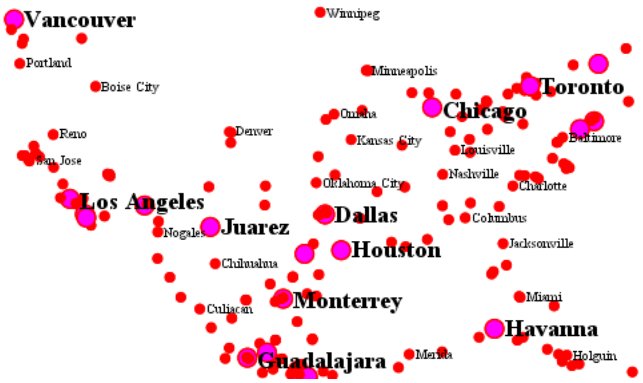
With a negative `spaceAround` value, overlapping is allowed:



With a positive `spaceAround` value of 10, each label is at least 20 pixels apart from others:

Positive `spaceAround` values actually provide twice the space that you might expect. This is because you can specify a spaceAround for one label as 5, and for another label (in another TextSymbolizer) as 3. The total distance between them is 8. Two labels in the first symbolizer ("5") will each be 5 pixels apart from each other, for a total of 10 pixels.

Note: Interaction between values in different TextSymbolizers



You can have multiple TextSymbolizers in your SLD file, each with a different `spaceAround` option. If all the `spaceAround` options are >=0, this will do what you would normally expect. If you have negative values ('allow overlap') then these labels can overlap labels that you've said should not be overlapping. If you don't like this behavior, it's not difficult to change - feel free to submit a patch!

---

followLine

The `followLine` option forces a label to follow the curve of the line. To use this option add the following to the `<TextSymbolizer>`.

Note: Straight Lines

You don't need to use followLine for straight lines. GeoServer will automatically follow the orientation of the line. However in this case `followLine` can be used to ensure the text isn't rendered if longer than the line.

```
<VendorOption name="followLine">true</VendorOption>
```

It is required to use `<LinePlacement>` along with this option to ensure that labels are placed along lines:

```
<LabelPlacement>
  <LinePlacement/>
</LabelPlacement>
```

maxDisplacement

The `maxDisplacement` option controls the displacement of the label along a line, around a point and inside a polygon.

For lines, normally GeoServer labels a line at its center point only. If this label conflicts with another one it may not be displayed at all. When this option is enabled the labeller will attempt to avoid conflic by using an alternate location within **maxDisplacement** pixels along the line from the pre-computed label point.

If used in conjunction with *repeat*, the value for `maxDisplacement` should always be **lower** than the value for `repeat`.

For points this causes the renderer to start circling around the point in search of a empty stop to place the label, step by step increasing the size of the circle until the max displacement is reached. The same happens for polygons, around the polygon labelling point (normally the centroid).

```
<VendorOption name="maxDisplacement">10</VendorOption>
```

repeat

The `repeat` option determines how often GeoServer displays labels along a line. Normally GeoServer labels each line only once, regardless of length. Specifying a positive value for this option makes the labeller attempt to draw the label every **repeat** pixels. For long or complex lines (such as contour lines) this makes labeling more informative.

```
<VendorOption name="repeat">100</VendorOption>
```

maxAngleDelta

When used in conjunction with *followLine*, the `maxAngleDelta` option sets the maximum angle, in degrees, between two subsequent characters in a curved label. Large angles create either visually disconnected words or overlapping characters. It is advised not to use angles larger than 30.

```
<VendorOption name="maxAngleDelta">15</VendorOption>
```

autoWrap

The `autoWrap` option wraps labels when they exceed the given width (in pixels). The size should be wide enough to accommodate the longest word, otherwise single words will be split over multiple lines.

```
<VendorOption name="autoWrap">50</VendorOption>
```

Labeling with autoWrap enabled

forceLeftToRight

The renderer tries to draw labels along lines so that the text is upright, for maximum legibility. This means a label may not follow the line orientation, but instead may be rotated 180° to display the text the right way up. In some cases altering the orientation of the label is not desired; for example, if the label is a directional arrow showing the orientation of the line.

The `forceLeftToRight` option can be set to `false` to disable label flipping, making the label always follow the inherent orientation of the line being labelled:

```
<VendorOption name="forceLeftToRight">false</VendorOption>
```

conflictResolution

By default labels are subject to **conflict resolution**, meaning the renderer will not allow any label to overlap with a label that has been already drawn. Setting the `conflictResolution` option to `false` causes this label to bypass conflict resolution. This means the label will be drawn even if it overlaps with other labels, and other labels drawn after it may overlap it.

```
<VendorOption name="conflictResolution">false</VendorOption>
```

goodnessOfFit

Geoserver will remove labels if they are a particularly bad fit for the geometry they are labeling.

Geometry	Goodness of Fit Algorithm
Point	Always returns 1.0 since the label is at the point
Line	Always returns 1.0 since the label is always placed on the line.
Polygon	The label is sampled approximately at every letter. The distance from these points to the polygon is determined and each sample votes based on how close it is to the polygon. (see LabelCacheDefault#goodnessOfFit())

The default value is 0.5, but it can be modified using:

```
<VendorOption name="goodnessOfFit">0.3</VendorOption>
```

polygonAlign

GeoServer normally tries to place labels horizontally within a polygon, and gives up if the label position is busy or if the label does not fit enough in the polygon. This option allows GeoServer to try alternate rotations for the labels.

```
<VendorOption name="polygonAlign">mbr</VendorOption>
```

Option	Description
`manual`	The default value. Only a rotation manually specified in the `<Rotation>` tag will be used
`ortho`	If the label does not fit horizontally and the polygon is taller than wider then vertical alignment will also be tried
`mbr`	If the label does not fit horizontally the minimum bounding rectangle will be computed and a label aligned to it will be tried out as well

graphic-resize

When a `<Graphic>` is specified for a label by default it is displayed at its native size and aspect ratio. The `graphic-resize` option instructs the renderer to magnify or stretch the graphic to fully contain the text of the label. If this option is used the `graphic-margin` option may also be specified.

```
<VendorOption name="graphic-resize">stretch</VendorOption>
```

Option	Description
`none`	Graphic is displayed at its native size (default)
`proportional`	Graphic size is increased uniformly to contain the label text
`stretch`	Graphic size is increased anisotropically to contain the label text

Labeling with a Graphic Mark "square" - L) at native size; R) with "graphic-resize"=stretch and "graphic-margin"=3

graphic-margin

The `graphic-margin` options specifies a margin (in pixels) to use around the label text when the `graphic-resize` option is specified.

```
<VendorOption name="graphic-margin">margin</VendorOption>
```

12.4.11 RasterSymbolizer

GeoServer supports the ability to display raster data in addition to vector data.

Raster data is not merely a picture, rather it can be thought of as a grid of georeferenced information, much like a graphic is a grid of visual information (with combination of reds, greens, and blues). Unlike graphics, which only contain visual data, each point/pixel in a raster grid can have many different attributes (bands), with possibly none of them having an inherently visual component.

With the above in mind, one needs to choose how to visualize the data, and this, like in all other cases, is done by using an SLD. The analogy to vector data is evident in the naming of the tags used. Vectors, consisting of points, line, and polygons, are styled by using the `<PointSymbolizer>`, `<LineSymbolizer>`, and `<PolygonSymbolizer>` tags. It is therefore not very surprising that raster data is styled with the tag <RasterSymbolizer>.

Syntax

The following elements can be used inside the `<RasterSymbolizer>` element.

- <Opacity>
- <ColorMap>
- <ChannelSelection>
- <ContrastEnhancement>
- <ShadedRelief> *
- <OverlapBehavior> *
- <ImageOutline> *

Warning: The starred (*) elements are not yet implemented in GeoServer.

Opacity

The `<Opacity>` element sets the transparency level for the entire rendered image. As is standard, the values range from zero (0) to one (1), with zero being transparent, and one being opaque. The syntax is:

```
<Opacity>0.5</Opacity>
```

where, in this case, the raster is rendered at 50% opacity.

ColorMap

The `<ColorMap>` element defines the color values for the pixels of a raster image, as either color gradients, or a mapping of specific values to fixed colors.

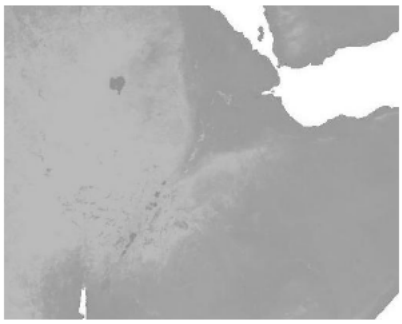
A color map is defined by a sequence of `<ColorMapEntry>` elements. Each `<ColorMapEntry>` element specifies a `color` and a `quantity` attribute. The quantity refers to the value of a raster pixel. The `color` value is denoted in standard hexadecimal RGB format (#RRGGBB). `<ColorMapEntry>` elements can also have `opacity` and `label` attributes. The `opacity` attribute overrides the global `<Opacity>` value. The `label` attribute is used to provide text for legends. A color map can contain up to 255 `<ColorMapEntry>` elements.

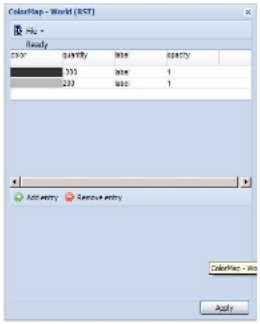
The simplest `<ColorMap>` has two color map entries. One specifyies a color for the "bottom" of the dataset, and the other specifyies a color for the "top" of the dataset. Pixels with values equal to or less than the minimum value are rendered with the bottom color (and opacity). Pixels with values equal to or great than the maximum value are rendered with the top color and opacity. The colors for values in between are automatically interpolated, making creating color gradients easy.

A color map can be refined by adding additional intermediate entries. This is useful if the dataset has discrete values rather than a gradient, or if a multi-colored gradient is desired. One entry is added for each different color to be used, along with the corresponding quantity value.

For example, a simple ColorMap can define a color gradient from color #323232 to color #BBBBBB over quantity values from -300 to 200:

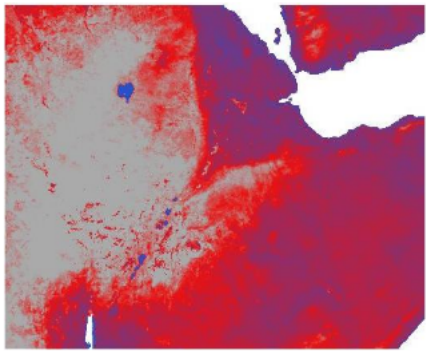
```
<ColorMap>
        <ColorMapEntry color="#323232" quantity="-300" label="label1" opacity="1"/>
        <ColorMapEntry color="#BBBBBB" quantity="200" label="label2" opacity="1"/>
</ColorMap>
```





A more refined example defines a color gradient from color #FFCC32 through color #BBBBBB, running through color #3645CC and color #CC3636. The bottom color #FFCC32 is defined to be transparent This simulates an alpha channel, since pixels with values of -300 and below will not be rendered. Notice that the default opacity is 1 (opaque) when not specified.

```
<ColorMap>
        <ColorMapEntry color="#FFCC32" quantity="-300" label="label1" opacity="0"/>
        <ColorMapEntry color="#3645CC" quantity="0" label="label2" opacity="1"/>
        <ColorMapEntry color="#CC3636" quantity="100" label="label3" opacity="1"/>
        <ColorMapEntry color="#BBBBBB" quantity="200" label="label4" opacity="1"/>
</ColorMap>
```

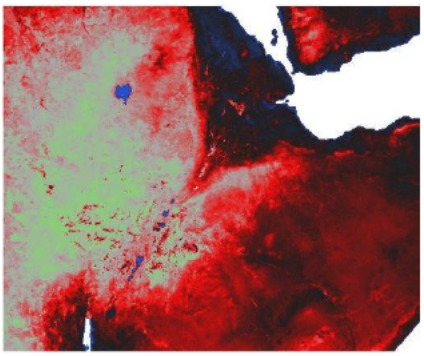


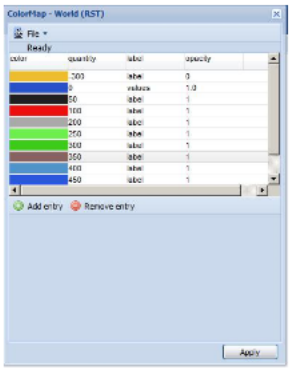
GeoServer extends the `<ColorMap>` element to allow two attributes: `type` and `extended`.

type The `<ColorMap>` `type` attribute specifies the kind of ColorMap to use. There are three different types of ColorMaps that can be specified: `ramp`, `intervals` and `values`.

`type="ramp"` is the default ColorMap type. It specifies that colors should be interpolated for values between the color map entries. The result is shown in the following example.

```
<ColorMap type="ramp">
        <ColorMapEntry color="#EEBE2F" quantity="-300" label="label" opacity="0"/>
        <ColorMapEntry color="#2851CC" quantity="0" label="values" opacity="1"/>
        <ColorMapEntry color="#211F1F" quantity="50" label="label" opacity="1"/>
        <ColorMapEntry color="#EE0F0F" quantity="100" label="label" opacity="1"/>
        <ColorMapEntry color="#AAAAAA" quantity="200" label="label" opacity="1"/>
        <ColorMapEntry color="#6FEE4F" quantity="250" label="label" opacity="1"/>
        <ColorMapEntry color="#3ECC1B" quantity="300" label="label" opacity="1"/>
        <ColorMapEntry color="#886363" quantity="350" label="label" opacity="1"/>
        <ColorMapEntry color="#5194CC" quantity="400" label="label" opacity="1"/>
        <ColorMapEntry color="#2C58DD" quantity="450" label="label" opacity="1"/>
        <ColorMapEntry color="#DDB02C" quantity="600" label="label" opacity="1"/>
</ColorMap>
```

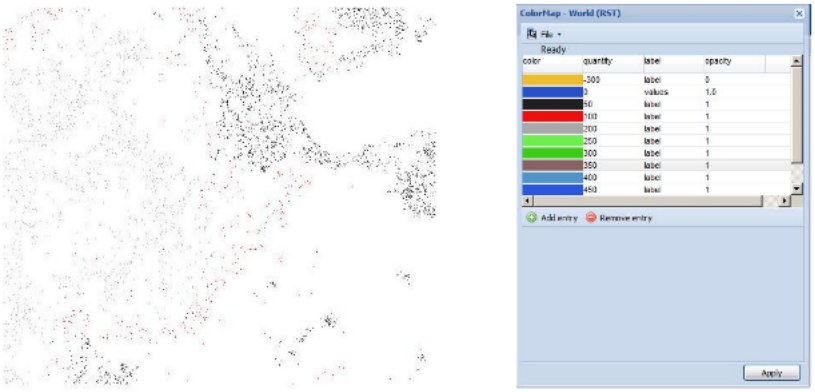




`type="values"` means that only pixels with the specified entry quantity values are rendered. Pixels with other values are not rendered. Using the example set of color map entries:

```
<ColorMap type="values">
        <ColorMapEntry color="#EEBE2F" quantity="-300" label="label" opacity="0"/>
        ...
        <ColorMapEntry color="#DDB02C" quantity="600" label="label" opacity="1"/>
</ColorMap>
```

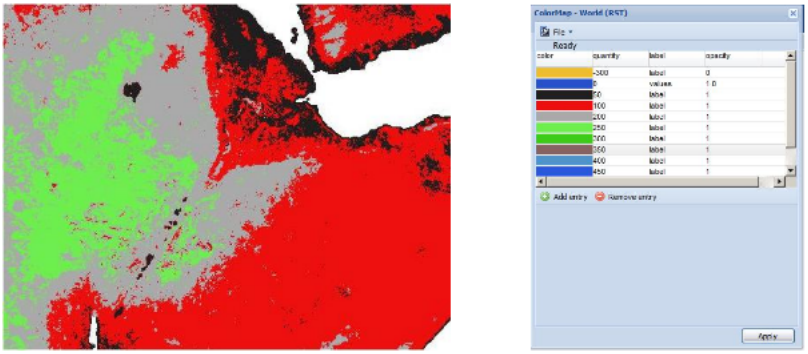
The result image is:



`type="intervals"` value means that each interval defined by two entries is rendered using the color of the first (lowest-value) entry. No color interpolation is applied across the intervals. Using the example set of color map entries:

```
<ColorMap type="intervals" extended="true">
        <ColorMapEntry color="#EEBE2F" quantity="-300" label="label" opacity="0"/>
        ...
        <ColorMapEntry color="#DDB02C" quantity="600" label="label" opacity="1"/>
</ColorMap>
```

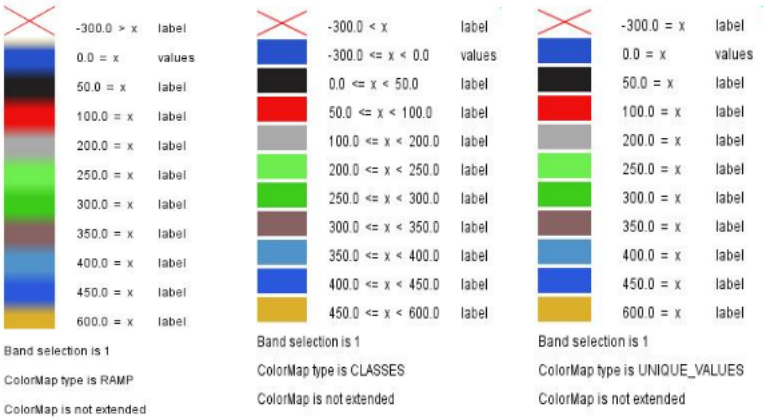
The result image is:



The color map type is also reflected in the legend graphic. A typical request for a raster legend is (using the `forceRule:true` option to force output of the color map):

```
http://localhost:8080/geoserver/wms?REQUEST=GetLegendGraphic&VERSION=1.0.0&&STYLE=raster100&FORMAT=ir
```

The legends returned for the different types are:



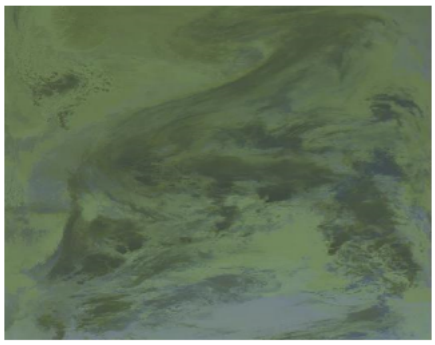
extended The `extended` attribute specifies whether the color map gradient uses 256 (8-bit) or 65536 (16-bit) colors. The value `false` (the default) specifies that the color scale is calculated using 8-bit color, and `true` specifies using 16-bit color.

ChannelSelection

The `<ChannelSelection>` element specifies how dataset bands are mapped to image color channels. Named dataset bands may be mapped to red, green and blue channels, or a single named band may be mapped to a grayscale channel.

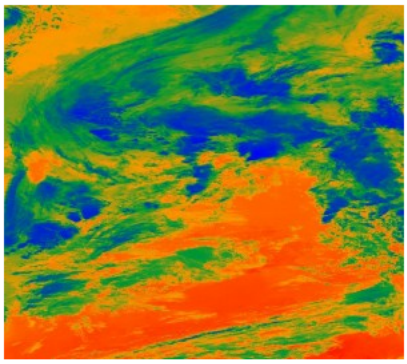
The following example maps source channels 1, 2 and 3 to the red, green, and blue color channels.

```
<ChannelSelection>
  <RedChannel>
    <SourceChannelName>1</SourceChannelName>
  </RedChannel>
  <GreenChannel>
    <SourceChannelName>2</SourceChannelName>
  </GreenChannel>
  <BlueChannel>
    <SourceChannelName>3</SourceChannelName>
  </BlueChannel>
</ChannelSelection>
```



The next example shows selecting a single band of an RGB image as a grayscale channel, and re-colorizing it via a ColorMap:

```
<RasterSymbolizer>
        <Opacity>1.0</Opacity>
        <ChannelSelection>
                <GrayChannel>
                        <SourceChannelName>1</SourceChannelName>
                </GrayChannel>
        </ChannelSelection>
        <ColorMap extended="true">
                <ColorMapEntry color="#0000ff" quantity="3189.0"/>
                <ColorMapEntry color="#009933" quantity="6000.0"/>
                <ColorMapEntry color="#ff9900" quantity="9000.0" />
                <ColorMapEntry color="#ff0000" quantity="14265.0"/>
        </ColorMap>
</RasterSymbolizer>
```



ContrastEnhancement

The `<ContrastEnhancement>` element is used to adjust the relative brightness of the image data. A `<ContrastEnhancement>` element can be specified for the entire image, or in individual `Channel` elements. In this way, different enhancements can be used on each channel.

There are three types of enhancements possible:

- Normalize
- Histogram
- GammaValue

`<Normalize>` means to expand the contrast so that the minimum quantity is mapped to minimum brightness, and the maximum quantity is mapped to maximum brightness.

`<Histogram>` is similar to Normalize, but the algorithm used attempts to produce an image with an equal number of pixels at all brightness levels.

`<GammaValue>` is a scaling factor that adjusts the brightness of the image. A value less than one (1) darkens the image, and a value greater than one (1) brightens it. The default is 1 (no change).

These examples turn on Normalize and Histogram, respectively:

```
<ContrastEnhancement>
        <Normalize/>
</ContrastEnhancement>
```

```
<ContrastEnhancement>
        <Histogram/>
</ContrastEnhancement>
```

This example increases the brightness of the image by a factor of two.

```
<ContrastEnhancement>
        <GammaValue>2</GammaValue>
</ContrastEnhancement>
```

ShadedRelief

Warning: Support for this element has not been implemented yet.

The `<ShadedRelief>` element can be used to create a 3-D effect, by selectively adjusting brightness. This is a nice effect to use on an elevation dataset. There are two types of shaded relief possible.

- BrightnessOnly
- ReliefFactor

BrightnessOnly, which takes no parameters, applies shading in WHAT WAY? ReliefFactor sets the amount of exaggeration of the shading (for example, to make hills appear higher). According to the OGC SLD specification, a value of around 55 gives "reasonable results" for Earth-based datasets:

```
<ShadedRelief>
        <BrightnessOnly />
        <ReliefFactor>55</ReliefFactor>
</ShadedRelief>
```

The above example turns on Relief shading in WHAT WAY?

OverlapBehavior

Warning: Support for this element has not been implemented yet.

Sometimes raster data is comprised of multiple image sets. Take, for example, a satellite view of the Earth at night . As all of the Earth can't be in nighttime at once, a composite of multiple images are taken. These images are georeferenced, and pieced together to make the finished product. That said, it is possible that two images from the same dataset could overlap slightly, and the OverlapBehavior element is designed to determine how this is handled. There are four types of OverlapBehavior:

- AVERAGE
- RANDOM
- LATEST_ON_TOP
- EARLIEST_ON_TOP

AVERAGE takes each overlapping point and displays their average value. **RANDOM** determines which image gets displayed according to chance (which can sometimes result in a crisper image). **LATEST_ON_TOP** and **EARLIEST_ON_TOP** sets the determining factor to be the internal timestamp on each image in the dataset. None of these elements have any parameters, and are all called in the same way:

```
<OverlapBehavior>
        <AVERAGE />
</OverlapBehavior>
```

The above sets the OverlapBehavior to AVERAGE.

ImageOutline

Warning: Support for this element has not been implemented yet.

Given the situation mentioned previously of the image composite, it is possible to style each image so as to have an outline. One can even set a fill color and opacity of each image; a reason to do this would be to "gray-out" an image. To use ImageOutline, you would define a <LineSymbolizer> or <PolygonSymbolizer> inside of the element:

```
<ImageOutline>
        <LineSymbolizer>
        <Stroke>
                        <CssParameter name="stroke">#0000ff</CssParameter>
        </Stroke>
        </LineSymbolizer>
</ImageOutline>
```

The above would create a border line (colored blue with a one pixel default thickness) around each image in the dataset.

12.5 SLD Extensions in GeoServer

GeoServer provides a number of vendor-specific extensions to SLD 1.0. Although not portable, the extensions make styling more powerful and concise and generate better-looking maps.

12.5.1 Geometry transformations in SLD

SLD symbolizers may contain an optional `<Geometry>` element, which allows specifying which geometry attribute is to be rendered. In the common case of a featuretype with a single geometry attribute this element is usually omitted, but it is useful when a featuretype has multiple geometry-valued attributes.

SLD 1.0 requires the `<Geometry>` content to be a `<ogc:PropertyName>`. GeoServer extends this to allow a general SLD expression to be used. The expression can contain filter functions that manipulate geometries by transforming them into something different. This facility is called SLD *geometry transformations*.

GeoServer provides a number of filter functions that can transform geometry. A full list is available in the *Filter Function Reference*. They can be used to do things such as extracting line vertices or endpoints, offsetting polygons, or buffering geometries.

Geometry transformations are computed in the geometry's original coordinate reference system, before any reprojection and rescaling to the output map is performed. For this reason, transformation parameters must be expressed in the units of the geometry CRS. This must be taken into account when using geometry transformations at different screen scales, since the parameters will not change with scale.

Examples

Let's look at some examples.

Extracting vertices

Here is an example that allows one to extract all the vertices of a geometry, and make them visible in a map, using the `vertices` function:

```
<PointSymbolizer>
  <Geometry>
    <ogc:Function name="vertices">
       <ogc:PropertyName>the_geom</ogc:PropertyName>
    </ogc:Function>
  </Geometry>
  <Graphic>
    <Mark>
      <WellKnownName>square</WellKnownName>
      <Fill>
        <CssParameter name="fill">#FF0000</CssParameter>
      </Fill>
    </Mark>
    <Size>6</Size>
  </Graphic>
</PointSymbolizer>
```

`View the full "Vertices" SLD`

Applied to the sample *tasmania_roads* layer this will result in:

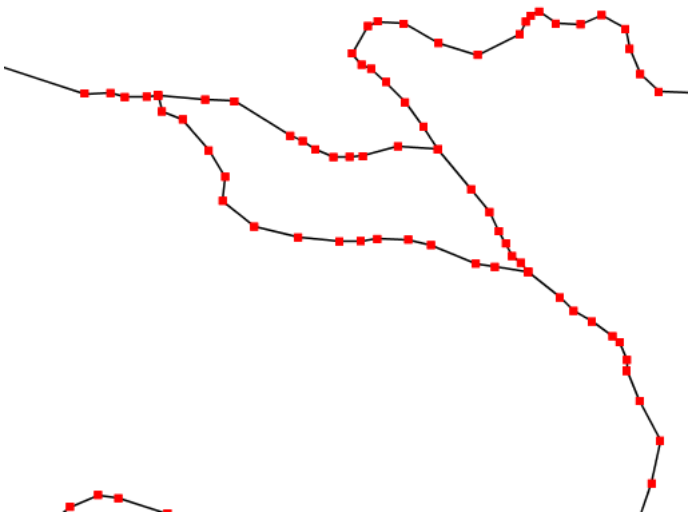


Figure 12.51: *Extracting and showing the vertices out of a geometry*

Start and end point

The *startPoint* and *endPoint* functions can be used to extract the start and end point of a line.

```
<PointSymbolizer>
  <Geometry>
    <ogc:Function name="startPoint">
      <ogc:PropertyName>the_geom</ogc:PropertyName>
    </ogc:Function>
  </Geometry>
  <Graphic>
    <Mark>
      <WellKnownName>square</WellKnownName>
      <Stroke>
        <CssParameter name="stroke">0x00FF00</CssParameter>
        <CssParameter name="stroke-width">1.5</CssParameter>
      </Stroke>
    </Mark>
    <Size>8</Size>
  </Graphic>
 </PointSymbolizer>
 <PointSymbolizer>
   <Geometry>
     <ogc:Function name="endPoint">
       <ogc:PropertyName>the_geom</ogc:PropertyName>
     </ogc:Function>
   </Geometry>
   <Graphic>
     <Mark>
       <WellKnownName>circle</WellKnownName>
       <Fill>
          <CssParameter name="fill">0xFF0000</CssParameter>
       </Fill>
     </Mark>
     <Size>4</Size>
   </Graphic>
 </PointSymbolizer>
```

View the full "StartEnd" SLD

Applied to the sample *tasmania_roads* layer this will result in:

Drop shadow

The *offset* function can be used to create drop shadow effects below polygons. Notice that the offset values reflect the fact that the data used in the example is in a geographic coordinate system.

```
  <PolygonSymbolizer>
    <Geometry>
       <ogc:Function name="offset">
          <ogc:PropertyName>the_geom</ogc:PropertyName>
          <ogc:Literal>0.00004</ogc:Literal>
          <ogc:Literal>-0.00004</ogc:Literal>
       </ogc:Function>
    </Geometry>
    <Fill>
      <CssParameter name="fill">#555555</CssParameter>
    </Fill>
  </PolygonSymbolizer>
```

View the full "Shadow" SLD

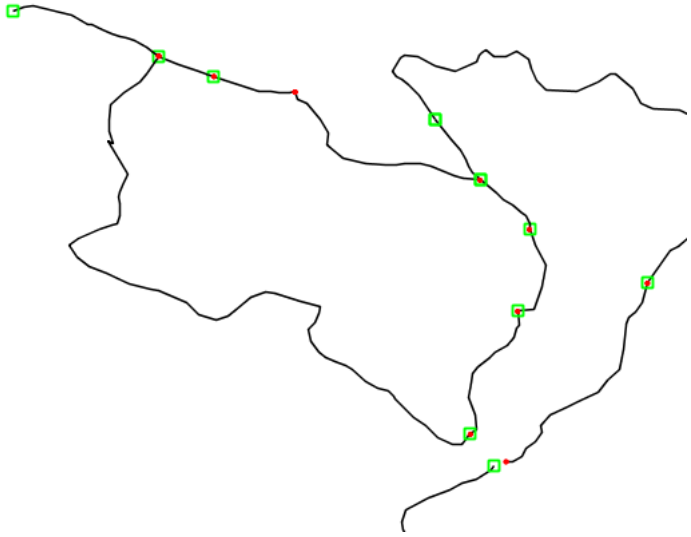


Figure 12.52: *Extracting start and end point of a line*

Applied to the sample *tasmania_roads* layer this will result in:

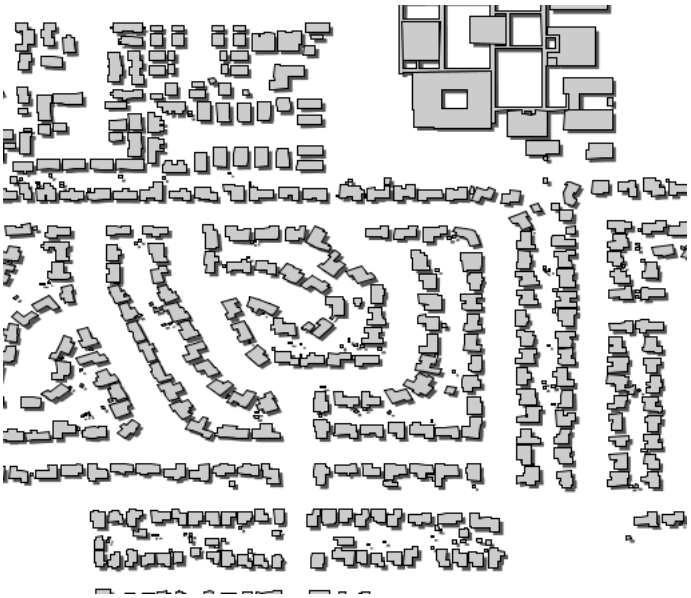


Figure 12.53: *Dropping building shadows*

Performance tips

GeoServer's filter functions contain a number of set-related or constructive geometric functions, such as `buffer`, `intersection`, `difference` and others. These can be used as geometry transformations, but they be can quite heavy in terms of CPU consumption so it is advisable to use them with care. One strategy is to activate them only at higher zoom levels, so that fewer features are processed.

Buffering can often be visually approximated by using very large strokes together with round line joins and line caps. This avoids incurring the performance cost of a true geometric buffer transformation.

Adding new transformations

Additional filter functions can be developed in Java and then deployed in a JAR file as a GeoServer plugin. A guide is not available at this time, but see the GeoTools `main` module for examples.

12.5.2 Rendering Transformations

Rendering Transformations allow processing to be carried out on datasets within the GeoServer rendering pipeline. A typical transformation computes a derived or aggregated result from the input data, allowing various useful visualization effects to be obtained. Transformations may transform data from one format into another (i.e vector to raster or vice-versa), to provide an appropriate format for display.

The following table lists examples of various kinds of rendering transformations available in GeoServer:

Type	Examples
Raster-to-Vector	**Contour** extracts contours from a DEM raster. **RasterAsPointCollections** extracts a vector field from a multi-band raster
Vector-to-Raster	**BarnesSurfaceInterpolation** computes a surface from scattered data points. **Heatmap** computes a heatmap surface from weighted data points.
Vector-to-Vector	**PointStacker** aggregates dense point data into clusters.

Rendering transformations are invoked within SLD styles. Parameters may be supplied to control the appearance of the output. The rendered output for the layer is produced by applying the styling rules and symbolizers in the SLD to the result of transformation.

Rendering transformations are implemented using the same mechanism as *WPS Processes*. They can thus also be executed via the WPS protocol, if required. Conversely, any WPS process can be executed as a transformation, as long as the input and output are appropriate for use within an SLD.

This section is a general guide to rendering transformation usage in GeoServer. For details of input, parameters, and output for any particular rendering transformation, refer to its own documentation.

Installation

Using Rendering Transformations requires the WPS extension to be installed. See *Installing the WPS extension*.

Note: The WPS service does not need to be **enabled** to use Rendering Transformations. To avoid unwanted consumption of server resources it may be desirable to disable the WPS service if it is not being used directly.

Usage

Rendering Transformations are invoked by adding the `<Transformation>` element to a `<FeatureTypeStyle>` element in an SLD document. This element specifies the name of the transformation process, and usually includes parameter values controlling the operation of the transformation.

The `<Transformation>` element syntax leverages the OGC Filter function syntax. The content of the element is a `<ogc:Function>` with the name of the rendering transformation process. Transformation processes may accept some number of parameters, which may be either required (in which case they must

be specified), or optional (in which case they may be omitted if the default value is acceptable). Parameters are supplied as name/value pairs. Each parameter's name and value are supplied via another function `<ogc:Function name="parameter">`. The first argument to this function is an `<ogc:Literal>` containing the name of the parameter. The optional following arguments provide the value for the parameter (if any). Some parameters accept only a single value, while others may accept a list of values. As with any filter function argument, values may be supplied in several ways:

- As a literal value
- As a computed expression
- As an SLD environment variable, whose actual value is supplied in the WMS request (see *Variable substitution in SLD*).
- As a predefined SLD environment variable (which allows obtaining values for the current request such as output image width and height).

The order of the supplied parameters is not significant.

Most rendering transformations take as input a dataset to be transformed. This is supplied via a special named parameter which does not have a value specified. The name of the parameter is determined by the particular transformation being used. When the transformation is executed, the input dataset is passed to it via this parameter.

The input dataset is determined by the same query mechanism as used for all WMS requests, and can thus be filtered in the request if required.

In rendering transformations which take as input a featuretype (vector dataset) and convert it to a raster dataset, in order to pass validation the SLD needs to mention the geometry attribute of the input dataset (even though it is not used). This is done by specifying the attribute name in the symbolizer `<Geometry>` element.

The output of the rendering transformation is styled using symbolizers appropriate to its format: *PointSymbolizer*, *LineSymbolizer*, *PolygonSymbolizer*, and *TextSymbolizer* for vector data, and *RasterSymbolizer* for raster coverage data.

If it is desired to display the input dataset in its orginal form, or transformed in another way, there are two options:

- Another `<FeatureTypeStyle>` can be used in the same SLD
- Another SLD can be created, and the layer displayed twice using the different SLDs

Notes

- Rendering transformations may not work correctly in tiled mode, unless they have been specifically written to accomodate it.

Examples

Contour extraction

`gs:Contour` is a **Raster-to-Vector** rendering transformation which extracts contour lines at specified levels from a raster DEM. The following SLD invokes the transformation and styles the contours as black lines.

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <StyledLayerDescriptor version="1.0.0"
3    xsi:schemaLocation="http://www.opengis.net/sld StyledLayerDescriptor.xsd"
```

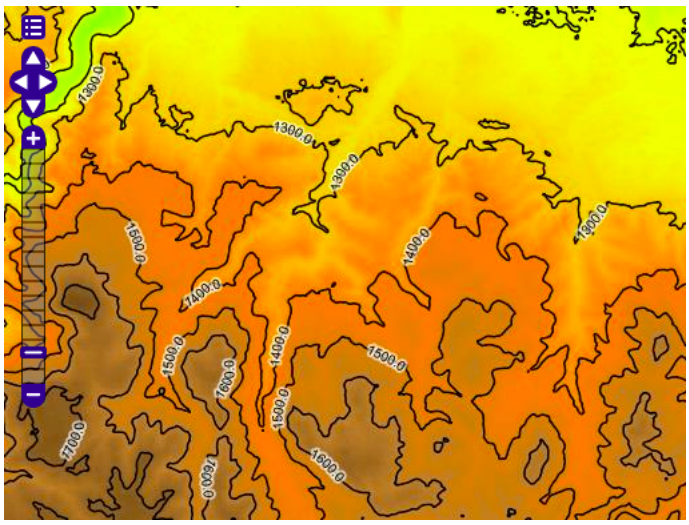
```
    xmlns="http://www.opengis.net/sld"
    xmlns:ogc="http://www.opengis.net/ogc"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <NamedLayer>
      <Name>contour_dem</Name>
      <UserStyle>
        <Title>Contour DEM</Title>
        <Abstract>Extracts contours from DEM</Abstract>
        <FeatureTypeStyle>
          <Transformation>
            <ogc:Function name="gs:Contour">
              <ogc:Function name="parameter">
                <ogc:Literal>data</ogc:Literal>
              </ogc:Function>
              <ogc:Function name="parameter">
                <ogc:Literal>levels</ogc:Literal>
                <ogc:Literal>1100</ogc:Literal>
                <ogc:Literal>1200</ogc:Literal>
                <ogc:Literal>1300</ogc:Literal>
                <ogc:Literal>1400</ogc:Literal>
                <ogc:Literal>1500</ogc:Literal>
                <ogc:Literal>1600</ogc:Literal>
                <ogc:Literal>1700</ogc:Literal>
                <ogc:Literal>1800</ogc:Literal>
              </ogc:Function>
            </ogc:Function>
          </Transformation>
          <Rule>
            <Name>rule1</Name>
            <Title>Contour Line</Title>
            <LineSymbolizer>
              <Stroke>
                <CssParameter name="stroke">#000000</CssParameter>
                <CssParameter name="stroke-width">1</CssParameter>
              </Stroke>
            </LineSymbolizer>
            <TextSymbolizer>
              <Label>
                <ogc:PropertyName>value</ogc:PropertyName>
              </Label>
              <Font>
                <CssParameter name="font-family">Arial</CssParameter>
                <CssParameter name="font-style">Normal</CssParameter>
                <CssParameter name="font-size">10</CssParameter>
              </Font>
              <LabelPlacement>
                <LinePlacement/>
              </LabelPlacement>
              <Halo>
                <Radius>
                  <ogc:Literal>2</ogc:Literal>
                </Radius>
                <Fill>
                  <CssParameter name="fill">#FFFFFF</CssParameter>
                  <CssParameter name="fill-opacity">0.6</CssParameter>
                </Fill>
              </Halo>
```

```
62                <Fill>
63                  <CssParameter name="fill">#000000</CssParameter>
64                </Fill>
65                <Priority>2000</Priority>
66                <VendorOption name="followLine">true</VendorOption>
67                <VendorOption name="repeat">100</VendorOption>
68                <VendorOption name="maxDisplacement">50</VendorOption>
69                <VendorOption name="maxAngleDelta">30</VendorOption>
70              </TextSymbolizer>
71            </Rule>
72          </FeatureTypeStyle>
73        </UserStyle>
74      </NamedLayer>
75    </StyledLayerDescriptor>
```

Key aspects of the SLD are:

- **Lines 14-15** define the rendering transformation, using the process `gs:Contour`.
- **Lines 16-18** supply the input data parameter, named `data` in this process.
- **Lines 19-29** supply values for the process's `levels` parameter, which specifies the elevation levels for the contours to extract.
- **Lines 35-40** specify a `LineSymbolizer` to style the contour lines.
- **Lines 41-70** specify a `TextSymbolizer` to show the contour levels along the lines.

The result of using this transformation is shown in the following map image (which also shows the underlying DEM raster):



Heatmap generation

`gs:Heatmap` is a **Vector-to-Raster** rendering transformation which generates a heatmap surface from weighted point data. The following SLD invokes a Heatmap rendering transformation on a featuretype with point geometries and an attribute `pop2000` supplying the weight for the points (in this example, a dataset of world urban areas is used). The output is styled using a color ramp across the output data value range [0 .. 1].

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StyledLayerDescriptor version="1.0.0"
    xsi:schemaLocation="http://www.opengis.net/sld StyledLayerDescriptor.xsd"
    xmlns="http://www.opengis.net/sld"
    xmlns:ogc="http://www.opengis.net/ogc"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <NamedLayer>
    <Name>Heatmap</Name>
    <UserStyle>
      <Title>Heatmap</Title>
      <Abstract>A heatmap surface showing population density</Abstract>
      <FeatureTypeStyle>
        <Transformation>
          <ogc:Function name="gs:Heatmap">
            <ogc:Function name="parameter">
              <ogc:Literal>data</ogc:Literal>
            </ogc:Function>
            <ogc:Function name="parameter">
              <ogc:Literal>weightAttr</ogc:Literal>
              <ogc:Literal>pop2000</ogc:Literal>
            </ogc:Function>
            <ogc:Function name="parameter">
              <ogc:Literal>radiusPixels</ogc:Literal>
              <ogc:Function name="env">
                <ogc:Literal>radius</ogc:Literal>
                <ogc:Literal>100</ogc:Literal>
              </ogc:Function>
            </ogc:Function>
            <ogc:Function name="parameter">
              <ogc:Literal>pixelsPerCell</ogc:Literal>
              <ogc:Literal>10</ogc:Literal>
            </ogc:Function>
            <ogc:Function name="parameter">
              <ogc:Literal>outputBBOX</ogc:Literal>
              <ogc:Function name="env">
                <ogc:Literal>wms_bbox</ogc:Literal>
              </ogc:Function>
            </ogc:Function>
            <ogc:Function name="parameter">
              <ogc:Literal>outputWidth</ogc:Literal>
              <ogc:Function name="env">
                <ogc:Literal>wms_width</ogc:Literal>
              </ogc:Function>
            </ogc:Function>
            <ogc:Function name="parameter">
              <ogc:Literal>outputHeight</ogc:Literal>
              <ogc:Function name="env">
                <ogc:Literal>wms_height</ogc:Literal>
              </ogc:Function>
            </ogc:Function>
          </ogc:Function>
        </Transformation>
        <Rule>
          <RasterSymbolizer>
          <!-- specify geometry attribute to pass validation -->
            <Geometry>
              <ogc:PropertyName>the_geom</ogc:PropertyName></Geometry>
```

```
59              <Opacity>0.6</Opacity>
60              <ColorMap type="ramp" >
61                <ColorMapEntry color="#FFFFFF" quantity="0" label="nodata"
62                  opacity="0"/>
63                <ColorMapEntry color="#FFFFFF" quantity="0.02" label="nodata"
64                  opacity="0"/>
65                <ColorMapEntry color="#4444FF" quantity=".1" label="nodata"/>
66                <ColorMapEntry color="#FF0000" quantity=".5" label="values" />
67                <ColorMapEntry color="#FFFF00" quantity="1.0" label="values" />
68              </ColorMap>
69            </RasterSymbolizer>
70          </Rule>
71        </FeatureTypeStyle>
72      </UserStyle>
73    </NamedLayer>
74  </StyledLayerDescriptor>
```

Key aspects of the SLD are:

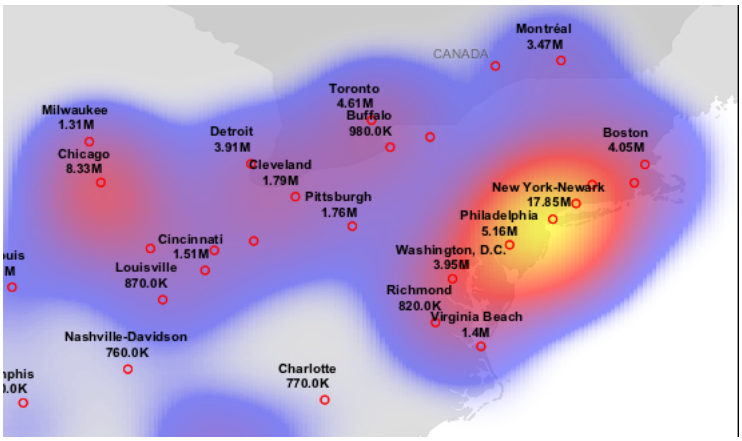
- **Lines 14-15** define the rendering transformation, using the process `gs:Heatmap`.
- **Lines 16-18** supply the input data parameter, named `data` in this process.
- **Lines 19-22** supply a value for the process's `weightAttr` parameter, which specifies the input attribute providing a weight for each data point.
- **Lines 23-29** supply the value for the `radiusPixels` parameter, which controls the "spread" of the heatmap around each point. In this SLD the value of this parameter may be supplied by a SLD substitution variable called `radius`, with a default value of `100` pixels.
- **Lines 30-33** supply the `pixelsPerCell` parameter, which controls the resolution at which the heatmap raster is computed.
- **Lines 34-38** supply the `outputBBOX` parameter, which is given the value of the standard SLD environment variable `wms_bbox`.
- **Lines 40-45** supply the `outputWidth` parameter, which is given the value of the standard SLD environment variable `wms_width`.
- **Lines 46-52** supply the `outputHeight` parameter, which is given the value of the standard SLD environment variable `wms_height`.
- **Lines 55-70** specify a `RasterSymbolizer` to style the computed raster surface. The symbolizer contains a ramped color map for the data range [0..1].
- **Line 58** specifies the geometry attribute of the input featuretype, which is necessary to pass SLD validation.

This transformation styles a layer to produce a heatmap surface for the data in the requested map extent, as shown in the image below. (The map image also shows the original input data points styled by another SLD, as well as a base map layer.)

12.5.3 Graphic symbology in GeoServer

Graphic symbology is supported via the SLD `<Graphic>` element. This element can appear in several contexts in SLD:

- in a *PointSymbolizer*, to display symbols at points
- in the `<Stroke>/<GraphicStroke>` element of a *LineSymbolizer* and *PolygonSymbolizer*, to display repeated symbols along lines and polygon boundaries.



- in the `<Stroke>/<GraphicFill>` element of a *LineSymbolizer* and *PolygonSymbolizer*, to fill lines and polygon boundaries with tiled symbols.
- in the `<Fill>/<GraphicFill>` element of a *PolygonSymbolizer*, to fill polygons with tiled symbols (stippling).
- in a *TextSymbolizer* to display a graphic behind or instead of text labels (this is a GeoServer extension).

`<Graphic>` contains either a `<Mark>` or an `<ExternalGraphic>` element. **Marks** are pure vector symbols whose geometry is predefined but with stroke and fill defined in the SLD itself. **External Graphics** are external files (such as PNG images or SVG graphics) that contain the shape and color information defining how to render a symbol.

In standard SLD the `<Mark>` and `<ExternalGraphic>` names are fixed strings. GeoServer extends this by providing *dynamic symbolizers*, which allow computing symbol names on a per-feature basis by embedding CQL expressions in them.

Marks

GeoServer supports the standard SLD `<Mark>` symbols, a user-expandable set of extended symbols, and also TrueType Font glyphs. The symbol names are specified in the `<WellKnownName>` element.

See also the *PointSymbolizer* reference for further details, as well as the examples in the *Points* Cookbook section.

Standard symbols

The SLD specification mandates the support of the following symbols:

Name	Description
`square`	A square
`circle`	A circle
`triangle`	A triangle pointing up
`star`	five-pointed star
`cross`	A square cross with space around (not suitable for hatch fills)
`x`	A square X with space around (not suitable for hatch fills)

Shape symbols

The shape symbols set adds extra symbols that are not part of the basic set. Their names are prefixed by `shape://`

Name	Description
`shape://vertline`	A vertical line (suitable for hatch fills or to make railroad symbols)
`shape://horline`	A horizontal line (suitable for hatch fills)
`shape://slash`	A diagonal line leaning forwards like the "slash" keyboard symbol (suitable for diagonal hatches)
`shape://backslash`	Same as `shape://slash`, but oriented in the opposite direction
`shape://dot`	A very small circle with space around
`shape://plus`	A + symbol, without space around (suitable for cross-hatch fills)
`shape://times`	A "X" symbol, without space around (suitable for cross-hatch fills)
`shape://oarrow`	An open arrow symbol (triangle without one side, suitable for placing arrows at the end of lines)
`shape://carrow`	A closed arrow symbol (closed triangle, suitable for placing arrows at the end of lines)

TTF marks

It is possible to create a mark using glyphs from any decorative or symbolic True Type Font, such as Wingdings, WebDings, or the many symbol fonts available on the internet. The syntax for specifying this is:

```
ttf://<fontname>#<hexcode>
```

where `fontname` is the full name of a TTF font available to GeoServer, and `hexcode` is the hexadecimal code of the symbol. To get the hex code of a symbol, use the "Char Map" utility available in most operating systems (Windows and Linux Gnome both have one).

For example, to use the "shield" symbol contained in the WebDings font, the Gnome `charmap` reports the symbol hex code as shown:

The SLD to use the shield glyph as a symbol is:

```
<PointSymbolizer>
    <Graphic>
      <Mark>
        <WellKnownName>ttf://Webdings#0x0064</WellKnownName>
        <Fill>
          <CssParameter name="fill">#AAAAAA</CssParameter>
        </Fill>
        <Stroke/>
      </Mark>
    <Size>16</Size>
  </Graphic>
</PointSymbolizer>
```

This results in the following map display:

Extending the Mark subsytem using Java

The Mark subsystem is user-extensible. To do this using Java code, implement the `MarkFactory` interface and declare the implementation in the `META-INF/services/org.geotools.renderer.style.MarkFactory` file.

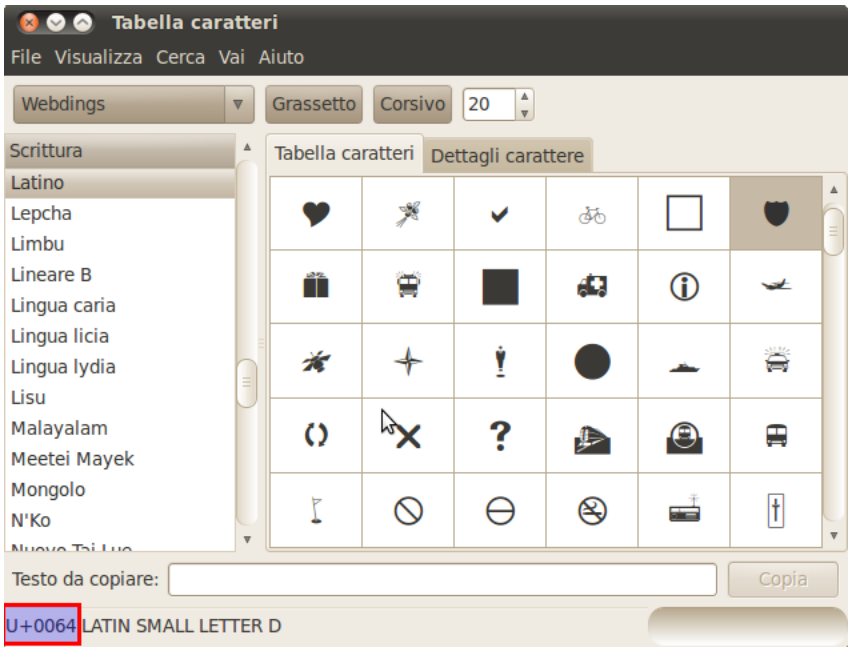


Figure 12.54: *Selecting a symbol hex code in the Gnome char map*

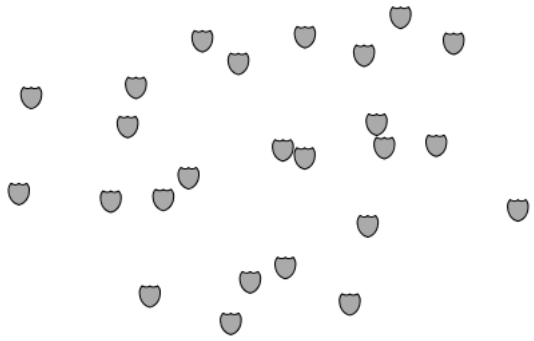


Figure 12.55: *Shield symbols rendered on the map*

For further information see the Javadoc of the GeoTools MarkFactory, along with the following example code:

- The factory SPI registration file
- The TTFMarkFactory implementation
- The ShapeMarkFactory implementation

External Graphics

`<ExternalGraphic>` is the other way to define point symbology. Unlike marks, external graphics are used as-is, so the specification is somewhat simpler. The element content specifies a graphic `<OnlineResource>` using a URL or file path, and the graphic `<Format>` using a MIME type:

```
<PointSymbolizer>
    <Graphic>
       <ExternalGraphic>
          <OnlineResource xlink:type="simple" xlink:href="http://mywebsite.com/pointsymbol.png" />
          <Format>image/png</Format>
       </ExternalGraphic>
    </Graphic>
</PointSymbolizer>
```

As with `<Mark>`, a `<Size>` element can be optionally specified. When using images as graphic symbols it is better to avoid resizing, as that may blur their appearance. Use images at their native resolution by omitting the `<Size>` element. In contrast, for SVG graphics specifying a `<Size>` is recommended. SVG files are a vector-based format describing both shape and color, so they scale cleanly to any size.

If the path of the symbol file is relative, the file is looked for under `$GEOSERVER_DATA_DIR/styles`. For example:

```
<PointSymbolizer>
  <Graphic>
    <ExternalGraphic>
      <OnlineResource xlink:type="simple" xlink:href="burg02.svg" />
      <Format>image/svg+xml</Format>
    </ExternalGraphic>
    <Size>20</Size>
  </Graphic>
</PointSymbolizer>
```

In this example an SVG graphic is being used, so the size is specified explicitly.

Symbol Positioning

Graphic symbols are rendered so that the center of the graphic extent lies on the placement point (or points, in the case of repeated or tiled graphics). If it is desired to have a graphic offset from a point (such as a symbol which acts as a pointer) it is necessary to offset the visible portion of the graphic within the overall extent. For images this can be accomplished by extending the image with transparent pixels. For SVG graphics this can be done by surrounding the shape with an invisible rectangle with the desired relative position.

Dynamic symbolizers

In standard SLD, the `Mark/WellKnowName` element and the `ExternalGraphic/OnlineResource/@xlink:href` attribute are fixed strings. This means they have the same value for all rendered features. When the symbols to be displayed vary depending on feature attributes this restriction leads to very verbose styling, as a separate `Rule` and `Symbolizer` must be used for each different symbol.

GeoServer improves this by allowing *CQL expressions* to be embedded inside the content of both `WellKnownName` and `OnlineResource/@xlink:href`. When the names of the symbols can be derived from the feature attribute values, this provides much more compact styling. CQL expressions can be embedded in a `<WellKnownName>` content string or an `<OnlineResource>` `xlink:href` attribute by using the syntax:

```
${<cql expression>}
```

Note: Currently `xlink:href` strings must be valid URLs *before* expression expansion is performed. This means that the URL cannot be completely provided by an expression. The `xlink:href` string must explicitly include at least the prefix `http://`

The simplest form of expression is a single attribute name, such as `${STATE_ABBR}`. For example, suppose we want to display the flags of the US states using symbols whose file names match the state name. The following style specifies the flag symbols using a single rule:

```
<ExternalGraphic>
   <OnlineResource xlink:type="simple"
                   xlink:href="http://mysite.com/tn_${STATE_ABBR}.jpg"/>
   <Format>image/jpeg</Format>
</ExternalGraphic>
```

If manipulation of the attribute values is required a full CQL expression can be specified. For example, if the values in the `STATE_ABBR` attribute are uppercase but the URL requires a lowercase name, the CQL `strToLowerCase` function can be used:

```
<ExternalGraphic>
   <OnlineResource xlink:type="simple"
            xlink:href="http://mysite.com/tn_${strToLowerCase(STATE_ABBR)}.jpg" />
   <Format>image/jpeg</Format>
</ExternalGraphic>
```

12.5.4 Variable substitution in SLD

Variable substitution in SLD is a GeoServer extension (starting in version 2.0.2) that allows passing values from WMS requests into SLD styles. This allows dynamically setting values such as colors, fonts, sizes and filter thresholds.

Variables are specified in WMS `GetMap` requests by using the `env` request parameter followed by a list of `name:value` pairs separated by semicolons:

```
...&env=name1:value1;name2=value2&...
```

In an SLD the variable values are accessed using the `env` function. The function retrieves a substitution variable value specified in the current request:

```
<ogc:Function name="env">
   <ogc:Literal>size</ogc:Literal>
</ogc:Function>
```

A default value can be provided. It will be used if the variable was not specified in the request:

```
<ogc:Function name="env">
   <ogc:Literal>size</ogc:Literal>
   <ogc:Literal>6</ogc:Literal>
</ogc:Function>
```

The `env` function can be used in an SLD anywhere an OGC expression is allowed. For example, it can be used in `CSSParameter` elements, in size and offset elements, and in rule filter expressions. It is also accepted in some places where full expressions are not allowed, such as in the `Mark/WellKnownName` element.

Predefined Variables

GeoServer has predefined variables which provide information about specific properties of the request output. These are useful when SLD parameters need to depend on output dimensions. The predefined variables are:

Name	Type	Description
`wms_bbox`	`ReferencedEnvelope`	the georeferenced extent of the request output
`wms_crs`	`CoordinateReferenceSystem`	the definition of the output coordinate reference system
`wms_srs`	`String`	the code for the output coordinate reference system
`wms_width`	`Integer`	the width (in pixels) of the output image
`wms_height`	`Integer`	the height (in pixels) of the output image
`wms_scale_denominator`	`Integer`	the denominator of the output map scale
`kmlOutputMode`	Either `vector` or empty	this variable gets set to `vector` when the kml generator is writing out vector features as placemarks, as opposed to ground overlays

Example

The following SLD symbolizer has been parameterized in three places, with default values provided in each case:

```
<PointSymbolizer>
  <Graphic>
    <Mark>
      <WellKnownName><ogc:Function name="env">
            <ogc:Literal>name</ogc:Literal>
            <ogc:Literal>square</ogc:Literal>
         </ogc:Function>
      </WellKnownName>
      <Fill>
        <CssParameter name="fill">
          #<ogc:Function name="env">
            <ogc:Literal>color</ogc:Literal>
            <ogc:Literal>FF0000</ogc:Literal>
          </ogc:Function>
        </CssParameter>
      </Fill>
    </Mark>
    <Size>
       <ogc:Function name="env">
          <ogc:Literal>size</ogc:Literal>
          <ogc:Literal>6</ogc:Literal>
       </ogc:Function>
    </Size>
```

```
  </Graphic>
</PointSymbolizer>
```

`Download the full SLD style`

When no variables are provided in the WMS request, the SLD uses the default values and renders the sample `sf:bugsites` dataset as shown:

Figure 12.56: *Default rendering*

If the request is changed to specify the following variable values:

```
&env=color:00FF00;name:triangle;size:12
```

the result is instead:

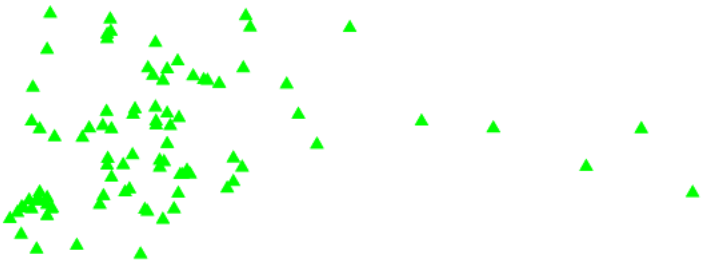


Figure 12.57: *Rendering with varialbes supplied*

12.5.5 Specifying symbolizer sizes in ground units

The SLD 1.0 specification allows giving symbolizer sizes in a single unit of measure: pixels. This means that the size of symbolizers is the same at all zoom levels (which is commonly the desired behaviour).

The Symbology Encoding 1.1 specification provides a `uom` attribute on `Symbolizer` elements. This allows specifying styling parameter sizes in ground units of metres or feet, as well as the default screen pixels.

When ground units are used, the screen size of styled elements increases as the map is zoomed in to larger scales. GeoServer supports the SE 1.1 `uom` attribute in its extended SLD 1.0 support.

Note: This extended feature is officially supported in GeoServer 2.1.0. It is available in GeoServer 2.0.3 if the `-DenableDpiUomRescaling=true` system variable is specified for the JVM.

The value of the `uom` attribute is a URI indicating the desired unit. The units of measure supported are those given in the SE 1.1 specification:

```
http://www.opengeospatial.org/se/units/metre
http://www.opengeospatial.org/se/units/foot
http://www.opengeospatial.org/se/units/pixel
```

Note: The `px` override modifier for parameters values is not currently supported.

Example

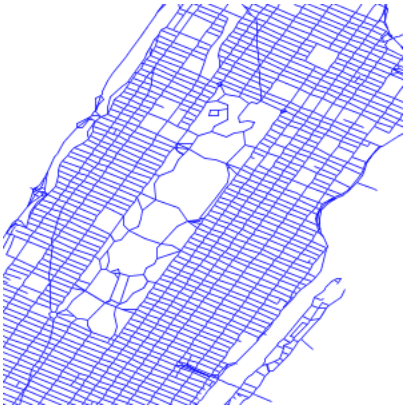
The following SLD shows the `uom` attribute used to specify the width of a `LineSymbolizer` in metres:

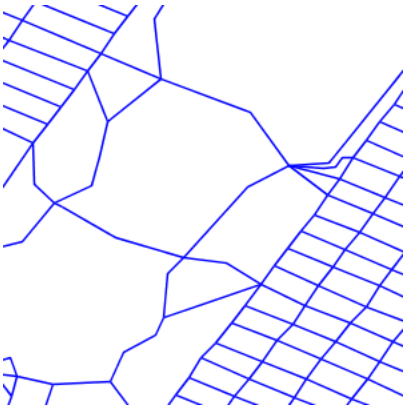
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StyledLayerDescriptor version="1.0.0" xmlns="http://www.opengis.net/sld" xmlns:ogc="http://www.openg
  xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <NamedLayer>
    <Name>5m blue line</Name>
    <UserStyle>
      <Title>tm blue line</Title>
      <Abstract>Default line style, 5m wide blue</Abstract>

      <FeatureTypeStyle>
        <Rule>
          <Title>Blue Line, 5m large</Title>
          <LineSymbolizer uom="http://www.opengeospatial.org/se/units/metre">
            <Stroke>
              <CssParameter name="stroke">#0000FF</CssParameter>
              <CssParameter name="stroke-width">5</CssParameter>
            </Stroke>
          </LineSymbolizer>
        </Rule>

      </FeatureTypeStyle>
    </UserStyle>
  </NamedLayer>
</StyledLayerDescriptor>
```

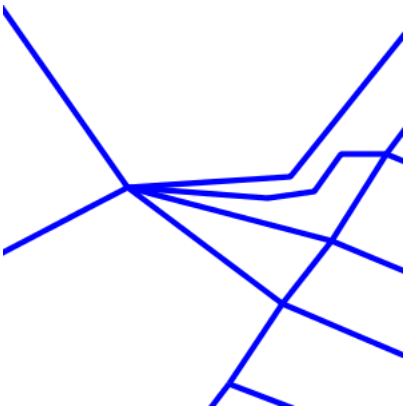
Applying the style to the `tiger:tiger_roads` dataset shows how the line widths increase as the map is zoomed in:

12.5.6 Label Obstacles

GeoServer implements an algorithm for label conflict resolution, to prevent labels from overlapping one another. By default this algorithm only considers conflicts with other labels. This can result in labels overlapping other symbolizers, which may produce an undesirable effect.







GeoServer supports a vendor option called `labelObstacle` that allows marking a symbolizer as an obstacle. This tells the labeller to avoid rendering labels that overlap it.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StyledLayerDescriptor version="1.0.0" xmlns="http://www.opengis.net/sld" xmlns:ogc="http://www.openg
  xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <NamedLayer>
    <UserStyle>

  <FeatureTypeStyle>
    <Rule>
      <PointSymbolizer>
        <Graphic>
          <ExternalGraphic>
            <OnlineResource
              xlink:type="simple"
              xlink:href="smileyface.png" />
            <Format>image/png</Format>
          </ExternalGraphic>
          <Size>32</Size>
        </Graphic>
        <VendorOption name="labelObstacle">true</VendorOption>
      </PointSymbolizer>
    </Rule>
  </FeatureTypeStyle>

    </UserStyle>
  </NamedLayer>
</StyledLayerDescriptor>
```

Applying the obstacle to a regular point style:

```
<PointSymbolizer>
  <Graphic>
    <ExternalGraphic>
      <OnlineResource
        xlink:type="simple"
        xlink:href="smileyface.png" />
      <Format>image/png</Format>
    </ExternalGraphic>
    <Size>32</Size>
  </Graphic>
  <VendorOption name="labelObstacle">true</VendorOption>
</PointSymbolizer>
```

Applying the obstacle to line/polygon style style:

Warning: Beware of marking a line or poly symbolizer as a label obstacle. The label conflict resolving routine is based on the bounding box so marking as a label obstacle will result in no label overlapping not only the geometry itself, but its bounding box as well.

12.5.7 Adding space around graphic fills

Starting with GeoServer 2.3.4 it is possible to add white space around symbols used inside graphic fills, effectively allowing to control the density of the symbols in the map.

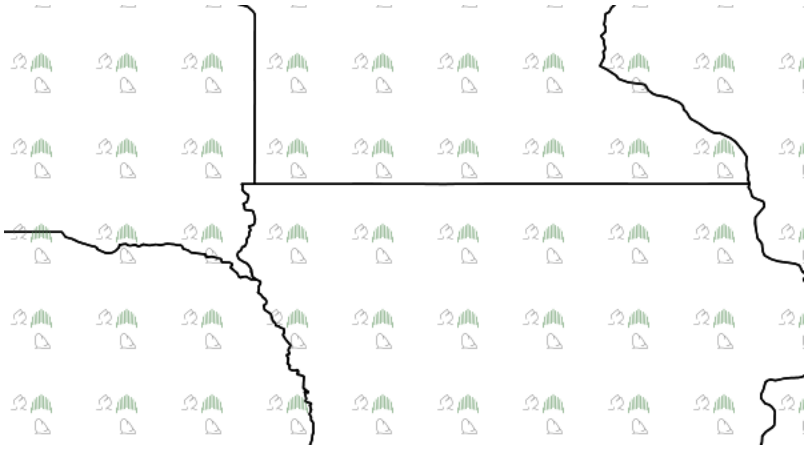
```
<PolygonSymbolizer>
  <Fill>
    <GraphicFill>
      <Graphic>
        <ExternalGraphic>
          <OnlineResource xlink:type="simple" xlink:href="./rockFillSymbol.png"/>
          <Format>image/png</Format>
        </ExternalGraphic>
      </Graphic>
    </GraphicFill>
  </Fill>
  <VendorOption name="graphic-margin">10</VendorOption>
</PolygonSymbolizer>
```

The above forces 10 pixels of white space above, below and on either side of the symbol, effectively adding 20 pixels of white space between the symbols in the fill. The `graphic-margin` can be expressed, just like the CSS margin, in four different ways:

- top,right,bottom,left (one explicit value per margin)
- top,right-left,bottom (three values, with right and left sharing the same value)
- top-bottom,right-left (two values, top and bottom sharing the same value)
- top-right-bottom-left (single value for all four margins)

The ability to specify different margins allows to use more than one symbol in a fill, and synchronize the relative positions of the various symbols to generate a composite fill:

```
<PolygonSymbolizer>
  <Fill>
    <GraphicFill>
      <Graphic>
        <ExternalGraphic>
          <OnlineResource xlink:type="simple" xlink:href="./boulderGeometry.png"/>
          <Format>image/png</Format>
        </ExternalGraphic>
      </Graphic>
    </GraphicFill>
  </Fill>
  <VendorOption name="graphic-margin">35 17 17 35</VendorOption>
</PolygonSymbolizer>
<PolygonSymbolizer>
  <Fill>
    <GraphicFill>
      <Graphic>
        <ExternalGraphic>
          <OnlineResource xlink:type="simple" xlink:href="./roughGrassFillSymbol.png"/>
          <Format>image/png</Format>
        </ExternalGraphic>
      </Graphic>
    </GraphicFill>
  </Fill>
  <VendorOption name="graphic-margin">16 16 32 32</VendorOption>
</PolygonSymbolizer>
```



12.5.8 Fills with randomized symbols

Starting with GeoServer 2.4.2 it is possible to generate fills by randomly repeating a symbol in the polygons to be filled. Or, to be more precise, generate the usual texture fill by repeating over and over a tile, whose contents is the random repetition of a fill. The random distribution is stable, so it will be the same across calls and tiles, and it's controlled by the seed used to generate the distribution.

The random fill is generated by specifying a GraphicFill with a Mark or ExternalGraphic, and then adding vendor options to control how the symbol is randomly repeated. Here is a table with options, default values, and possible values:

Option	Default value	Description
random	none	Activates random distribution of symbol. Possible values are **none**, **free**, **grid**. **none** disables random distribution, **free** generates a completely random distribution, **grid** will generate a regular grid of positions, and only randomizes the position of the symbol around the cell centers, providing a more even distribution in space
random-tile-size	256	Size the the texture fill tile that will contain the randomly distributed symbols
random-rotation	none	Activates random symbol rotation. Possible values are **none** (no rotation) or **free**
random-symbol-count	16	The number of symbols in the tile. The number of symbols actually painted can be lower, as the distribution will ensure no two symbols overlap with each other.
random-seed	0	The "seed" used to generate the random distribution. Changing this value will result in a different symbol distribution

Here is an example:

```
<sld:PolygonSymbolizer>
 <sld:Fill>
   <sld:GraphicFill>
     <sld:Graphic>
       <sld:Mark>
         <sld:WellKnownName>shape://slash</sld:WellKnownName>
         <sld:Stroke>
           <sld:CssParameter name="stroke">#0000ff</sld:CssParameter>
           <sld:CssParameter name="stroke-linecap">round</sld:CssParameter>
           <sld:CssParameter name="stroke-width">4</sld:CssParameter>
```

```
          </sld:Stroke>
        </sld:Mark>
        <sld:Size>8</sld:Size>
      </sld:Graphic>
    </sld:GraphicFill>
  </sld:Fill>
  <sld:VendorOption name="random-seed">5</sld:VendorOption>
  <sld:VendorOption name="random">grid</sld:VendorOption>
  <sld:VendorOption name="random-tile-size">100</sld:VendorOption>
  <sld:VendorOption name="random-rotation">free</sld:VendorOption>
  <sld:VendorOption name="random-symbol-count">50</sld:VendorOption>
</sld:PolygonSymbolizer>
```

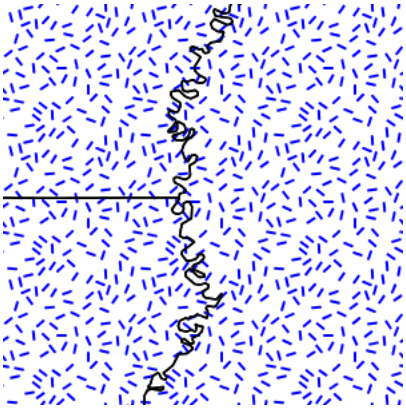


Figure 12.58: *Random distribution of a diagonal line*

Randomized distributions can also be used for thematic mapping, for example, here is the SLD for a version of topp:states that displays the number of inhabitantìs varying the density of a random point distribution:

```
<?xml version="1.0" encoding="UTF-8"?>
<sld:UserStyle xmlns="http://www.opengis.net/sld" xmlns:sld="http://www.opengis.net/sld" xmlns:ogc="h
  <sld:Name>Default Styler</sld:Name>
  <sld:FeatureTypeStyle>
    <sld:Name>name</sld:Name>
    <sld:Rule>
      <ogc:Filter>
        <ogc:And>
          <ogc:Not>
            <ogc:PropertyIsLessThan>
              <ogc:PropertyName>PERSONS</ogc:PropertyName>
              <ogc:Literal>2000000</ogc:Literal>
            </ogc:PropertyIsLessThan>
          </ogc:Not>
          <ogc:Not>
            <ogc:PropertyIsGreaterThanOrEqualTo>
              <ogc:PropertyName>PERSONS</ogc:PropertyName>
              <ogc:Literal>4000000</ogc:Literal>
            </ogc:PropertyIsGreaterThanOrEqualTo>
          </ogc:Not>
        </ogc:And>
      </ogc:Filter>
      <sld:PolygonSymbolizer>
        <sld:Fill>
          <sld:GraphicFill>
```

```
          <sld:Graphic>
            <sld:Mark>
              <sld:WellKnownName>circle</sld:WellKnownName>
              <sld:Fill>
                <sld:CssParameter name="fill">#a9a9a9</sld:CssParameter>
              </sld:Fill>
            </sld:Mark>
            <sld:Size>2</sld:Size>
          </sld:Graphic>
        </sld:GraphicFill>
      </sld:Fill>
      <sld:VendorOption name="random">grid</sld:VendorOption>
      <sld:VendorOption name="random-tile-size">100</sld:VendorOption>
      <sld:VendorOption name="random-symbol-count">150</sld:VendorOption>
    </sld:PolygonSymbolizer>
    <sld:LineSymbolizer>
      <sld:Stroke/>
    </sld:LineSymbolizer>
  </sld:Rule>
  <sld:Rule>
    <ogc:Filter>
      <ogc:PropertyIsLessThan>
        <ogc:PropertyName>PERSONS</ogc:PropertyName>
        <ogc:Literal>2000000</ogc:Literal>
      </ogc:PropertyIsLessThan>
    </ogc:Filter>
    <sld:PolygonSymbolizer>
      <sld:Fill>
        <sld:GraphicFill>
          <sld:Graphic>
            <sld:Mark>
              <sld:WellKnownName>circle</sld:WellKnownName>
              <sld:Fill>
                <sld:CssParameter name="fill">#a9a9a9</sld:CssParameter>
              </sld:Fill>
            </sld:Mark>
            <sld:Size>2</sld:Size>
          </sld:Graphic>
        </sld:GraphicFill>
      </sld:Fill>
      <sld:VendorOption name="random">grid</sld:VendorOption>
      <sld:VendorOption name="random-tile-size">100</sld:VendorOption>
      <sld:VendorOption name="random-symbol-count">50</sld:VendorOption>
    </sld:PolygonSymbolizer>
    <sld:LineSymbolizer>
      <sld:Stroke/>
    </sld:LineSymbolizer>
  </sld:Rule>
  <sld:Rule>
    <ogc:Filter>
      <ogc:PropertyIsGreaterThanOrEqualTo>
        <ogc:PropertyName>PERSONS</ogc:PropertyName>
        <ogc:Literal>4000000</ogc:Literal>
      </ogc:PropertyIsGreaterThanOrEqualTo>
    </ogc:Filter>
    <sld:PolygonSymbolizer>
      <sld:Fill>
        <sld:GraphicFill>
```

```
            <sld:Graphic>
              <sld:Mark>
                <sld:WellKnownName>circle</sld:WellKnownName>
                <sld:Fill>
                  <sld:CssParameter name="fill">#a9a9a9</sld:CssParameter>
                </sld:Fill>
              </sld:Mark>
              <sld:Size>2</sld:Size>
            </sld:Graphic>
          </sld:GraphicFill>
        </sld:Fill>
        <sld:VendorOption name="random">grid</sld:VendorOption>
        <sld:VendorOption name="random-tile-size">100</sld:VendorOption>
        <sld:VendorOption name="random-symbol-count">500</sld:VendorOption>
      </sld:PolygonSymbolizer>
      <sld:LineSymbolizer>
        <sld:Stroke/>
      </sld:LineSymbolizer>
    </sld:Rule>
  </sld:FeatureTypeStyle>
</sld:UserStyle>
```

Figure 12.59: *Thematic map via point density approach*

12.6 SLD Tips and Tricks

This section details various advanced strategies for working with SLD.

12.6.1 Styling mixed geometry types

On occasion one might need to style a geometry column whose geometry type can be different for each feature (some are polygons, some are points, etc), and use different styling for different geometry types.

SLD 1.0 does not provide a clean solution for dealing with this situation. Point, Line, and Polygon symbolizers do not select geometry by type, since each can apply to all geometry types:

- Point symbolizers apply to any kind of geometry. If the geometry is not a point, the centroid of the geometry is used.
- Line symbolizers apply to both lines and polygons. For polygons the boundary is styled.
- Polygon symbolizers apply to lines, by adding a closing segment connecting the first and last points of the line.

There is also no standard filter predicate to identify geometry type which could be used in rules.

This section suggests a number of ways to accomplish styling by geometry type. They require either data restructuring or the use of non-standard filter functions.

Restructuring the data

There are a few ways to restructure the data so that it can be styled by geometry type using only standard SLD constructs.

Split the table

The first and obvious one is to split the original table into a set of separate tables, each one containing a single geometry type. For example, if table `findings` has a geometry column that can contain point, lines, and polygons, three tables can be created, each one containing a single geometry type.

Separate geometry columns

A second way is to use one table and separate geometry columns. So, if the table `findings` has a `geom` column, the restructured table will have `point`, `line` and `polygon` columns, each of them containing just one geometry type. After the restructuring, the symbolizers will refer to a specific geometry, for example:

```
<PolygonSymbolizer>
    <Geometry><ogc:PropertyName>polygon</ogc:PropertyName></Geometry>
</PolygonSymbolizer>
```

This way each symbolizer will match only the geometry types it is supposed to render, and skip over the rows that contain a null value.

Add a geometry type column

A third way is to add a geometry type column allowing standard filtering constructs to be used, and then build a separate rule per geometry type. In the example above a new attribute, `gtype` will be added containing the values `Point`, `Line` and `Polygon`. The following SLD template can be used after the change:

```
<Rule>
   <ogc:Filter>
      <ogc:PropertyIsEqualTo>
         <ogc:PropertyName>gtype</ogc:PropertyName>
         <ogc:Literal>Point</ogc:Literal>
      </ogc:PropertyIsEqualTo>
   </ogc:Filter>
   <PointSymbolizer>
      ...
   </PointSymbolizer>
</Rule>
<Rule>
   <ogc:Filter>
      <ogc:PropertyIsEqualTo>
         <ogc:PropertyName>gtype</ogc:PropertyName>
         <ogc:Literal>Line</ogc:Literal>
      </ogc:PropertyIsEqualTo>
```

```
   </ogc:Filter>
   <LineSymbolizer>
      ...
   </LineSymbolizer>
</Rule>
<Rule>
   <ogc:Filter>
      <ogc:PropertyIsEqualTo>
         <ogc:PropertyName>gtype</ogc:PropertyName>
         <ogc:Literal>Polygon</ogc:Literal>
      </ogc:PropertyIsEqualTo>
   </ogc:Filter>
   <PolygonSymbolizer>
      ...
   </PolygonSymbolizer>
</Rule>
```

The above suggestions assume that restructuring the data is technically possible. This is usually true in spatial databases that provide functions that allow determining the geometry type.

Create views

A less invasive way to get the same results without changing the structure of the table is to create views that have the required structure. This allows the original data to be kept intact, and the views may be used for rendering.

Using SLD rules and filter functions

SLD 1.0 uses the OGC Filter 1.0 specification for filtering out the data to be styled by each rule. Filters can contain *Filter functions* to compute properties of geometric values. In GeoServer, filtering by geometry type can be done using the `geometryType` or `dimension` filter functions.

Note: The Filter Encoding specification provides a standard syntax for filter functions, but does not mandate a specific set of functions. SLDs using these functions may not be portable to other styling software.

---

geometryType function

The `geometryType` function takes a geometry property and returns a string, which (currently) is one of the values `Point`, `LineString`, `LinearRing`, `Polygon`, `MultiPoint`, `MultiLineString`, `MultiPolygon` and `GeometryCollection`.

Using this function, a `Rule` matching only single points can be written as:

```
<Rule>
   <ogc:PropertyIsEqualTo>
      <ogc:Function name="geometryType">
         <ogc:PropertyName>geom</ogc:PropertyName>
      </ogc:Function>
      <ogc:Literal>Point</ogc:Literal>
   </ogc:PropertyIsEqualTo>
   <PointSymbolizer>
     ...
   </PointSymbolizer>
</Rule>
```

The filter is more complex if it has to match all linear geometry types. In this case, it looks like:

```
<Rule>
   <ogc:Filter>
     <ogc:PropertyIsEqualTo>
       <ogc:Function name="in3">
          <ogc:Function name="geometryType">
              <ogc:PropertyName>geom</ogc:PropertyName>
          </ogc:Function>
          <ogc:Literal>LineString</ogc:Literal>
          <ogc:Literal>LinearRing</ogc:Literal>
          <ogc:Literal>MultiLineString</ogc:Literal>
       </ogc:Function>
       <ogc:Literal>true</ogc:Literal>
     </ogc:PropertyIsEqualTo>
   </ogc:Filter>
   <LineSymbolizer>
     ...
   </LineSymbolizer>
</Rule>
```

This filter is read as `geometryType(geom) in ("LineString", "LinearRing", "MultiLineString")`. Filter functions in Filter 1.0 have a fixed number of arguments, so there is a series of `in` functions whose names correspond to the number of arguments they accept: `in2`, `in3`, ..., `in10`.

dimension function

A slightly simpler alternative is to use the geometry `dimension` function to select geometries of a desired dimension. Dimension 0 selects Points and MultiPoints, dimension 1 selects LineStrings, LinearRings and MultiLineStrings, and dimension 2 selects Polygons and MultiPolygons. The following example shows how to select linear geometries:

```
<Rule>
   <ogc:PropertyIsEqualTo>
      <ogc:Function name="dimension">
         <ogc:PropertyName>geom</ogc:PropertyName>
      </ogc:Function>
      <ogc:Literal>1</ogc:Literal>
   </ogc:PropertyIsEqualTo>
   <LineSymbolizer>
     ...
   </LineSymbolizer>
</Rule>
```

12.6.2 Styling using Transformation Functions

The Symbology Encoding 1.1 specification defines the following **transformation functions**:

- `Recode` transforms a set of discrete attribute values into another set of values
- `Categorize` transforms a continuous-valued attribute into a set of discrete values
- `Interpolate` transforms a continuous-valued attribute into another continuous range of values

These functions provide a concise way to compute styling parameters from feature attribute values. Geoserver implements them as *Filter functions* with the same names.

Note: The GeoServer function syntax is slightly different to the SE 1.1 definition, since the specification defines extra syntax elements which are not available in GeoServer functions.

These functions can make style documents more concise, since they express logic which would otherwise require many separate rules or complex Filter expressions, They even allow logic which is impossible to express any other way. A further advantage is that they often provide superior performance to explicit rules.

One disadvantage of using these functions for styling is that they are not displayed in WMS legend graphics.

Recode

The `Recode` filter function transforms a set of discrete values for an attribute into another set of values. The function can be used within SLD styling parameters to convert the value of a feature attribute into specific values for a parameter such as color, size, width, opacity, etc.

The recoding is defined by a set of *(input, output)* value pairs.

Example

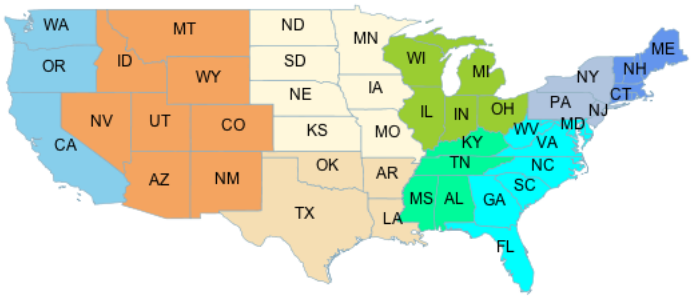
Consider a chloropleth map of the US states dataset using the fill color to indicate the topographic regions for the states. The dataset has an attribute `SUB_REGION` containing the region code for each state. The `Recode` function is used to map each region code into a different color.

The symbolizer for this style is:

```
<PolygonSymbolizer>
   <Fill>
     <CssParameter name="fill">
       <ogc:Function name="Recode">
         <!-- Value to transform -->
         <ogc:Function name="strTrim">
           <ogc:PropertyName>SUB_REGION</ogc:PropertyName>
         </ogc:Function>

         <!-- Map of input to output values -->
         <ogc:Literal>N Eng</ogc:Literal>
         <ogc:Literal>#6495ED</ogc:Literal>

         <ogc:Literal>Mid Atl</ogc:Literal>
         <ogc:Literal>#B0C4DE</ogc:Literal>

         <ogc:Literal>S Atl</ogc:Literal>
         <ogc:Literal>#00FFFF</ogc:Literal>

         <ogc:Literal>E N Cen</ogc:Literal>
         <ogc:Literal>#9ACD32</ogc:Literal>

         <ogc:Literal>E S Cen</ogc:Literal>
         <ogc:Literal>#00FA9A</ogc:Literal>

         <ogc:Literal>W N Cen</ogc:Literal>
         <ogc:Literal>#FFF8DC</ogc:Literal>
```

```
          <ogc:Literal>W S Cen</ogc:Literal>
          <ogc:Literal>#F5DEB3</ogc:Literal>

          <ogc:Literal>Mtn</ogc:Literal>
          <ogc:Literal>#F4A460</ogc:Literal>

          <ogc:Literal>Pacific</ogc:Literal>
          <ogc:Literal>#87CEEB</ogc:Literal>
        </ogc:Function>
      </CssParameter>
    </Fill>
</PolygonSymbolizer>
```

This style produces the following output:



Categorize

The `Categorize` filter function transforms a continuous-valued attribute into a set of discrete values. The function can be used within SLD styling parameters to convert the value of a feature attribute into specific values for a parameter such as color, size, width, opacity, etc.

The categorization is defined by a list of alternating output values and data thresholds. The threshold values define the breaks between the input ranges. Inputs are converted into output values depending on which range they fall in.

Example

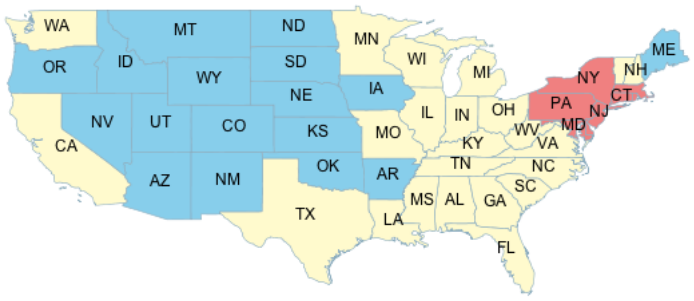
Consider a chloropleth map of the US states dataset using the fill color to indicate a categorization of the states by population. The dataset has attributes `PERSONS` and `LAND_KM` from which the population density is computed using the `Div` operator. This value is input to the `Categorize` function, which is used to assign different colors to the density ranges [<= 20], [20 - 100], and [> 100].

The symbolizer for this style is:

```
<PolygonSymbolizer>
   <Fill>
     <CssParameter name="fill">
       <ogc:Function name="Categorize">
         <!-- Value to transform -->
         <ogc:Div>
           <ogc:PropertyName>PERSONS</ogc:PropertyName>
           <ogc:PropertyName>LAND_KM</ogc:PropertyName>
```

```
          </ogc:Div>

          <!-- Output values and thresholds -->
          <ogc:Literal>#87CEEB</ogc:Literal>
          <ogc:Literal>20</ogc:Literal>
          <ogc:Literal>#FFFACD</ogc:Literal>
          <ogc:Literal>100</ogc:Literal>
          <ogc:Literal>#F08080</ogc:Literal>

        </ogc:Function>
      </CssParameter>
    </Fill>
</PolygonSymbolizer>
```

This style produces the following output:



Interpolate

The `Interpolate` filter function transforms a continuous-valued attribute into another continuous range of values. The function can be used within SLD styling parameters to convert the value of a feature attribute into a continuous-valued parameter such as color, size, width, opacity, etc.

The transformation is defined by a set of *(input, output)* control points chosen along a desired mapping curve. Piecewise interpolation along the curve is used to compute an output value for any input value.

The function is able to compute either numeric or color values as output. This is known as the **interpolation method**, and is specified by an optional parameter with a value of `numeric` (the default) or `color`.

The *shape* of the mapping curve between control points is specified by the **interpolation mode**, which is an optional parameter with values of `linear` (the default), `cubic`, or `cosine`.

Example

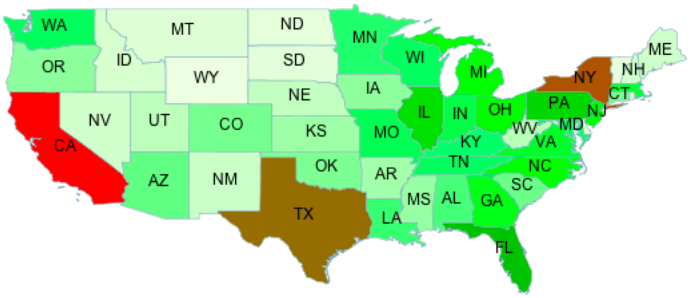
Interpolating over color ranges allows concise definition of continuously-varying colors for chloropleth (thematic) maps. As an example, consider a map of the US states dataset using the fill color to indicate the population of the states. The dataset has an attribute `PERSONS` containing the population of each state. The population values lie in the range 0 to around 30,000,000. The interpolation curve is defined by three control points which assign colors to the population levels 0, 9,000,000 and 23,000,000. The colors for population values are computed by piecewise linear interpolation along this curve. For example, a state with a population of 16,000,000 is displayed with a color midway between the ones for the middle and upper control points. States with populations greater than 23,000,000 are displayed with the last color.

Because the interpolation is being performed over color values, the method parameter is supplied, with a value of `color`. Since the default linear interpolation is used, no interpolation mode is supplied,

The symbolizer for this style is:

```
<PolygonSymbolizer>
   <Fill>
     <CssParameter name="fill">
       <ogc:Function name="Interpolate">
         <!-- Property to transform -->
         <ogc:PropertyName>PERSONS</ogc:PropertyName>

         <!-- Mapping curve definition pairs (input, output) -->
         <ogc:Literal>0</ogc:Literal>
         <ogc:Literal>#fefeee</ogc:Literal>

         <ogc:Literal>9000000</ogc:Literal>
         <ogc:Literal>#00ff00</ogc:Literal>

         <ogc:Literal>23000000</ogc:Literal>
         <ogc:Literal>#ff0000</ogc:Literal>

         <!-- Interpolation method -->
         <ogc:Literal>color</ogc:Literal>

         <!-- Interpolation mode - defaults to linear -->
       </ogc:Function>
     </CssParameter>
   </Fill>
</PolygonSymbolizer>
```

This symbolizer produces the following output:



CHAPTER 13

Services

GeoServer serves data using standard protocols established by the Open Geospatial Consortium:

- The **Web Feature Service** (WFS) supports requests for geographical feature data (with vector geometry and attributes).
- The **Web Map Service** (WMS) supports requests for map images (and other formats) generated from geographical data.
- The **Web Coverage Service** (WCS) supports requests for coverage data (rasters).

These services are the primary way that GeoServer supplies geospatial information.

13.1 Web Feature Service

This section describes the Web Feature Service.

13.1.1 WFS basics

GeoServer provides support for the Open Geospatial Consortium (OGC) Web Feature Service (WFS) specification, versions **1.0.0**, **1.1.0**, and **2.0.0**. WFS defines a standard for exchanging vector data over the Internet. With a compliant WFS, clients can query both the data structure and the source data. Advanced WFS operations also support feature locking and edit operations.

GeoServer is the reference implementation of all three versions of the standard, completely implementing every part of the protocol. This includes the basic operations of *GetCapabilities*, *DescribeFeatureType*, and *GetFeature*, as well as more advanced options such as *Transaction*. GeoServer WFS is also integrated with its *Security* system to limit access to data and transactions, and supports a variety of *WFS output formats*, making the raw data more widely available.

Differences between WFS versions

The major differences between the WFS versions are:

- WFS 1.1.0 and 2.0.0 return GML3 as the default GML, whereas in WFS 1.0.0, the default is GML2. GML3 adopts marginally different ways of specifying a geometry. GeoServer supports requests in both GML3 and GML2 formats.
- In WFS 1.1.0 and 2.0.0, the SRS (Spatial Reference System, or projection) is specified with `urn:x-ogc:def:crs:EPSG:XXXX`, whereas in WFS 1.0.0 the specification was

`http://www.opengis.net/gml/srs/epsg.xml#XXXX`. This change has implications for the *axis order* of the returned data.

- WFS 1.1.0 and 2.0.0 support on-the-fly reprojection of data, which supports returning the data in a SRS other than the native SRS.
- WFS 2.0.0 introduces a new version of the filter encoding specification, adding support for temporal filters.
- WFS 2.0.0 supports joins via a GetFeature request.
- WFS 2.0.0 adds the ability to page results of a GetFeature request via the `startIndex` and `count` parameters. GeoServer now supports this functionality in WFS 1.0.0 and 1.1.0.
- WFS 2.0.0 supports stored queries, which are regular WFS queries stored on the server such that they may be invoked by passing the appropriate identifier with a WFS request.
- WFS 2.0.0 supports SOAP (Simple Object Access Protocol) as an alternative to the OGC interface.

Note: There are also two changes to parameter names which can cause confusion. WFS 2.0.0 uses the `count` parameter to limit the number of features returned rather than the `maxFeatures` parameter used in previous versions. It also changed `typeName` to `typeNames` although GeoServer will accept either.

Axis ordering

WFS 1.0.0 servers return geographic coordinates in longitude/latitude (x/y) order, the most common way to distribute data. For example, most shapefiles adopt this order by default.

However, the traditional axis order for geographic and cartographic systems is the opposite—latitude/longitude (y/x)—and the later WFS specifications respect this. The default axis ordering support is:

- Latitude/longitude—WFS 1.1.0 and WFS 2.0.0
- Longitude/latitude—WFS 1.0.0

This may cause difficulties when switching between servers with different WFS versions, or when upgrading your WFS. To minimize confusion and increase interoperability, GeoServer has adopted the following assumptions when specifying projections in the following formats:

Representation	Assumed axis order
`EPSG:xxxx`	longitude/latitude (x/y)
`http://www.opengis.net/gml/srs/epsg.xml#xxxx`	longitude/latitude (x/y)
`urn:x-ogc:def:crs:EPSG:xxxx`	latitude/longitude (y/x)

13.1.2 WFS reference

The Web Feature Service (WFS) is a standard created by the Open Geospatial Consortium (OGC) for creating, modifying and exchanging vector format geographic information on the Internet using HTTP. A WFS encodes and transfers information in Geography Markup Language (GML), a subset of XML.

The current version of WFS is **2.0.0**. GeoServer supports versions 2.0.0, 1.1.0, and 1.0.0. Although there are some important differences between the versions, the request syntax often remains the same.

A related OGC specification, the *Web Map Service*, defines the standard for exchanging geographic information in digital image format.

Benefits of WFS

The WFS standard defines the framework for providing access to, and supporting transactions on, discrete geographic features in a manner that is independent of the underlying data source. Through a combination of discovery, query, locking, and transaction operations, users have access to the source spatial and attribute data in a manner that allows them to interrogate, style, edit (create, update, and delete), and download individual features. The transactional capabilities of WFS also support the development and deployment of collaborative mapping applications.

Operations

All versions of WFS support the these operations:

Operation	Description
`GetCapabilities`	Generates a metadata document describing a WFS service provided by server as well as valid WFS operations and parameters
`DescribeFeatureType`	Returns a description of feature types supported by a WFS service
`GetFeature`	Returns a selection of features from a data source including geometry and attribute values
`LockFeature`	Prevents a feature from being edited through a persistent feature lock
`Transaction`	Edits existing feature types by creating, updating, and deleting

The following operations are available in **version 2.0.0 only**:

Operation	Description
`GetPropertyValue`	Retrieves the value of a feature property or part of the value of a complex feature property from the data store for a set of features identified using a query expression
`GetFeatureWithLock`	Returns a selection of features and also applies a lock on those features
`CreateStoredQuery`	Create a stored query on the WFS server
`DropStoredQuery`	Deletes a stored query from the WFS server
`ListStoredQueries`	Returns a list of the stored queries on a WFS server
`DescribeStoredQueries`	Returns a metadata document describing the stored queries on a WFS server

The following operations are available in **version 1.1.0 only**:

Operation	Description
`GetGMLObject`	Retrieves features and elements by ID from a WFS

Note: In the examples that follow, the fictional URL `http://example.com/geoserver/wfs` is used for illustration. To test the examples, substitute the address of a valid WFS. Also, although the request would normally be defined on one line with no breaks, breaks are added for clarity in the examples provided.

Exceptions

WFS also supports a number of formats for reporting exceptions. The supported values for exception reporting are:

Format	Syntax	Description
XML	`exceptions=text/xml`	*(default)* XML output
JSON	`exceptions=application/json`	Simple JSON
JSONP	`exceptions=text/javascript`	Return a JSONP in the form: parseResponse(...jsonp...). See *WMS vendor parameters* to change the callback name. Note that this format is disabled by default (See *Global variables affecting WMS*).

GetCapabilities

The **GetCapabilities** operation is a request to a WFS server for a list of the operations and services, or *capabilities*, supported by that server.

To issue a GET request using HTTP:

```
http://example.com/geoserver/wfs?
  service=wfs&
  version=1.1.0&
  request=GetCapabilities
```

The equivalent request using POST:

```
<GetCapabilities
 service="WFS"
 xmlns="http://www.opengis.net/wfs"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.opengis.net/wfs
 http://schemas.opengis.net/wfs/1.1.0/wfs.xsd"/>
```

GET requests are simplest to decode, but the POST requests are equivalent.

The parameters for GetCapabilities are:

Parameter	Required?	Description
`service`	Yes	Service name—Value is `WFS`
`version`	Yes	Service version—Value is the current version number. The full version number must be supplied (“1.1.0”, “1.0.0”), not the abbreviated form (“1” or “1.1”).
`request`	Yes	Operation name—Value is `GetCapabilities`

Although all of the above parameters are technically required as per the specification, GeoServer will provide default values if any parameters are omitted from a request.

The GetCapabilities response is a lengthy XML document, the format of which is different for each of the supported versions. There are five main components in a GetCapabilities document:

Component	Description
`ServiceIdentification`	Contains basic header information for the request such as the `Title` and `ServiceType`. The `ServiceType` indicates which version(s) of WFS are supported.
`ServiceProvider`	Provides contact information about the company publishing the WFS service, including telephone, website, and email.
`OperationsMetadata`	Describes the operations that the WFS server supports and the parameters for each operation. A WFS server may be configured not to respond to the operations listed above.
`FeatureTypeList`	Lists the feature types published by a WFS server. Feature types are listed in the form `namespace:featuretype`. The default projection of the feature type is also listed, along with the bounding box for the data in the stated projection.
`Filter_Capabilities`	Lists the filters, or expressions, that are available to form query predicates, for example, `SpatialOperators` (such as `Equals`, `Touches`) and `ComparisonOperators` (such as `LessThan`, `GreaterThan`). The filters themselves are not included in the GetCapabilities document.

DescribeFeatureType

DescribeFeatureType requests information about an individual feature type before requesting the actual data. Specifically, the operation will request a list of features and attributes for the given feature type, or list the feature types available.

The parameters for DescribeFeatureType are:

Parameter	Required?	Description
`service`	Yes	Service name—Value is `WFS`
`version`	Yes	Service version—Value is the current version number
`request`	Yes	Operation name—Value is `DescribeFeatureType`
`typeNames`	Yes	Name of the feature type to describe (`typeName` for WFS 1.1.0 and earlier)
`exceptions`	No	Format for reporting exceptions—default value is `application/vnd.ogc.se_xml`
`outputFormat`	No	Defines the scheme description language used to describe feature types

To return a list of feature types, the GET request would be as follows. This request will return the list of feature types, sorted by namespace:

```
http://example.com/geoserver/wfs?
  service=wfs&
  version=2.0.0&
  request=DescribeFeatureType
```

To list information about a specific feature type called `namespace:featuretype`, the GET request would be:

```
http://example.com/geoserver/wfs?
  service=wfs&
  version=2.0.0&
  request=DescribeFeatureType&
  typeNames=namespace:featuretype
```

GetFeature

The **GetFeature** operation returns a selection of features from the data source.

This request will execute a GetFeature request for a given layer `namespace:featuretype`:

```
http://example.com/geoserver/wfs?
  service=wfs&
  version=2.0.0&
  request=GetFeature&
  typeNames=namespace:featuretype
```

Executing this command will return the geometries for all features in given a feature type, potentially a large amount of data. To limit the output you can restrict the GetFeature request to a single feature by including an additional parameter, `featureID` and providing the ID of a specific feature. In this case, the GET request would be:

```
http://example.com/geoserver/wfs?
  service=wfs&
  version=2.0.0&
  request=GetFeature&
  typeNames=namespace:featuretype&
  featureID=feature
```

If the ID of the feature is unknown but you still want to limit the amount of features returned, use the `count` parameter for WFS 2.0.0 or the `maxFeatures` parameter for earlier WFS versions. In the examples below, `N` represents the number of features to return:

```
http://example.com/geoserver/wfs?
  service=wfs&
  version=2.0.0&
  request=GetFeature&
  typeNames=namespace:featuretype&
  count=N

http://example.com/geoserver/wfs?
  service=wfs&
  version=1.1.0&
  request=GetFeature&
  typeName=namespace:featuretype&
  maxFeatures=N
```

Exactly which N features will be returned depends in the internal structure of the data. However, you can sort the returned selection based on an attribute value. In the following example, an attribute is included in the request using the `sortBy=attribute` parameter (replace `attribute` with the attribute you wish to sort by):

```
http://example.com/geoserver/wfs?
   service=wfs&
   version=2.0.0&
   request=GetFeature&
   typeNames=namespace:featuretype&
   count=N&
   sortBy=attribute
```

The default sort operation is to sort in ascending order. Some WFS servers require the sort order to be specified, even if an ascending order sort if required. In this case, append a `+A` to the request. Conversely, add a `+D` to the request to sort in descending order as follows:

```
http://example.com/geoserver/wfs?
  service=wfs&
  version=2.0.0&
  request=GetFeature&
  typeNames=namespace:featuretype&
  count=N&
  sortBy=attribute+D
```

There is no obligation to use `sortBy` with `count` in a GetFeature request, but they can be used together to manage the returned selection of features more effectively.

To restrict a GetFeature request by attribute rather than feature, use the `propertyName` key in the form `propertyName=attribute`. You can specify a single attribute, or multiple attributes separated by commas. To search for a single attribute in all features, the following request would be required:

```
http://example.com/geoserver/wfs?
  service=wfs&
  version=2.0.0&
  request=GetFeature&
  typeNames=namespace:featuretype&
  propertyName=attribute
```

For a single property from just one feature, use both `featureID` and `propertyName`:

```
http://example.com/geoserver/wfs?
  service=wfs&
  version=2.0.0&
  request=GetFeature&
  typeNames=namespace:featuretype&
  featureID=feature&
  propertyName=attribute
```

For more than one property from a single feature, use a comma-seaprated list of values for `propertyName`:

```
http://example.com/geoserver/wfs?
  service=wfs&
  version=2.0.0&
  request=GetFeature&
  typeNames=namespace:featuretype&
  featureID=feature&
  propertyName=attribute1,attribute2
```

While the above permutations for a GetFeature request focused on non-spatial parameters, it is also possible to query for features based on geometry. While there are limited options available in a GET request for spatial queries (more are available in POST requests using filters), filtering by bounding box (BBOX) is supported.

The BBOX parameter allows you to search for features that are contained (or partially contained) inside a box of user-defined coordinates. The format of the BBOX parameter is `bbox=a1,b1,a2,b2``where ``a1`, `b1`, `a2`, and `b2` represent the coordinate values. The order of coordinates passed to the BBOX parameter depends on the coordinate system used. (This is why the coordinate syntax isn't represented with `x` or `y`.) To specify the coordinate system, append `srsName=CRS` to the WFS request, where `CRS` is the Coordinate Reference System you wish to use.

As for which corners of the bounding box to specify, the only requirement is for a bottom corner (left or right) to be provided first. For example, bottom left and top right, or bottom right and top left.

An example request involving returning features based on bounding box would be in the following format:

```
http://example.com/geoserver/wfs?
  service=wfs&
  version=2.0.0&
  request=GetFeature&
  typeNames=namespace:featuretype&
  srsName=CRS
  bbox=a1,b1,a2,b2
```

LockFeature

A **LockFeature** operation provides a long-term feature locking mechanism to ensure consistency in edit transactions. If one client fetches a feature and makes some changes before submitting it back to the WFS, locks prevent other clients from making any changes to the same feature, ensuring a transaction that can be serialized. If a WFS server supports this operation, it will be reported in the server's GetCapabilities response.

In practice, few clients support this operation.

Transaction

The **Transaction** operation can create, modify, and delete features published by a WFS. Each transaction will consist of zero or more Insert, Update, and Delete elements, with each transaction element performed in order. Every GeoServer transaction is *atomic*, meaning that if any of the elements fail, the transaction is abandoned and the data is unaltered. A WFS server that supports **transactions** is sometimes known as a WFS-T server. **GeoServer fully supports transactions.**

More information on the syntax of transactions can be found in the WFS specification and in the *GeoServer sample requests*.

GetGMLObject

Note: This operation is valid for **WFS version 1.1.0 only**.

A **GetGMLObject** operation accepts the identifier of a GML object (feature or geometry) and returns that object. This operation is relevant only in situations that require *Complex Features* by allowing clients to extract just a portion of the nested properties of a complex feature. As a result, this operation is not widely used by client applications.

GetPropertyValue

Note: This operation is valid for **WFS version 2.0.0 only**.

A **GetPropertyValue** operation retrieves the value of a feature property, or part of the value of a complex feature property, from a data source for a given set of features identified by a query.

This example retrieves the geographic content only of the features in the `topp:states` layer:

```
http://example.com/geoserver/wfs?
  service=wfs&
  version=2.0.0&
  request=GetPropertyValue&
  typeNames=topp:states&
  valueReference=the_geom
```

The same example in a POST request:

```
<wfs:GetPropertyValue service='WFS' version='2.0.0'
 xmlns:topp='http://www.openplans.org/topp'
 xmlns:fes='http://www.opengis.net/fes/2.0'
 xmlns:wfs='http://www.opengis.net/wfs/2.0'
 valueReference='the_geom'>
  <wfs:Query typeNames='topp:states'/>
</wfs:GetPropertyValue>
```

To retrieve value for a different attribute, alter the `valueReference` parameter.

GetFeatureWithLock

Note: This operation is valid for **WFS version 2.0.0 only**.

A **GetFeatureWithLock** operation is similar to a **GetFeature** operation, except that when the set of features are returned from the WFS server, the features are also locked in anticipation of a subsequent transaction operation.

This POST example retrieves the features of the `topp:states` layer, but in addition locks those features for five minutes.

```
<wfs:GetFeatureWithLock service='WFS' version='2.0.0'
 handle='GetFeatureWithLock-tc1' expiry='5' resultType='results'
 xmlns:topp='http://www.openplans.org/topp'
 xmlns:fes='http://www.opengis.net/fes/2.0'
 xmlns:wfs='http://www.opengis.net/wfs/2.0'
 valueReference='the_geom'>
  <wfs:Query typeNames='topp:states'/>
</wfs:GetFeatureWithLock>
```

To adjust the lock time, alter the `expiry` parameter.

CreateStoredQuery

Note: This operation is valid for **WFS version 2.0.0 only**.

A **CreateStoredQuery** operation creates a stored query on the WFS server. The definition of the stored query is encoded in the `StoredQueryDefinition` parameter and is given an ID for a reference.

This POST example creates a new stored query (called "myStoredQuery") that filters the `topp:states` layer to those features that are within a given area of interest (`${AreaOfInterest}`):

```
<wfs:CreateStoredQuery service='WFS' version='2.0.0'
 xmlns:wfs='http://www.opengis.net/wfs/2.0'
 xmlns:fes='http://www.opengis.org/fes/2.0'
 xmlns:gml='http://www.opengis.net/gml/3.2'
 xmlns:myns='http://www.someserver.com/myns'
 xmlns:topp='http://www.openplans.org/topp'>
  <wfs:StoredQueryDefinition id='myStoredQuery'>
    <wfs:Parameter name='AreaOfInterest' type='gml:Polygon'/>
    <wfs:QueryExpressionText
     returnFeatureTypes='topp:states'
     language='urn:ogc:def:queryLanguage:OGC-WFS::WFS_QueryExpression'
     isPrivate='false'>
      <wfs:Query typeNames='topp:states'>
        <fes:Filter>
          <fes:Within>
            <fes:ValueReference>the_geom</fes:ValueReference>
             ${AreaOfInterest}
          </fes:Within>
        </fes:Filter>
      </wfs:Query>
    </wfs:QueryExpressionText>
  </wfs:StoredQueryDefinition>
</wfs:CreateStoredQuery>
```

DropStoredQuery

Note: This operation is valid for **WFS version 2.0.0 only**.

A **DropStoredQuery** operation drops a stored query previous created by a CreateStoredQuery operation. The request accepts the ID of the query to drop.

This example will drop a stored query with an ID of `myStoredQuery`:

```
http://example.com/geoserver/wfs?
  request=DropStoredQuery&
  storedQuery_Id=myStoredQuery
```

The same example in a POST request:

```
<wfs:DropStoredQuery
 xmlns:wfs='http://www.opengis.net/wfs/2.0'
 service='WFS' id='myStoredQuery'/>
```

ListStoredQueries

Note: This operation is valid for **WFS version 2.0.0 only**.

A **ListStoredQueries** operation returns a list of the stored queries currently maintained by the WFS server.

This example lists all stored queries on the server:

```
http://example.com/geoserver/wfs?
  request=ListStoredQueries&
  service=wfs&
  version=2.0.0
```

The same example in a POST request:

```
<wfs:ListStoredQueries service='WFS'
 version='2.0.0'
 xmlns:wfs='http://www.opengis.net/wfs/2.0'/>
```

DescribeStoredQueries

Note: This operation is valid for **WFS version 2.0.0 only**.

A **DescribeStoredQuery** operation returns detailed metadata about each stored query maintained by the WFS server. A description of an individual query may be requested by providing the ID of the specific query. If no ID is provided, all queries are described.

This example describes the exsting stored query with an ID of `urn:ogc:def:query:OGC-WFS::GetFeatureById`:

```
http://example.com/geoserver/wfs?
  request=DescribeStoredQueries&
  storedQuery_Id=urn:ogc:def:query:OGC-WFS::GetFeatureById
```

The same example in a POST request:

```
<wfs:DescribeStoredQueries
 xmlns:wfs='http://www.opengis.net/wfs/2.0'
 service='WFS'>
  <wfs:StoredQueryId>urn:ogc:def:query:OGC-WFS::GetFeatureById</wfs:StoredQueryId>
</wfs:DescribeStoredQueries>
```

13.1.3 WFS output formats

WFS returns features and feature information in a number of formats. The syntax for specifying an output format is:

```
outputFormat=<format>
```

where `<format>` is one of the following options:

Format	Syntax	Notes
GML2	`outputFormat=GML2`	Default option for WFS 1.0.0
GML3	`outputFormat=GML3`	Default option for WFS 1.1.0 and 2.0.0
Shapefile	`outputFormat=shape-zip`	ZIP archive will be generated containing the shapefile (see *Shapefile output customization* below)
JSON	`outputFormat=application/json`	Returns a GeoJSON or a JSON output. Note `outputFormat=json` is only supported for getFeature (for backward compatibility).
JSONP	`outputFormat=text/javascript`	Returns a JSONP in the form: `parseResponse(...json...)`. See *WMS vendor parameters* to change the callback name. Note that this format is disabled by default (See *Global variables affecting WMS*).
CSV	`outputFormat=csv`	Returns a CSV (comma-separated values) file

Note: Some additional output formats (such as *Excel*) are available with the use of an extension. The full list of output formats supported by a particular GeoServer instance can be found by performing a WFS *GetCapabilities* request.

Shapefile output customization

The shapefile output format output can be customized by preparing a *Freemarker template* which will configure the file name of the archive (ZIP file) and the files it contains. The default template is:

```
zip=${typename}
shp=${typename}${geometryType}
txt=wfsrequest
```

The `zip` property is the name of the archive, the `shp` property is the name of the shapefile for a given feature type, and `txt` is the dump of the actual WFS request.

The properties available in the template are:

- `typename`—Feature type name (for the `zip` property this will be the first feature type if the request contains many feature types)
- `geometryType`—Type of geometry contained in the shapefile. This is only used if the output geometry type is generic and the various geometries are stored in one shapefile per type.
- `workspace`—Workspace of the feature type
- `timestamp`—Date object with the request timestamp
- `iso_timestamp`—String (ISO timestamp of the request at GMT) in `yyyyMMdd_HHmmss` format

Format options as parameter in WFS requests

GeoServer provides the `format_options` vendor-specific parameter to specify parameters that are specific to each format. The syntax is:

```
format-options=param1:value1;param2:value2;...
```

The currently supported format option in WFS output is:

- `filename`—Applies only to the SHAPE-ZIP output format. If a file name is provided, the name is used as the output file name. For example, `format_options=filename:roads.zip`. If a file name is not specified, the output file name is inferred from the requested feature type name.

13.1.4 WFS vendor parameters

WFS vendor parameters are non-standard request parameters defined by an implementation to provide enhanced capabilities. GeoServer supports a variety of vendor-specific WFS parameters.

CQL filters

In WFS *GetFeature* GET requests, the `cql_filter` parameter can be used to specify a filter in ECQL (Extended Common Query Language) format. ECQL provides a more compact and readable syntax compared to OGC XML filters.

For full details see the *ECQL Reference* and *CQL and ECQL* tutorial.

The following example illustrates a GET request OGC filter:

```
filter=%3CFilter%20xmlns:gml=%22http://www.opengis.net/gml%22%3E%3CIntersects%3E%3CPropertyName%3Ethe
```

Using ECQL, the identical filter would be defined as follows:

```
cql_filter=INTERSECT(the_geom,%20POINT%20(-74.817265%2040.5296504))
```

Format options

The `format_options` parameter is a container for other parameters that are format-specific. The syntax is:

```
format_options=param1:value1;param2:value2;...
```

The supported format option is:

- `callback` (default is `parseResponse`)—Specifies the callback function name for the JSONP response format

Reprojection

As WFS 1.1.0 and 2.0.0 both support data reprojection, GeoServer can store the data in one projection and return GML in another projection. While not part of the specification, GeoServer supports this using WFS 1.0.0 as well. When submitting a WFS *GetFeature* GET request, you can add this parameter to specify the reprojection SRS as follows:

```
srsName=<srsName>
```

The code for the projection is represented by `<srsName>`, for example `EPSG:4326`. For POST requests, you can add the same code to the `Query` element.

XML request validation

GeoServer is less strict than the WFS specification when it comes to the validity of an XML request. To force incoming XML requests to be valid, use the following parameter:

```
strict=[true|false]
```

The default option for this parameter is `false`.

For example, the following request is invalid:

```
<wfs:GetFeature service="WFS" version="1.0.0"
 xmlns:wfs="http://www.opengis.net/wfs">
  <Query typeName="topp:states"/>
</wfs:GetFeature>
```

The request is invalid for two reasons:

- The `Query` element should be prefixed with `wfs:`.
- The namespace prefix has not been mapped to a namespace URI.

That said, the request would still be processed by default. Executing the above command with the `strict=true` parameter, however, would result in an error. The correct syntax should be:

```
<wfs:GetFeature service="WFS" version="1.0.0"
 xmlns:wfs="http://www.opengis.net/wfs"
 xmlns:topp="http://www.openplans.org/topp">
  <wfs:Query typeName="topp:states"/>
</wfs:GetFeature>
```

GetCapabilities namespace filter

WFS *GetCapabilities* requests may be filtered to return only those layers that correspond to a particular namespace by adding the `<namespace>` parameter to the request.

Note: This parameter only affects GetCapabilities requests.

To apply this filter, add the following code to your request:

```
namespace=<namespace>
```

Although providing an invalid namespace will not result in any errors, the GetCapabilities document returned will not contain any layer information.

Warning: Using this parameter may result your GetCapabilities document becoming invalid, as the WFS specification requires the document to return at least one layer.

Note: This filter is related to *Virtual OWS Services*.

13.1.5 WFS schema mapping

One of the functions of the GeoServer WFS is to automatically map the internal schema of a dataset to a feature type schema. This mapping is performed according to the following rules:

- The name of the feature element maps to the name of the dataset.

- The name of the feature type maps to the name of the dataset with the string "Type" appended to it.
- The name of each attribute of the dataset maps to the name of an element particle contained in the feature type.
- The type of each attribute of the dataset maps to the appropriate XML schema type (`xsd:int`, `xsd:double`, and so on).

For example, a dataset has the following schema:

```
myDataset(intProperty:Integer, stringProperty:String, floatProperty:Float, geometry:Point)
```

This schema would be mapped to the following XML schema, available via a `DescribeFeatureType` request for the `topp:myDataset` type:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:gml="http://www.opengis.net/gml"
 xmlns:topp="http://www.openplans.org/topp"
 targetNamespace="http://www.openplans.org/topp"
 elementFormDefault="qualified">

  <xsd:import namespace="http://www.opengis.net/gml"
   schemaLocation="http://localhost:8080/geoserver/schemas/gml/3.1.1/base/gml.xsd"/>

  <xsd:complexType name="myDatasetType">
    <xsd:complexContent>
      <xsd:extension base="gml:AbstractFeatureType">
        <xsd:sequence>
          <xsd:element maxOccurs="1" minOccurs="0" name="intProperty" nillable="true" type="xsd:int"/
          <xsd:element maxOccurs="1" minOccurs="0" name="stringProperty" nillable="true" type="xsd:st
          <xsd:element maxOccurs="1" minOccurs="0" name="floatProperty" nillable="true" type="xsd:dou
          <xsd:element maxOccurs="1" minOccurs="0" name="geometry" nillable="true" type="gml:PointPro
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="myDataset" substitutionGroup="gml:_Feature" type="topp:myDatasetType"/>

</xsd:schema>
```

Schema customization

The GeoServer WFS supports a limited amount of schema output customization. A custom schema may be useful for the following:

- Limiting the attributes which are exposed in the feature type schema
- *Changing* the types of attributes in the schema
- Changing the structure of the schema (for example, changing the base feature type)

For example, it may be useful to limit the exposed attributes in the example dataset described above. Start by retrieving the default output as a benchmark of the complete schema. With the feature type schema listed above, the `GetFeature` request would be as follows:

```
<topp:myDataset gml:id="myDataset.1">
 <topp:intProperty>1</topp:intProperty>
  <topp:stringProperty>one</topp:stringProperty>
  <topp:floatProperty>1.1</topp:floatProperty>
```

```
  <topp:geometry>
    <gml:Point srsName="urn:x-ogc:def:crs:EPSG:4326">
      <gml:pos>1.0 1.0</gml:pos>
    </gml:Point>
  </topp:geometry>
</topp:myDataset>
```

To remove `floatProperty` from the list of attributes, the following steps would be required:

1. The original schema is modified to remove the `floatProperty`, resulting in the following type definition:

   ```
   <xsd:complexType name="myDatasetType">
     <xsd:complexContent>
       <xsd:extension base="gml:AbstractFeatureType">
         <xsd:sequence>
           <xsd:element maxOccurs="1" minOccurs="0" name="intProperty" nillable="true" type="xsd:in
           <xsd:element maxOccurs="1" minOccurs="0" name="stringProperty" nillable="true" type="xsd
           <!-- remove the floatProperty element
           <xsd:element maxOccurs="1" minOccurs="0" name="floatProperty" nillable="true" type="xsd:
           -->
           <xsd:element maxOccurs="1" minOccurs="0" name="geometry" nillable="true" type="gml:Point
         </xsd:sequence>
       </xsd:extension>
       </xsd:complexContent>
   </xsd:complexType>
   ```

2. The modification is saved in a file named `schema.xsd`.

3. The `schema.xsd` file is copied into the feature type directory for the `topp:myDataset` which is:

   ```
   $GEOSERVER_DATA_DIR/workspaces/<workspace>/<datastore>/myDataset/
   ```

   where `<workspace>` is the name of the workspace containing your data store and `<datastore>` is the name of the data store which contains `myDataset`

The modified schema will only be available to GeoServer when the configuration is reloaded or GeoServer is restarted.

A subsequent `DescribeFeatureType` request for `topp:myDataset` confirms the `floatProperty` element is absent:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:gml="http://www.opengis.net/gml"
 xmlns:topp="http://www.openplans.org/topp"
 targetNamespace="http://www.openplans.org/topp"
 elementFormDefault="qualified">

  <xsd:import namespace="http://www.opengis.net/gml"
   schemaLocation="http://localhost:8080/geoserver/schemas/gml/3.1.1/base/gml.xsd"/>

  <xsd:complexType name="myDatasetType">
    <xsd:complexContent>
      <xsd:extension base="gml:AbstractFeatureType">
        <xsd:sequence>
          <xsd:element maxOccurs="1" minOccurs="0" name="intProperty" nillable="true" type="xsd:int"
          <xsd:element maxOccurs="1" minOccurs="0" name="stringProperty" nillable="true" type="xsd:st
          <xsd:element maxOccurs="1" minOccurs="0" name="geometry" nillable="true" type="gml:PointPro
        </xsd:sequence>
      </xsd:extension>
```

```
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="myDataset" substitutionGroup="gml:_Feature" type="topp:myDatasetType"/>

</xsd:schema>
```

A `GetFeature` request will now return features that don't include the `floatProperty` attribute:

```
<topp:myDataset gml:id="myDataset.1">
  <topp:intProperty>1</topp:intProperty>
  <topp:stringProperty>one</topp:stringProperty>
  <topp:geometry>
    <gml:Point srsName="urn:x-ogc:def:crs:EPSG:4326">
      <gml:pos>1.0 1.0</gml:pos>
    </gml:Point>
  </topp:geometry>
</topp:myDataset>
```

Type changing

Schema customization may be used to perform some **type changing**, although this is limited by the fact that a changed type must be in the same *domain* as the original type. For example, integer types must be changed to integer types, temporal types to temporal types, and so on.

The most common change type requirement is for geometry attributes. In many cases the underlying data set does not have the necessary metadata to report the specific geometry type of a geometry attribute. The automatic schema mapping would result in an element definition similar to the following:

```
<xsd:element maxOccurs="1" minOccurs="0" name="geometry" nillable="true" type="gml:GeometryPropertyTy
```

However if the specific type of the geometry is known, the element definition above could be altered. For point geometry, the element definition could be altered to :

```
<xsd:element maxOccurs="1" minOccurs="0" name="geometry" nillable="true" type="gml:PointPropertyType"
```

13.2 Web Map Service

13.2.1 WMS basics

GeoServer provides support for Open Geospatial Consortium (OGC) **Web Map Service (WMS)** versions 1.1.1 and 1.3.0. This is the most widely used standard for generating maps on the web, and it is the primary interface to request map products from GeoServer. Using WMS makes it possible for clients to overlay maps from several different sources in a seamless way.

GeoServer's WMS implementation fully supports the standard, and is certified compliant against the OGC's test suite. It includes a wide variety of rendering and labeling options, and is one of the fastest WMS Servers for both raster and vector data.

GeoServer WMS supports reprojection to any **coordinate reference system** in the EPSG database. It is possible to add additional coordinate systems if the Well Known Text definition is known. See *Coordinate Reference System Handling* for details.

GeoServer fully supports the **Styled Layer Descriptor (SLD)** standard, and uses SLD files as its native styling language. For more information on how to style data in GeoServer see the section *Styling*

Differences between WMS versions

The major differences between versions 1.1.1 and 1.3.0 are:

- In 1.1.1 geographic coordinate systems specified with the `EPSG` namespace are defined to have an axis ordering of longitude/latitude. In 1.3.0 the ordering is latitude/longitude. See *Axis Ordering* below for more details.
- In the GetMap operation the `srs` parameter is called `crs` in 1.3.0. GeoServer supports both keys regardless of version.
- In the GetFeatureInfo operation the `x` and `y` parameters are called `i` and `j` in 1.3.0. GeoServer supports both keys regardless of version, except when in CITE-compliance mode.

Axis Ordering

The WMS 1.3 specification mandates that the axis ordering for geographic coordinate systems defined in the EPSG database be *latitude/longitude*, or *y/x*. This is contrary to the fact that most spatial data is usually in *longitude/latitude*, or *x/y*. This requires that the coordinate order in the `BBOX` parameter be reversed for `SRS` values which are geographic coordinate systems.

For example, consider the WMS 1.1 request using the WGS84 SRS (EPSG:4326):

```
geoserver/wms?VERSION=1.1.1&REQUEST=GetMap&SRS=epsg:4326&BBOX=-180,-90.180,90&...
```

The equivalent WMS 1.3 request is:

```
geoserver/wms?VERSION=1.1.1&REQUEST=GetMap&CRS=epsg:4326&BBOX=-90,-180,90,180&...
```

Note that the coordinates specified in the `BBOX` parameter are reversed.

13.2.2 WMS reference

Introduction

The OGC Web Map Service (WMS) specification defines an HTTP interface for requesting georeferenced map images from a server. GeoServer supports WMS 1.1.1, the most widely used version of WMS, as well as WMS 1.3.0.

The relevant OGC WMS specifications are:

- OGC Web Map Service Implementation Specification, Version 1.1.1
- OGC Web Map Service Implementation Specification, Version 1.3.0

GeoServer also supports some extensions to the WMS specification made by the Styled Layer Descriptor (SLD) standard to control the styling of the map output. These are defined in:

- OpenGIS Styled Layer Descriptor Profile of the Web Map Service Implementation Specification, Version 1.1.0

Benefits of WMS

WMS provides a standard interface for requesting a geospatial map image. The benefit of this is that WMS clients can request images from multiple WMS servers, and then combine them into a single view for the user. The standard guarantees that these images can all be overlaid on one another as they actually would be in reality. Numerous servers and clients support WMS.

Operations

WMS requests can perform the following operations:

Operation	Description
`Exceptions`	If an exception occur
`GetCapabilities`	Retrieves metadata about the service, including supported operations and parameters, and a list of the available layers
`GetMap`	Retrieves a map image for a specified area and content
`GetFeatureInfo` (optional)	Retrieves the underlying data, including geometry and attribute values, for a pixel location on a map
`DescribeLayer` (optional)	Indicates the WFS or WCS to retrieve additional information about the layer.
`GetLegendGraphic` (optional)	Retrieves a generated legend for a map

Exceptions

Formats in which WMS can report exceptions. The supported values for exceptions are:

Format	Syntax	Notes
XML	`EXCEPTIONS=application/vnd.ogc.se_xml`	Xml output (The default format)
PNG	`EXCEPTIONS=application/vnd.ogc.se_inimage`	Generates an image
Blank	`EXCEPTIONS=application/vnd.ogc.se_blank`	Generates a blank image
JSON	`EXCEPTIONS=application/json`	Simple Json representation.
JSONP	`EXCEPTIONS=text/javascript`	Return a JsonP in the form: paddingOutput(...jsonp...). See *WMS vendor parameters* to change the callback name. Note that this format is disabled by default (See *Global variables affecting WMS*).

GetCapabilities

The **GetCapabilities** operation requests metadata about the operations, services, and data ("capabilities") that are offered by a WMS server.

The parameters for the GetCapabilities operation are:

Parameter	Required?	Description
`service`	Yes	Service name. Value is `WMS`.
`version`	Yes	Service version. Value is one of `1.0.0`, `1.1.0`, `1.1.1`, `1.3`.
`request`	Yes	Operation name. Value is `GetCapabilities`.

GeoServer provides the following vendor-specific parameters for the GetCapabilities operation. They are fully documented in the *WMS vendor parameters* section.

Parameter	Required?	Description
`namespace`	No	limits response to layers in a given namespace

A example GetCapabilities request is:

```
http://localhost:8080/geoserver/wms?
service=wms&
version=1.1.1&
request=GetCapabilities
```

There are three parameters being passed to the WMS server, `service=wms`, `version=1.1.1`, and `request=GetCapabilities`. The `service` parameter tells the WMS server that a WMS request is forthcoming. The `version` parameter refers to which version of WMS is being requested. The `request` parameter specifies the GetCapabilities operation. The WMS standard requires that requests always includes these three parameters. GeoServer relaxes these requirements (by setting the default version if omitted), but for standard-compliance they should always be specified.

The response is a Capabilities XML document that is a detailed description of the WMS service. It contains three main sections:

Service	Contains service metadata such as the service name, keywords, and contact information for the organization operating the server.
Request	Describes the operations the WMS service provides and the parameters and output formats for each operation. If desired GeoServer can be configured to disable support for certain WMS operations.
Layer	Lists the available coordinate systems and layers. In GeoServer layers are named in the form "namespace:layer". Each layer provides service metadata such as title, abstract and keywords.

GetMap

The **GetMap** operation requests that the server generate a map. The core parameters specify one or more layers and styles to appear on the map, a bounding box for the map extent, a target spatial reference system, and a width, height, and format for the output. The information needed to specify values for parameters such as `layers`, `styles` and `srs` can be obtained from the Capabilities document.

The response is a map image, or other map output artifact, depending on the format requested. GeoServer provides a wide variety of output formats, described in *WMS output formats*.

The standard parameters for the GetMap operation are:

Parameter	Required?	Description
service	Yes	Service name. Value is WMS.
version	Yes	Service version. Value is one of 1.0.0, 1.1.0, 1.1.1, 1.3.
request	Yes	Operation name. Value is GetMap.
layers	Yes	Layers to display on map. Value is a comma-separated list of layer names.
styles	Yes	Styles in which layers are to be rendered. Value is a comma-separated list of style names, or empty if default styling is required. Style names may be empty in the list, to use default layer styling.
srs *or* crs	Yes	Spatial Reference System for map output. Value is in form EPSG:nnn. crs is the parameter key used in WMS 1.3.0.
bbox	Yes	Bounding box for map extent. Value is minx,miny,maxx,maxy in units of the SRS.
width	Yes	Width of map output, in pixels.
height	Yes	Height of map output, in pixels.
format	Yes	Format for the map output. See *WMS output formats* for supported values.
transparent	No	Whether the map background should be transparent. Values are true or false. Default is false
bgcolor	No	Background color for the map image. Value is in the form RRGGBB. Default is FFFFFF (white).
exceptions	No	Format in which to report exceptions. Default value is application/vnd.ogc.se_xml.
time	No	Time value or range for map data. See *Time Support in Geoserver WMS* for more information.
sld	No	A URL referencing a *StyledLayerDescriptor* XML file which controls or enhances map layers and styling
sld_body	No	A URL-encoded *StyledLayerDescriptor* XML document which controls or enhances map layers and styling

GeoServer provides a number of useful vendor-specific parameters for the GetMap operation. These are documented in the *WMS vendor parameters* section.

Example WMS request for topp:states layer to be output as a PNG map image in SRS EPGS:4326 and using default styling is:

```
http://localhost:8080/geoserver/wms?
request=GetMap
&service=WMS
&version=1.1.1
&layers=topp%3Astates
&styles=population
&srs=EPSG%3A4326
&bbox=-145.15104058007,21.731919794922,-57.154894212888,58.961058642578&
&width=780
&height=330
&format=image%2Fpng
```

The standard specifies many of the parameters as being mandatory, GeoServer provides the *WMS Reflector* to allow many of them to be optionally specified.

Experimenting with this feature is a good way to get to know the GetMap parameters.

Example WMS request using a GetMap XML document is:

```
<?xml version="1.0" encoding="UTF-8"?>
<ogc:GetMap xmlns:ogc="http://www.opengis.net/ows"
            xmlns:gml="http://www.opengis.net/gml"
   version="1.1.1" service="WMS">
   <StyledLayerDescriptor version="1.0.0">
```

```
      <NamedLayer>
        <Name>topp:states</Name>
        <NamedStyle><Name>population</Name></NamedStyle>
      </NamedLayer>
   </StyledLayerDescriptor>
   <BoundingBox srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
      <gml:coord><gml:X>-130</gml:X><gml:Y>24</gml:Y></gml:coord>
      <gml:coord><gml:X>-55</gml:X><gml:Y>50</gml:Y></gml:coord>
   </BoundingBox>
   <Output>
      <Format>image/png</Format>
      <Size><Width>550</Width><Height>250</Height></Size>
   </Output>
</ogc:GetMap>
```

Time

As of GeoServer 2.2.0, GeoServer supports a TIME attribute for WMS GetMap requests as described in version 1.3 of the WMS specification. This parameter allows filtering a dataset by temporal slices as well as spatial tiles for rendering. See *Time Support in Geoserver WMS* for information on its use.

GetFeatureInfo

The **GetFeatureInfo** operation requests the spatial and attribute data for the features at a given location on a map. It is similar to the WFS *GetFeature* operation, but less flexible in both input and output. Since GeoServer provides a WFS service we recommend using it instead of `GetFeatureInfo` whenever possible.

The one advantage of `GetFeatureInfo` is that the request uses an (x,y) pixel value from a returned WMS image. This is easier to use for a naive client that is not able to perform true geographic referencing.

The standard parameters for the GetFeatureInfo operation are:

Parameter	Required?	Description
`service`	Yes	Service name. Value is `WMS`.
`version`	Yes	Service version. Value is one of `1.0.0`, `1.1.0`, `1.1.1`, `1.3`.
`request`	Yes	Operation name. Value is `GetFeatureInfo`.
`layers`	Yes	See *GetMap*
`styles`	Yes	See *GetMap*
`srs` *or* `crs`	Yes	See *GetMap*
`bbox`	Yes	See *GetMap*
`width`	Yes	See *GetMap*
`height`	Yes	See *GetMap*
`query_layers`	Yes	Comma-separated list of one or more layers to query.
`info_format`	No	Format for the feature information response. See below for values.
`feature_coun`	No	Maximum number of features to return. Default is 1.
`x` or `i`	Yes	X ordinate of query point on map, in pixels. 0 is left side. `i` is the parameter key used in WMS 1.3.0.
`y` or `j`	Yes	Y ordinate of query point on map, in pixels. 0 is the top. `j` is the parameter key used in WMS 1.3.0.
`exceptions`	No	Format in which to report exceptions. The default value is `application/vnd.ogc.se_xml`.

Geoserver supports a number of output formats for the `GetFeatureInfo` response. Server-styled HTML is the most commonly-used format. For maximum control and customisation the client should use GML3 and style the raw data itself. The supported formats are:

Format	Syntax	Notes
TEXT	`info_format=text/plain`	Simple text output. (The default format)
GML 2	`info_format=application/vnd.ogc.gml`	Works only for Simple Features (see *Complex Features*)
GML 3	`info_format=application/vnd.ogc.gml/3.1.1`	Works for both Simple and Complex Features (see *Complex Features*)
HTML	`info_format=text/html`	Uses HTML templates that are defined on the server. See *GetFeatureInfo Templates* for information on how to template HTML output.
JSON	`info_format=application/json`	Simple Json representation.
JSONP	`info_format=text/javascript`	Returns a JsonP in the form: `parseResponse(...json...)`. See *WMS vendor parameters* to change the callback name. Note that this format is disabled by default (See *Global variables affecting WMS*).

GeoServer provides the following vendor-specific parameters for the GetFeatureInfo operation. They are fully documented in the *WMS vendor parameters* section.

Parameter	Required?	Description
`buffer`	No	width of search radius around query point.
`cql_filter`	No	Filter for returned data, in ECQL format
`filter`	No	Filter for returned data, in OGC Filter format
`propertyName`	No	Feature properties to be returned

An example request for feature information from the `topp:states` layer in HTML format is:

```
http://localhost:8080/geoserver/wms?
request=GetFeatureInfo
&service=WMS
&version=1.1.1
&layers=topp%3Astates
&styles=
&srs=EPSG%3A4326
&format=image%2Fpng
&bbox=-145.151041%2C21.73192%2C-57.154894%2C58.961059
&width=780
&height=330
&query_layers=topp%3Astates
&info_format=text%2Fhtml
&feature_count=50
&x=353
&y=145
&exceptions=application%2Fvnd.ogc.se_xml
```

An example request for feature information in GeoJSON format is:

```
http://localhost:8080/geoserver/wms?
&INFO_FORMAT=application/json
&REQUEST=GetFeatureInfo
&EXCEPTIONS=application/vnd.ogc.se_xml
&SERVICE=WMS
&VERSION=1.1.1
&WIDTH=970&HEIGHT=485&X=486&Y=165&BBOX=-180,-90,180,90
&LAYERS=COUNTRYPROFILES:grp_administrative_map
```

```
&QUERY_LAYERS=COUNTRYPROFILES:grp_administrative_map
&TYPENAME=COUNTRYPROFILES:grp_administrative_map
```

The result will be:

```
{
"type":"FeatureCollection",
"features":[
   {
      "type":"Feature",
      "id":"dt_gaul_geom.fid-138e3070879",
      "geometry":{
         "type":"MultiPolygon",
         "coordinates":[
            [
               [
                  [
                     XXXXXXXXXX,
                     XXXXXXXXXX
                  ],
                  ...
                  [
                     XXXXXXXXXX,
                     XXXXXXXXXX
                  ]
               ]
            ]
         ]
      },
      "geometry_name":"at_geom",
      "properties":{
         "bk_gaul":X,
         "at_admlevel":0,
         "at_iso3":"XXX",
         "ia_name":"XXXX",
         "at_gaul_l0":X,
         "bbox":[
            XXXX,
            XXXX,
            XXXX,
            XXXX
         ]
      }
   }
],
"crs":{
   "type":"EPSG",
   "properties":{
      "code":"4326"
   }
},
"bbox":[
   XXXX,
   XXXX,
   XXXX,
   XXXX
]
}
```

DescribeLayer

The **DescribeLayer** operation is used primarily by clients that understand SLD-based WMS. In order to make an SLD one needs to know the structure of the data. WMS and WFS both have operations to do this, so the **DescribeLayer** operation just routes the client to the appropriate service.

The standard parameters for the DescribeLayer operation are:

Parameter	Required?	Description
`service`	Yes	Service name. Value is `WMS`.
`version`	Yes	Service version. Value is `1.1.1`.
`request`	Yes	Operation name. Value is `DescribeLayer`.
`layers`	Yes	See *GetMap*
`exceptions`	No	Format in which to report exceptions. The default value is `application/vnd.ogc.se_xml`.

Geoserver supports a number of output formats for the `DescribeLayer` response. Server-styled HTML is the most commonly-used format. The supported formats are:

Format	Syntax	Notes
TEXT	`output_format=text/xml`	Same as default.
GML 2	`output_format=application/vnd.ogc.wms_xml`	The default format.
JSON	`output_format=application/json`	Simple Json representation.
JSONP	`output_format=text/javascript`	Return a JsonP in the form: paddingOutput(...jsonp...). See *WMS vendor parameters* to change the callback name. Note that this format is disabled by default (See *Global variables affecting WMS*).

An example request in XML (default) format on a layer is:

```
http://localhost:8080/geoserver/topp/wms?service=WMS
&version=1.1.1
&request=DescribeLayer
&layers=topp:coverage
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE WMS_DescribeLayerResponse SYSTEM "http://localhost:8080/geoserver/schemas/wms/1.1.1/WMS_Des
<WMS_DescribeLayerResponse version="1.1.1">
   <LayerDescription name="topp:coverage" owsURL="http://localhost:8080/geoserver/topp/wcs?" owsType=
      <Query typeName="topp:coverage"/>
   </LayerDescription>
</WMS_DescribeLayerResponse>
```

An example request for feature description in JSON format on a layer group is:

```
http://localhost:8080/geoserver/wms?service=WMS
&version=1.1.1
&request=DescribeLayer
&layers=sf:roads,topp:tasmania_roads,nurc:mosaic
&outputFormat=application/json
```

The result will be:

```
{
version: "1.1.1",
layerDescriptions: [
{
```

```
   layerName: "sf:roads",
   owsURL: "http://localhost:8080/geoserver/wfs/WfsDispatcher?",
   owsType: "WFS",
   typeName: "sf:roads"
},
{
   layerName: "topp:tasmania_roads",
   owsURL: "http://localhost:8080/geoserver/wfs/WfsDispatcher?",
   owsType: "WFS",
   typeName: "topp:tasmania_roads"
},
{
   layerName: "nurc:mosaic",
   owsURL: "http://localhost:8080/geoserver/wcs?",
   owsType: "WCS",
   typeName: "nurc:mosaic"
}
]
}
```

GetLegendGraphic

The **GetLegendGraphic** operation provides a mechanism for generating legend graphics as images, beyond the LegendURL reference of WMS Capabilities. It generates a legend based on the style defined on the server, or alternatively based on a user-supplied SLD. For more information on this operation and the various options that GeoServer supports see *GetLegendGraphic*.

13.2.3 Time Support in Geoserver WMS

For layers that are properly configured with a TIME dimension, GeoServer supports a TIME attribute in GetMap requests to specify a temporal subset for rendering. For example, you might have a single dataset with weather observations collected over time and choose to plot a single day's worth of observations.

Specifying a Time

The format used for specifying a time in the WMS TIME parameter is based on ISO-8601. Times may be specified to a precision of 1 millisecond; GeoServer does not represent time queries with more precision than this. Times follow the general format:

```
yyyy-MM-ddThh:mm:ss.SSSZ
```

That is, a day specified by a 4-digit year, 2-digit month, and 2-digit day-of-month field, and an instant on that day specified by 2-digit hour, minute, and second fields, with an arbitrary number of decimal digits after the seconds field. The day and instant seconds are separated with a capital 'T', and the entire thing is suffixed with a 'Z' (indicating 'Zulu' or *UTC <http://en.wikipedia.org/wiki/Coordinated_Universal_Time>* for the time zone. The WMS specification does not provide for other time zones.)

GeoServer will apply the TIME value to all temporally enabled layers in the LAYERS parameter of the GetMap request. Layers without a temporal component will be served normally - allowing clients to include reference information like political boundaries along with temporal data.

Examples

- December 12, 2001 at 6:00 PM would be represented as:

  ```
  TIME=2001-12-12T18:00:00.0Z
  ```

- May 5, 1993 at 11:34 PM would be represented as:

  ```
  TIME=1993-05-05T11:34:00.0Z
  ```

Specifying a Periodicity

The periodicity is also specified in ISO-8601 format: a capital P followed by one or more interval lengths, each consisting of a number and a letter identifying a time unit:

Unit	Abbreviation
Years	`Y`
Months	`M`
Days	`D`
Hours	`H`
Minutes	`M`
Seconds	`S`

The Year/Month/Day group of values must be separated from the Hours/Minutes/Seconds group by a `T` character. Additionally, fields which contain a 0 may be omitted entirely, and the T may be omitted if hours, minutes, and seconds are all omitted. Fractional values are permitted, but only for the most specific value that is included.

The period must divide evenly into the interval defined by the start/end times.

Examples of Periods

- One hour:

  ```
  P0Y0M0DT1H0M0S
  ```

  ```
  PT1H0M0S
  ```

  ```
  PT1H
  ```

- 90 minutes (equivalent to 1 hour, 30 minutes):

  ```
  P0Y0M0DT1H30M0S
  ```

  ```
  PT1H30M
  ```

  ```
  P90M
  ```

- 18 months:

  ```
  P1Y6M0DT0H0M0S
  ```

  ```
  P1Y6M0D
  ```

  ```
  P0Y18M0DT0H0M0S
  ```

  ```
  P18M
  ```

```
*But not* `` P1.25Y3M
```

Specifying an Interval

A client may request information over a continuous interval instead of a single instant by specifying a start and end time, separated by a / character. In this scenario the start and end are both inclusive; that is, samples from exactly the endpoints of the specified range will be included in the rendered tile.

Examples

Description	Time specification
The month of September 2002	`2002-09-01T00:00:00.0Z/2002-10-01T23:59:59.999Z`
The entire day of December 25, 2010	`2010-12-25T00:00:00.0Z/2010-12-25T23:59:59.999Z`

Note: Because the time interval is inclusive, we cannot precisely specify a concept such as "all times within day x". We must choose between incorrectly accepting observations that occur at the end point, and incorrectly excluding some fraction of the final second of the interval. In practice, GeoServer and many data storage engines have limited resolution in their representations, so approximating a range to the nearest millisecond is 'as good as we can do.' It is possible that this technical constraint may be lifted at some point in the future.

Reduced Accuracy Times

The WMS specification also allows time specifications to be truncated, by omitting some suffix of the time string. In this case, GeoServer treats the time as a range whose length is equal to one of the most precise unit specified in the time string. If time specification omits all fields except year, it identifies a range one year long starting at the beginning of that year, etc.

GeoServer implements this by adding the appropriate unit, then subtracting 1 millisecond. This avoids surprising results when using an interval that aligns with the actual sampling frequency of the data - for example, if yearly data is natively stored with dates like 2001-01-01T00:00:00.0Z, 2002-01-01T00:00:00Z, etc. then a request for 2001 would include the samples for both 2001 and 2002 otherwise.

Examples

Description	Reduced Accuracy Time	Equivalent Range
The month of September 2002	`2002-09`	`2002-09-01T00:00:00.0Z/2002-10-01T23:59:59.999Z`
The day of December 25, 2010	`2010-12-25`	`2010-12-25T00:00:00.0Z/2010-12-25T23:59:59.999Z`

Ranges with Reduced Accuracy Times

Reduced accuracy times are also allowed when specifying ranges. In this case, GeoServer effectively expands the start and end times as described above, and then includes any samples from after the beginning of the start interval and before the end of the end interval.

Description	Reduced Accuracy Time	Equivalent Range
The months of September through December 2002	`2002-09/2002-12`	`2002-09-01T00:00:00.0Z/2002-12-31T23:59:59.9`
12pm through 6pm, December 25, 2010	`2010-12-25T12/2010-12-25T18`	`2010-12-25T12:00:00.0Z/2010-12-25T18:59:59.9`

Specifying a List of Times

For some formats, GeoServer can generate an animation. In this case, the client must specify multiple times, one for each frame. When multiple times are needed, the client should simply format each time as described above, and separate them with commas.

If the list is evenly spaced (for example, daily or hourly samples) then the list may be specified as a range, using a start time, end time, and period separated by slashes.

Examples

Description	List notation	Range notation
Noon every day for the week of August 12-18, 2012	`TIME=2012-08-12T12:00:00.0Z,2012-08-13T12:00:00.0Z,2012-08-14T12:00:00.0Z,2012`	`TIME=2012-08-12T12:00:00.0Z/2012-08-18T12:00:00.0Z/P1D`
Midnight on the first of September, October, and November 1999	`TIME=1999-09-01T00:00:00.0Z,1999-10-01T00:00:00.0Z,1999-11-01T00:00:00.0Z`	`TIME=1999-09-01T00:00:00.0Z/1999-11-01T00:00:00.0Z/P1M`

Note: GeoServer currently does not support lists of ranges, so all list queries effectively have a resolution of 1 millisecond. If you use reduced accuracy notation when specifying a range, each range will be automatically converted to the instant at the beginning of the range.

13.2.4 WMS output formats

WMS returns images in a number of possible formats. This page shows a list of the output formats. The syntax for setting an output format is:

```
format=<format>
```

where `<format>` is any of the options below.

Note: The list of output formats supported by a GeoServer instance can be found by a WMS *GetCapabilities* request.

Format	Syntax	Notes
PNG	`format=image/png`	Default
PNG8	`format=image/png8`	Same as PNG, but computes an optimal 256 color (8 bit) palette, so the image size is usually smaller
JPEG	`format=image/jpeg`	
GIF	`format=image/gif`	
TIFF	`format=image/tiff`	
TIFF8	`format=image/tiff8`	Same as TIFF, but computes an optimal 256 color (8 bit) palette, so the image size is usually smaller
GeoTIFF	`format=image/geotiff`	Same as TIFF, but includes extra GeoTIFF metadata
GeoTIFF8	`format=image/geotiff8`	Same as TIFF, but includes extra GeoTIFF metadata and computes an optimal 256 color (8 bit) palette, so the image size is usually smaller
SVG	`format=image/svg`	
PDF	`format=application/pdf`	
GeoRSS	`format=rss`	
KML	`format=kml`	
KMZ	`format=kmz`	
OpenLayers	`format=application/openlayers`	Generates an OpenLayers HTML application.

13.2.5 WMS vendor parameters

WMS vendor parameters are non-standard request parameters that are defined by an implementation to provide enhanced capabilities. GeoServer supports a variety of vendor-specific WMS parameters.

angle

The `angle` parameter rotates the output map clockwise around its center. The syntax is:

```
angle=<x>
```

where `<x>` is the number of degrees to rotate by.

Map rotation is supported in all raster formats, PDF, and SVG when using the Batik producer (which is the default).

buffer

The `buffer` parameter specifies the number of additional border pixels that are used in the `GetMap` and `GetFeatureInfo` operations. The syntax is:

```
buffer=<bufferwidth>
```

where `<bufferwidth>` is the width of the buffer in pixels.

In the *GetMap* operation, buffering includes features that lie outside the request bounding box, but whose styling is thick enough to be visible inside the map area.

In the *GetFeatureInfo* operation, buffering creates a "search radius" around the location of the request. Feature info is returned for features intersecting the search area. This is useful when working with an OpenLayers map (such as those generated by the *Layer Preview* page) since it relaxes the need to click precisely on a point for the appropriate feature info to be returned.

In both operations GeoServer attempts to compute the `buffer` value automatically by inspecting the styles for each layer. All active symbolizers are evaluated, and the size of the largest is used (i.e. largest point symbolizer, thickest line symbolizer). Automatic buffer sizing cannot be computed if:

- the SLD contains sizes that are specified as feature attribute values
- the SLD contains external graphics and does not specify their size explicitly

In this event, the following defaults are used:

- 0 pixels for *GetMap* requests
- 5 pixels for *GetFeatureInfo* requests (a different min value can be set via the `org.geoserver.wms.featureinfo.minBuffer` system variable, e.g., add `-Dorg.geoserver.wms.featureinfo.minBuffer=10` to make the min buffer be 10 pixels)

If these are not sufficiently large, the explicit parameter can be used.

cql_filter

The `cql_filter` parameter is similar to the standard `filter` parameter, but the filter is expressed using ECQL (Extended Common Query Language). ECQL provides a more compact and readable syntax compared to OGC XML filters. For full details see the *ECQL Reference* and *CQL and ECQL* tutorial.

If more than one layer is specified in the `layers` parameter, then a separate filter can be specified for each layer, separated by semicolons. The syntax is:

```
cql_filter=filter1;filter2...
```

An example of a simple CQL filter is:

```
cql_filter=INTERSECT(the_geom,%20POINT%20(-74.817265%2040.5296504))
```

env

The `env` parameter defines the set of substitution values that can be used in SLD variable substitution. The syntax is:

```
env=param1:value1;param2:value2;...
```

See *Variable substitution in SLD* for more information.

featureid

The `featureid` parameter filters by feature ID, a unique value given to all features. Multiple features can be selected by separating the featureids by comma, as in this example:

```
featureid=states.1,states.45
```

filter

The WMS specification allows only limited filtering of data. GeoServer enhances the WMS filter capability to match that provided by WFS. The `filter` parameter can specify a list of OGC XML filters. The list is enclosed in parentheses: **( )**. When used in a GET request, the XML tag brackets must be URL-encoded.

If more than one layer is specified in the `layers` parameter then a separate filter can be specified for each layer.

An example of an OGC filter encoded in a GET request is:

```
filter=%3CFilter%20xmlns:gml=%22http://www.opengis.net/gml%22%3E%3CIntersects%3E%3CPropertyName%3Ethe
```

format_options

The `format_options` is a container for parameters that are format-specific. The syntax is:

```
format_options=param1:value1;param2:value2;...
```

The supported format options are:

- `antialiasing` (values = `on`, `off`, `text`): controls the use of antialiased rendering in raster output.
- `callback`: specifies the callback function name for the jsonp response format (default is `parseResponse`).
- `dpi`: sets the rendering DPI (dots-per-inch) for raster outputs. The OGC standard output resolution is 90 DPI. If you need to create high resolution images (e.g for printing) it is advisable to request a larger image size and specify a higher DPI. In general, the image size should be increased by a factor equal to `targetDPI/90`, with the target dpi set in the format options. For example, to print a 100x100 image at 300 DPI request a 333x333 image with the DPI value set to 300: `&width=333&height=333&format_options=dpi:300`
- `layout`: specifies a layout name to use. Layouts are used to add decorators such as compasses and legends. This capability is discussed further in the *WMS Decorations* section.
- `quantizer` ((values = `octree`, `mediancut`): controls the color quantizer used to produce PNG8 images. GeoServer 2.2.0 provides two quantizers, a fast RGB quantizer called `octree` that does not handle translucency and a slower but more accurate RGBA quantizer called `mediancut`. By default the first is used on opaque images, whilst the second is enabled if the client asks for a transparent image (`transparent=true`). This vendor parameter can be used to manually force the usage of a particular quantizer.
- `kmattr` ((values = `true`,''false'')): determines whether the KML returned by GeoServer should include clickable attributes or not. This parameter primarily affects Google Earth rendering.
- `legend` ((values = `true`,''false'')): KML may add the legend.
- `kmscore` ((values = between `0` to force raster output and `100` to force vector output)): parameter sets whether GeoServer should render KML data as vector or raster. This parameter primarily affects Google Earth rendering.
- `kmltitle`: parameter sets the KML title.

maxFeatures and startIndex

The parameters `maxFeatures` and `startIndex` can be used together to provide "paging" support. Paging is helpful in situations such as KML crawling, where it is desirable to be able to retrieve the map in sections when there are a large number of features.

The `startindex=n` parameter specifies the index from which to start rendering in an ordered list of features. `n` must be a positive integer.

The `maxfeatures=n` parameter sets a limit on the amount of features rendered. `n` must be a positive integer. When used with `startindex`, the features rendered will be the ones starting at the `startindex` value.

Note that not all layers support paging. For a layer to be queried in this way, the underlying feature source must support paging. This is usually the case for databases (such as PostGIS).

namespace

The `namespace` parameter causes WMS *GetCapabilities* responses to be filtered to only contain layers in to a particular namespace. The syntax is:

```
namespace=<namespace>
```

where `<namespace>` is the namespace prefix.

Warning: Using an invalid namespace prefix will not cause an error, but the capabilities document returned will contain no layers, only layer groups.

Note: This affects the capabilities document only, not other requests. Other WMS operations will still process all layers, even when a namespace is specified.

palette

It is sometimes advisable (for speed and bandwidth reasons) to downsample the bit depth of returned maps. The way to do this is to create an image with a limited color palette, and save it in the `palettes` directory inside your GeoServer Data Directory. It is then possible to specify the `palette` parameter of the form:

```
palette=<image>
```

where `<image>` is the filename of the palette image (without the extension). To force a web-safe palette, use the syntax `palette=safe`. For more information see the tutorial on *Paletted Images*

propertyName

The `propertyName` parameter specifies which properties are included in the response of the `GetFeatureInfo` operation. The syntax is the same as in the WFS `GetFeature` operation. For a request for a single layer the syntax is:

```
propertyName=name1,...,nameN
```

For multiple layers the syntax is:

```
propertyName=(nameLayer11,...,nameLayer1N)...(name1LayerN,...,nameNLayerN)
```

The nature of the properties depends on the layer type:

- For vector layers the names specify the feature attributes.
- For raster layers the names specify the bands.
- For cascaded WMS layers the names specify the GML properties to be returned by the remote server.

tiled

Meta-tiling prevents issues with duplicated labels when using a tiled client such as OpenLayers. When meta-tiling is used, images are rendered and then split into smaller tiles (by default in a 3x3 pattern) before being served. In order for meta-tiling to work, the tile size *must* be set to 256x256 pixels, and the `tiled` and `tilesorigin` parameters must be specified.

The `tiled` parameter controls whether meta-tiling is used. The syntax is:

```
tiled=[yes|no]
```

To invoke meta-tiling use `tiled=yes`.

tilesorigin

The `tilesorigin` parameter is also required for meta-tiling. The syntax is:

```
tilesorigin=x,y
```

where `x` and `y` are the coordinates of the lower left corner (the "origin") of the tile grid system.

OpenLayers example

In OpenLayers, a good way to specify the `tilesorigin` is to reference the map extents directly.

> **Warning:** If the map extents are modified dynamically, the `tilesorigin` of each meta-tiled layer must be updated accordingly.

The following code shows how to specify the meta-tiling parameters:

```
var options = {
    ...
    maxExtent: new OpenLayers.Bounds(-180, -90, 180, 90),
    ...
};
map = new OpenLayers.Map('map', options);

tiled = new OpenLayers.Layer.WMS(
    "Layer name", "http://localhost:8080/geoserver/wms",
    {
        srs: 'EPSG:4326',
        width: 391,
        styles: '',
        height: 550,
        layers: 'layerName',
        format: 'image/png',
        tiled: true,
        tilesorigin: [map.maxExtent.left, map.maxExtent.bottom]
    },
    {buffer: 0}
);
```

13.2.6 WMS configuration

Layer Groups

A Layer Group is a group of layers that can be referred to by one layer name. For example, if you put three layers (call them layer_A, layer_B, and layer_C) under the one "Layer Group" layer, then when a user makes a WMS getMap request for that group name, they will get a map of those three layers.

For information on configuring Layer Groups in the Web Administration Interface see *Layer Groups*

Request limits

The request limit options allow the administrator to limit the resources consumed by each WMS `GetMap` request.

The following table shows the option names, a description, and the minimum GeoServer version at which the option is available (older versions will ignore it if set).

Option	Description	Version
maxRequestMemory	Sets the maximum amount of memory a single GetMap request is allowed to use (in kilobytes). The limit is checked before request execution by estimating how much memory would be required to produce the output in the format requested. For example, for an image format the estimate is based on the size of the required rendering memory (which is determined by the image size, the pixel bit depth, and the number of active FeatureTypeStyles at the requested scale). If the estimated memory size is below the limit, the request is executed; otherwise it is cancelled.	1.7.5
maxRenderingTime	Sets the maximum amount of time GeoServer will spend processing a request (in seconds). This time limits the "blind processing" portion of the request, that is, the time taken to read data and compute the output result (which may occur concurrently). If the execution time reaches the limit, the request is cancelled. The time required to write results back to the client is not limited by this parameter, since this is determined by the (unknown) network latency between the server and the client. For example, in the case of PNG/JPEG image generation, this option limits the data reading and rendering time, but not the time taken to write the image out.	1.7.5
maxRenderingErrors	Sets the maximum amount of rendering errors tolerated by a GetMap request. By default GetMap makes a best-effort attempt to serve the result, ignoring invalid features, reprojection errors and the like. Setting a limit on the number of errors ignored can make it easier to notice issues, and conserves CPU cycles by reducing the errors which must be handled and logged	1.7.5

The default value of each limit is `0`, which specifies that the limit is not applied.

If any of the request limits is exceeded, the GetMap operation is cancelled and a `ServiceException` is returned to the client.

When setting the above limits it is suggested that peak conditions be taken into consideration. For example, under normal circumstances a GetMap request may take less than a second. Under high load it is acceptable for it to take longer, but it's usually not desirable to allow a request to go on for 30 minutes.

The following table shows examples of reasonable values for the request limits:

Option	Value	Rationale
maxRequestMemory	16384	16MB are sufficient to render a 2048x2048 image at 4 bytes per pixel (full color and transparency), or a 8x8 meta-tile when using GeoWebCache or TileCache. Note that the rendering process uses a separate memory buffer for each FeatureTypeStyle in an SLD, so this also affects the maximum image size. For example, if an SLD contains two FeatureTypeStyle elements in order to draw cased lines for a highway, the maximum image size will be limited to 1448x1448 (the memory requirement increases with the product of the image dimensions, so halving the memory decreases image dimensions by only about 30%)
maxRenderingTime	120	A request that processes for a full two minutes is probably rendering too many features, regardless of the current server load. This may be caused by a request against a big layer using a style that does not have suitable scale dependencies
maxRenderingErrors	100	Encountering 100 errors is probably the result of a request trying to reproject a big data set into a projection that is not appropriate for the output extent, resulting in many reprojection failures.

13.2.7 Global variables affecting WMS

This document details the set of global variables that can affect WMS behaviour. Each global variable can be set as an environment variable, as a servlet context variable, or as a Java system property, just like the well known `GEOSERVER_DATA_DIRECTORY` setting. Refer to *Setting the Data Directory* for details on how a global variable can be specified.

MAX_FILTER_RULES

A integer number (defaults to 20) When drawing a style containing multiple active rules the renderer combines the filters of the rules in OR and adds them to the standard bounding box filter. This behaviour is active up until the maximum number of filter rules is reached, past that the rule filters are no more added to avoid huge queries. By default up to 20 rules are combined, past 20 rules only the bounding box filter is used. Turning it off (setting it to 0) can be useful if the styles are mostly classifications, detrimental if the rule filters are actually filtering a good amount of data out.

OPTIMIZE_LINE_WIDTH

Can be `true` or `false` (defaults to: `false`). When `true` any stroke whose width is less than 1.5 pixels gets slimmed down to "zero", which is actually not zero, but a very thin line. That was the behaviour GeoServer used to default to before the 2.0 series. When `false` the stroke width is not modified and it's possible to specify widths less than one pixel. This is the default behaviour starting from the 2.0.0 release

USE_STREAMING_RENDERER

Can be `true` or `false` (defaults to: `false`). When `true` the *StreamingRenderer* is used for all data. The *StreamingRenderer* is the one used by default for all data sources by shapefiles, it is usually faster at rendering styles with multiple `FeatureTypeStyle` elements but slower at rendering high amount of data.

ENABLE_JSONP

Can be `true` or `false` (defaults to: `false`). When `true` the JSONP (text/javascript) output format is enabled.

13.2.8 GetLegendGraphic

This chapter describes whether to use the GetLegendGraphics request. The SLD Specifications 1.0.0 gives a good description about GetLegendGraphic requests:

The GetLegendGraphic operation itself is optional for an SLD-enabled WMS. It provides a general mechanism for acquiring legend symbols, beyond the LegendURL reference of WMS Capabilities. Servers supporting the GetLegendGraphic call might code LegendURL references as GetLegendGraphic for interface consistency. Vendor-specific parameters may be added to GetLegendGraphic requests and all of the usual OGC-interface options and rules apply. No XML-POST method for GetLegendGraphic is presently defined.

Here is an example invocation:

```
http://localhost:8080/geoserver/wms?REQUEST=GetLegendGraphic&VERSION=1.0.0&FORMAT=image/png&WIDTH=20
```

which would produce four 20x20 icons that graphically represent the rules of the default style of the topp:states layer.

Figure 13.1: *Sample legend*

In the following table the whole set of GetLegendGraphic parameters that can be used.

Parameter	Required	Description
REQUEST	Required	Value must be "GetLegendRequest".
LAYER	Required	Layer for which to produce legend graphic.
STYLE	Optional	Style of layer for which to produce legend graphic. If not present, the default style is selected. The style may be any valid style available for a layer, including non-SLD internally-defined styles.
FEATURETYPE	Optional	Feature type for which to produce the legend graphic. This is not needed if the layer has only a single feature type.
RULE	Optional	Rule of style to produce legend graphic for, if applicable. In the case that a style has multiple rules but no specific rule is selected, then the map server is obligated to produce a graphic that is representative of all of the rules of the style.
SCALE	Optional	In the case that a RULE is not specified for a style, this parameter may assist the server in selecting a more appropriate representative graphic by eliminating internal rules that are out-of-scope. This value is a standardized scale denominator, defined in Section 10.2. Specifying the scale will also make the symbolizers using Unit Of Measure resize according to the specified scale.
SLD	Optional	This parameter specifies a reference to an external SLD document. It works in the same way as the SLD= parameter of the WMS GetMap operation.
SLD_BODY	Optional	This parameter allows an SLD document to be included directly in an HTTP-GET request. It works in the same way as the SLD_BODY= parameter of the WMS GetMap operation.
FORMAT	Required	This gives the MIME type of the file format in which to return the legend graphic. Allowed values are the same as for the FORMAT= parameter of the WMS GetMap request.
WIDTH	Optional	This gives a hint for the width of the returned graphic in pixels. Vector-graphics can use this value as a hint for the level of detail to include.
HEIGHT	Optional	This gives a hint for the height of the returned graphic in pixels.
EXCEPTIONS	Optional	This gives the MIME type of the format in which to return exceptions. Allowed values are the same as for the EXCEPTIONS= parameter of the WMS GetMap request.

Controlling legend appearance with LEGEND_OPTIONS

GeoServer allows finer control over the legend appearance via the vendor parameter `LEGEND_OPTIONS`. The general format of `LEGEND_OPTIONS` is the same as `FORMAT_OPTIONS`, that is:

```
...&LEGEND_OPTIONS=key1:v1;key2:v2;...;keyn:vn
```

Here is a description of the various parameters that can be used in `LEGEND_OPTIONS`:

- **fontName (string)** the name of the font to be used when generating rule titles. The font must be available on the server
- **fontStyle (string)** can be set to italic or bold to control the text style. Other combination are not allowed right now but we could implement that as well.
- **fontSize (integer)** allows us to set the Font size for the various text elements. Notice that default size is 12.
- **fontColor (hex)** allows us to set the color for the text of rules and labels (see above for recommendation on how to create values). Values are expressed in `0xRRGGBB` format

- **fontAntiAliasing (true/false)** when true enables antialiasing for rule titles
- **bgColor (hex)** background color for the generated legend, values are expressed in `0xRRGGBB` format
- **dpi (integer)** sets the DPI for the current request, in the same way as it is supported by GetMap. Setting a DPI larger than 91 (the default) makes all fonts, symbols and line widths grow without changing the current scale, making it possible to get a high resolution version of the legend suitable for inclusion in printouts
- **forceLabels** "on" means labels will always be drawn, even if only one rule is available. "off" means labels will never be drawn, even if multiple rules are available. Off by default.

Here is a sample request sporting all the options:

```
http://localhost:8080/geoserver/wms?REQUEST=GetLegendGraphic&VERSION=1.0.0&FORMAT=image/png&WIDTH=20
```

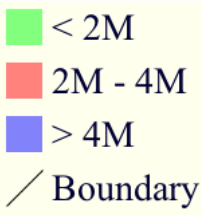


Figure 13.2: *Using LEGEND_OPTIONS to control the output*

Raster Legends Explained

This chapter aim to briefly describe the work that I have performed in order to support legends for raster data that draw information taken from the various bits of the SLD 1.0 RasterSymbolizer element. Recall, that up to now there was no way to create legends for raster data, therefore we have tried to fill the gap by providing an implementation of the getLegendGraphic request that would work with the ColorMap element of the SLD 1.0 RasterSymbolizer. Notice that some "debug" info about the style, like colormap type and band used are printed out as well.

What's a raster legend

Here below I have drawn the structure of a typical legend, where some elements of interests are parameterized.

Take as an instance one of the SLD files attached to this page, each row in the above table draws its essence from the ColorMapEntry element as shown here below:

```
<ColorMapEntry color="#732600" quantity="9888" opacity="1.0" label="<-70 mm"/>
```

The producer for the raster legend will make use of this elements in order to build the legend, with this regards, notice that:

- the width of the Color element is driven by the requested width for the GetLegendGraphic request
- the width and height of label and rules is computed accordingly to the used Font and Font size for the prepared text (**no new line management for the moment**)
- the height of the Color element is driven by the requested width for the GetLegendGraphic request, but notice that for ramps we expand this a little since the goal is to turn the various Color elements into a single long strip

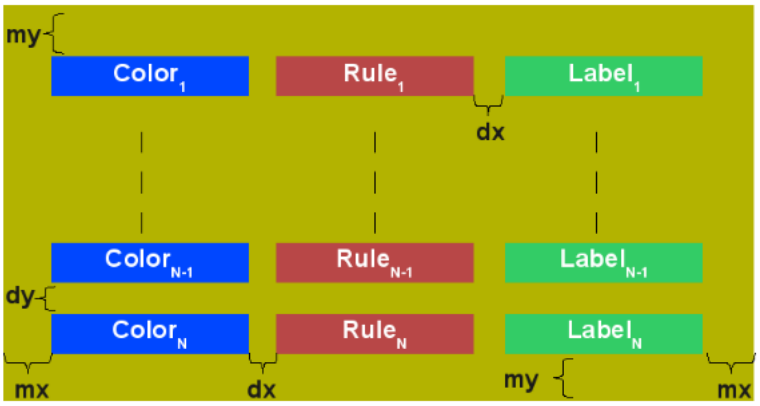


Figure 13.3: *The structure of a typical legend*

- the height of each row is set to the maximum height of the single elements
- the width of each row is set to the sum of the width of the various elements plus the various paddings
- **dx,dy** the spaces between elements and rows are set to the 15% of the requested width and height. Notice that **dy** is ignored for the colormaps of type **ramp** since they must create a continous color strip.
- **mx,my** the margins from the border of the legends are set to the 1.5% of the total size of the legend

Just to jump right to the conclusions (which is a bad practice I know, but no one is perfect), here below I am adding an image of a sample legend with all the various options at work. The request that generated it is the following:

```
http://localhost:8081/geoserver/wms?REQUEST=GetLegendGraphic&VERSION=1.0.0&FORMAT=image/png&WIDTH=10(
```

Do not worry if it seems like something written in ancient dead language, I am going to explain the various params here below. Nevertheless it is important to point out that basic info on how to create and set params can be found in this page.

Raster legends' types

As you may know (well, actually you might not since I never wrote any real docs about the RasterSymbolizer work I did) GeoServer supports three types of ColorMaps:

- **ramp** this is what SLD 1.0 dictates, which means a linear interpolation weighted on values between the colors of the various ColorMapEntries.
- **values** this is an extensions that allows link quantities to colors as specified by the ColorMapEntries quantities. Values not specified are translated into transparent pixels.
- **classes** this is an extensions that allows pure classifications based o intervals created from the ColorMapEntries quantities. Values not specified are translated into transparent pixels.

Here below I am going to list various examples that use the attached styles on a rainfall floating point geotiff.

ColorMap type is VALUES

Refer to the SLD rainfall.sld in attachment.

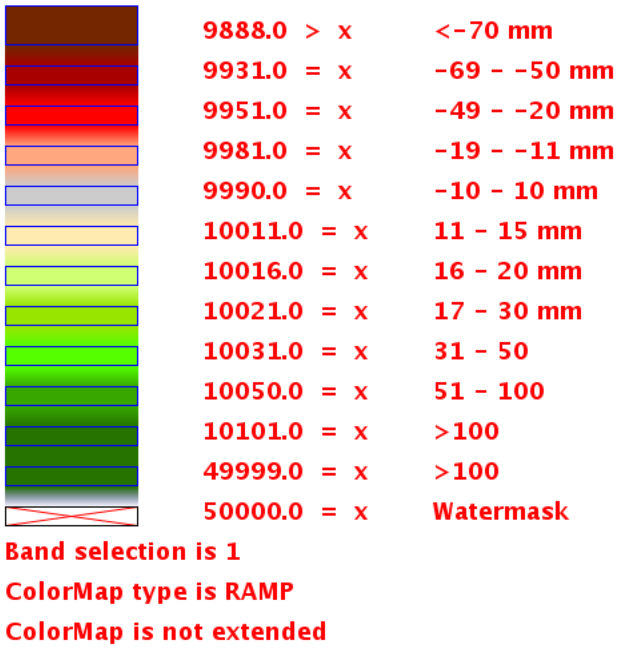


Figure 13.4: *Example of a raster legend*

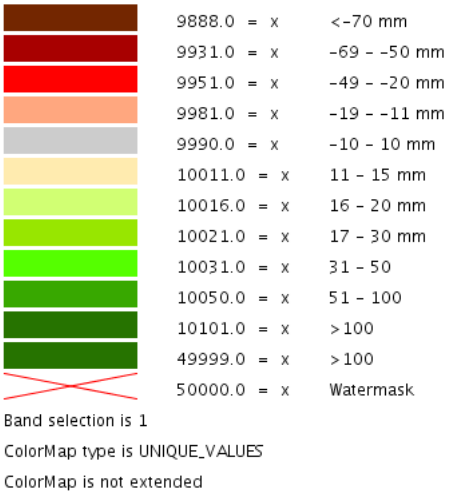


Figure 13.5: *Raster legend - VALUES type*

ColorMap type is CLASSES

Refer to the SLD rainfall_classes.sld in attachment.

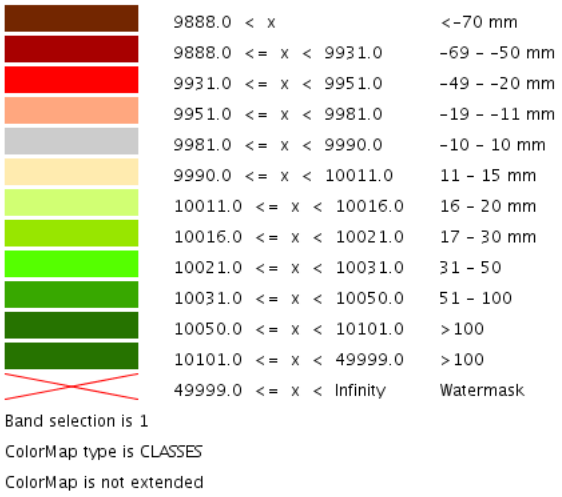


Figure 13.6: *Raster legend - CLASSES type*

ColorMap type is RAMP

Refer to the SLD rainfall_classes.sld in attachment. Notice that the first legend show the default border behavior while the second has been force to draw a border for the breakpoint color of the the colormap entry quantity described by the rendered text. Notice that each color element has a part that show the fixed color from the colormap entry it depicts (the lowest part of it, the one that has been outlined by the boder in the second legend here below) while the upper part of the element has a gradient tha connects each element to the previous one to point out the fact that we are using linear interpolation.

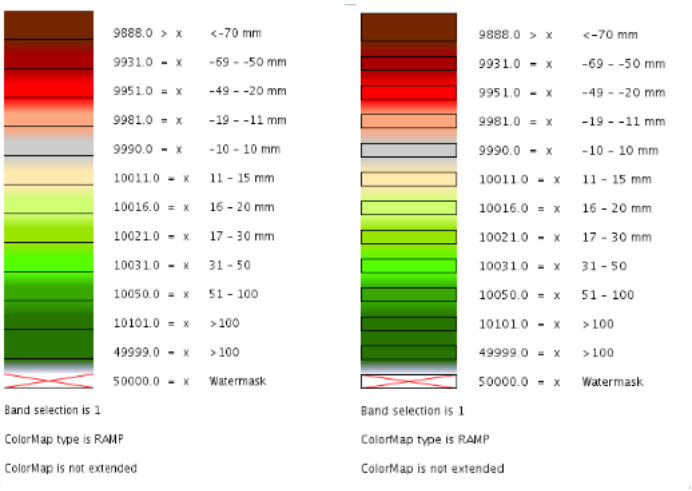


Figure 13.7: *Raster legend - RAMP type*

The various control parameters and how to set them

I am now going to briefly explain the various parameters tht we can use to control the layout and content of the legend (refer also to this page). Here below I have put a request that puts all the various options at tow:

```
http://localhost:8081/geoserver/wms?REQUEST=GetLegendGraphic&VERSION=1.0.0&FORMAT=image/png&WIDTH=10(
```

Let's now examine all the interesting elements, one by one. Notice that I am not going to discuss the mechanics of the GetLegendGraphic operation, for that you may want to refer to the SLD 1.0 spec, my goal is to briefly discuss the LEGEND_OPTIONS parameter.

- **forceRule (boolean)** by defaul rules for a ColorMapEntry are not drawn to keep the legend small and compact, unless there are not labels at all. You can change this behaviour by setting this parameter to true.
- **dx,dy,mx,my (double)** can be used to set the margin and the buffers between elements
- **border (boolean)** activates or deactivates the boder on the color elements in order to make the separations cleare. Notice that I decided to **always** have a line that would split the various color elements for the ramp type of colormap.
- **borderColor (hex)** allows us to set the color for the border in 0xRRGGBB format

13.3 Web Coverage Service

13.3.1 WCS basics

GeoServer provides support for Open Geospatial Consortium (OGC) Web Map Service (WCS) versions 1.0 and 1.1. One can think of WCS as the equivalent of *Web Feature Service*, but for raster data instead of vector data. It lets you get at the raw coverage information, not just the image. GeoServer is the reference implementation for WCS 1.1.

13.3.2 WCS reference

Introduction

The Web Coverage Service (WCS) is a standard created by the OGC that refers to the receiving of geospatial information as 'coverages': digital geospatial information representing space-varying phenomena. One can think of it as *Web Feature Service* for *raster* data. It gets the 'source code' of the map, but in this case its not raw vectors but raw imagery.

An important distinction must be made between WCS and *Web Map Service*. They are similar, and can return similar formats, but a WCS is able to return more information, including valuable metadata and more formats. It additionally allows more precise queries, potentially against multi-dimensional backend formats.

Benefits of WCS

WCS provides a standard interface for how to request the raster source of a geospatial image. While a WMS can return an image it is generally only useful as an image. The results of a WCS can be used for complex modeling and analysis, as it often contains more information. It also allows more complex querying - clients can extract just the portion of the coverage that they need.

Operations

WCS can perform the following operations:

Operation	Description
`GetCapabilities`	Retrieves a list of the server's data, as well as valid WCS operations and parameters
`DescribeCoverage`	Retrieves an XML document that fully describes the request coverages.
`GetCoverage`	Returns a coverage in a well known format. Like a WMS GetMap request, but with several extensions to support the retrieval of coverages.

GetCapabilities

The **GetCapabilities** operation is a request to a WCS server for a list of what operations and services ("capabilities") are being offered by that server.

A typical GetCapabilities request would look like this (at URL `http://www.example.com/wcs`):

Using a GET request (standard HTTP):

```
http://www.example.com/wcs?
service=wcs&
AcceptVersions=1.1.0&
request=GetCapabilities
```

Here there are three parameters being passed to our WCS server, `service=wcs`, `AcceptVersions=1.1.0`, and `request=GetCapabilities`. At a bare minimum, it is required that a WCS request have the service and request parameters. GeoServer relaxes these requirements (setting the default version if omitted), but "officially" they are mandatory, so they should always be included. The *service* key tells the WCS server that a WCS request is forthcoming. The *AcceptsVersion* key refers to which version of WCS is being requested. The *request* key is where the actual GetCapabilities operation is specified.

WCS additionally supports the Sections parameter that lets a client only request a specific section of the Capabilities Document.

DescribeCoverage

The purpose of the **DescribeCoverage** request is to additional information about a Coverage a client wants to query. It returns information about the crs, the metadata, the domain, the range and the formats it is available in. A client generally will need to issue a DescribeCoverage request before being sure it can make the proper GetCoverage request.

GetCoverage

The **GetCoverage** operation requests the actual spatial data. It can retrieve subsets of coverages, and the result can be either the coverage itself or a reference to it. The most powerful thing about a GetCoverage request is its ability to subset domains (height and time) and ranges. It can also do resampling, encode in different data formats, and return the resulting file in different ways.

13.3.3 WCS output formats

WCS output formats are configured coverage by coverage. The current list of output formats follows:

Images:

- JPEG - (format=jpeg)
- GIF - (format=gif)
- PNG - (format=png)
- Tiff - (format=tif)
- BMP - (format=bmp)

Georeferenced formats:

- GeoTiff - (format=geotiff)
- GTopo30 - (format=gtopo30)
- ArcGrid - (format=ArcGrid)
- GZipped ArcGrid - (format=ArcGrid-GZIP)

Beware, in the case of ArcGrid, the GetCoverage request must make sure the x and y resolution are equal, otherwise an exception will be thrown (ArcGrid is designed to have square cells).

13.3.4 WCS Vendor Parameters

Requests to the WCS GetCapabilities operation can be filtered to only return layers corresponding to a particular namespace.

Sample code:

```
http://example.com/geoserver/wcs?
   service=wcs&
   version=1.0.0&
   request=GetCapabilities&
   namespace=topp
```

Using an invalid namespace prefix will not cause any errors, but the document returned will not contain information on any layers.

13.3.5 WCS configuration

Coverage processing

The WCS processing chain can be tuned in respect of how raster overviews and read subsampling are used.

The overview policy has four possible values:

Option	Description	Version
Lower resolution overview	Looks up the two overviews with a resolution closest to the one requested and chooses the one at the lower resolution.	2.0.3
Don't use overviews	Overviews will be ignored, the data at its native resolution will be used instead. This is the default value.	2.0.3
Higher resolution overview	Looks up the two overviews with a resolution closest to the one requested and chooses the one at the higher resolution.	2.0.3
Closest overview	Looks up the overview closest to the one requested	2.0.3

While reading coverage data at a resolution lower than the one available on persistent storage its common to use subsampling, that is, read one every N pixels as a way to reduce the resolution of the data read in memory. **Use subsampling** controls wheter subsampling is enabled or not.

Request limits

The request limit options allow the administrator to limit the resources consumed by each WCS `GetCoverage` request.

The request limits limit the size of the image read from the source and the size of the image returned to the client. Both of these limits are to be considered a worst case scenario and are setup to make sure the server never gets asked to deal with too much data.

Option	Description	Version
Maximum input memory	Sets the maximum amount of memory, in kilobytes, a GetCoverage request might use, at most, to read a coverage from the data source. The memory is computed as `rw * rh * pixelsize`, where `rw` and `rh` are the size of the raster to be read and `pixelsize` is the dimension or a pixel (e.g., a RGBA image will have 32bit pixels, a batimetry might have 16bit signed int ones)	2.0.3
Maximum output memory	Sets the maximum amount of memory, in kilobytes, a GetCoverage request might use, at most, to host the resulting raster. The memory is computed as `ow * oh * pixelsize`, where `ow` and `oh` are the size of the raster to be generated in output.	2.0.3

To understand the limits let's consider a very simplified examle in which no tiles and overviews enter the game:

- The request hits a certain area of the original raster. Reading it at full resolution requires grabbing a raster of size `rw * rh`, which has a certain number of bands, each with a certain size. The amount of memory used for the read will be `rw * rh * pixelsize`. This is the value measured by the input memory limit
- The WCS performs the necessary processing: band selection, resolution change (downsampling or upsampling), reprojection
- The resuling raster will have size `ow * oh` and will have a certain number of bands, possibly less than the input data, each with a certain size. The amount of memory used for the final raster will be `ow * oh * pixelsize`. This is the value measured by the output memory limit.
- Finally the resulting raster will be encoded in the output format. Depending on the output format structure the size of the result might be higher than the in memory size (ArcGrid case) or smaller (for example in the case of GeoTIFF output, which is normally LZW compressed)

In fact reality is a bit more complicated:

- The input source might be tiled, which means there is no need to fully read in memory the region, but it is sufficient to do so one tile at a time. The input limits won't consider inner tiling when computing the limits, but if all the input coverages are tiled the input limits should be designed considering the amount of data to be read from the persistent storage as opposed to the amount of data to be stored in memory
- The reader might be using overviews or performing subsampling during the read to avoid actually reading all the data at the native resolution should the output be subsampled
- The output format might be tile aware as well (GeoTIFF is), meaning it might be able to write out one tile at a time. In this case not even the output raster will be stored in memory fully at any given time.

Only a few input formats are so badly structure that they force the reader to read the whole input data in one shot, and should be avoided. Examples are: * JPEG or PNG images with world file * Single tiled and JPEG compressed GeoTIFF files

13.4 Virtual OWS Services

The different types of services in GeoServer include WFS, WMS, and WCS, commonly referred to as "OWS" services. These services are global in that each service publishes ever layer configured on the server. WFS publishes all vector layer (feature types), WCS publishes all raster layers (coverages), and WMS publishes everything.

A *virtual service* is a view of the global service that consists only of a subset of the layers. Virtual services are based on GeoServer workspaces. For each workspace that exists a virtual service exists along with it. The virtual service publishes only those layers that fall under the corresponding workspace.

Warning: Virtual services only apply to the core OWS services, and not OWS services accessed through GeoWebCache. It also does not apply to other subsystems such as REST.

When a client accesses a virtual service that client only has access to those layers published by that virtual service. Access to layers in the global service via the virtual service will result in an exception. This makes virtual services ideal for compartmentalizing access to layers. A service provider may wish to create multiple services for different clients handing one service url to one client, and a different service url to another client. Virtual services allow the service provider to achieve this with a single GeoServer instance.

13.4.1 Filtering by workspace

Consider the following snippets of the WFS capabilities document from the GeoServer release configuration that list all the feature types:

```
http://localhost:8080/geoserver/wfs?request=GetCapabilities

<wfs:WFS_Capabilities>

  <FeatureType xmlns:tiger="http://www.census.gov">
   <Name>tiger:poly_landmarks</Name>
--
  <FeatureType xmlns:tiger="http://www.census.gov">
   <Name>tiger:poi</Name>
--
  <FeatureType xmlns:tiger="http://www.census.gov">
   <Name>tiger:tiger_roads</Name>
--
  <FeatureType xmlns:sf="http://www.openplans.org/spearfish">
   <Name>sf:archsites</Name>
--
  <FeatureType xmlns:sf="http://www.openplans.org/spearfish">
   <Name>sf:bugsites</Name>
--
  <FeatureType xmlns:sf="http://www.openplans.org/spearfish">
   <Name>sf:restricted</Name>
--
  <FeatureType xmlns:sf="http://www.openplans.org/spearfish">
   <Name>sf:roads</Name>
--
```

```
  <FeatureType xmlns:sf="http://www.openplans.org/spearfish">
   <Name>sf:streams</Name>
--
  <FeatureType xmlns:topp="http://www.openplans.org/topp">
   <Name>topp:tasmania_cities</Name>
--
  <FeatureType xmlns:topp="http://www.openplans.org/topp">
   <Name>topp:tasmania_roads</Name>
--
  <FeatureType xmlns:topp="http://www.openplans.org/topp">
   <Name>topp:tasmania_state_boundaries</Name>
--
  <FeatureType xmlns:topp="http://www.openplans.org/topp">
   <Name>topp:tasmania_water_bodies</Name>
--
  <FeatureType xmlns:topp="http://www.openplans.org/topp">
   <Name>topp:states</Name>
--
  <FeatureType xmlns:tiger="http://www.census.gov">
   <Name>tiger:giant_polygon</Name>

</wfs:WFS_Capabilities>
```

The above document lists every feature type configured on the server. Now consider the following capabilities request:

```
http://localhost:8080/geoserver/topp/wfs?request=GetCapabilities
```

The part of interest in the above request is the "topp" prefix to the wfs service. The above url results in the following feature types in the capabilities document:

```
<wfs:WFS_Capabilities>

   <FeatureType xmlns:topp="http://www.openplans.org/topp">
    <Name>topp:tasmania_cities</Name>
 --
   <FeatureType xmlns:topp="http://www.openplans.org/topp">
    <Name>topp:tasmania_roads</Name>
 --
   <FeatureType xmlns:topp="http://www.openplans.org/topp">
    <Name>topp:tasmania_state_boundaries</Name>
 --
   <FeatureType xmlns:topp="http://www.openplans.org/topp">
    <Name>topp:tasmania_water_bodies</Name>
 --
   <FeatureType xmlns:topp="http://www.openplans.org/topp">
    <Name>topp:states</Name>

 </wfs:WFS_Capabilities>
```

The above feature types correspond to those configured on the server as part of the "topp" workspace.

The consequence of a virtual service is not only limited to the capabilities document of the service. When a client accesses a virtual service it is restricted to only those layers for all operations. For instance, consider the following WFS feature request:

```
http://localhost:8080/geoserver/topp/wfs?request=GetFeature&typename=tiger:roads
```

The above request results in an exception. Since the request feature type "tiger:roads" is not in the "topp"

workspace the client will receive an error stating that the requested feature type does not exist.

13.4.2 Filtering by layer

It is possible to further filter a global service by specifying the name of layer as part of the virtual service. For instance consider the following capabilities document:

```
http://localhost:8080/geoserver/topp/states/wfs?request=GetCapabilities
```

The part of interest is the "states" prefix to the wfs service. The above url results in the following capabilities document that contains a single feature type:

```
<wfs:WFS_Capabilities>

  <FeatureType xmlns:topp="http://www.openplans.org/topp">
   <Name>topp:states</Name>

<wfs:WFS_Capabilities>
```

13.4.3 Turning off global services

It is possible to completely restrict access to the global OWS services by setting a configuration flag. When global access is disabled OWS services may only occur through a virtual service. Any client that tries to access a service globally will receive an exception.

To disable global services log into the GeoServer web administration interface and navigate to "Global Settings". Uncheck the "Enable Global Services" check box.

CHAPTER 14

REST configuration

GeoServer provides a RESTful interface through which clients can retrieve information about an instance and make configuration changes. Using the REST interface's simple HTTP calls, clients can configure GeoServer without needing to use the *Web Administration Interface*.

REST is an acronym for "REpresentational State Transfer". REST adopts a fixed set of operations on named resources, where the representation of each resource is the same for retrieving and setting information. In other words, you can retrieve (read) data in an XML format and also send data back to the server in similar XML format in order to set (write) changes to the system.

Operations on resources are implemented with the standard primitives of HTTP: GET to read; and PUT, POST, and DELETE to write changes. Each resource is represented as a URL, such as `http://GEOSERVER_HOME/rest/workspaces/topp`.

For further information about the REST API, refer to the *REST configuration API reference* section. For practical examples, refer to the *REST configuration examples* section.

14.1 REST configuration API reference

This section describes the GeoServer REST configuration API.

14.1.1 API details

This page contains information on the REST API architecture.

Authentication

REST requires that the client be authenticated. By default, the method of authentication used is Basic authentication. See the *Security* section for how to change the authentication method.

Status codes

An HTTP request uses a status code to relay the outcome of the request to the client. Different status codes are used for various purposes throughout this document. These codes are described in detail by the HTTP specification.

The most common status codes are listed below, along with their descriptions:

Status code	Description	Notes
200	OK	The request was successful
201	Created	A new resource was successfully created, such as a new feature type or data store
403	Forbidden	Often denotes a permissions mismatch
404	Not Found	Endpoint or resource was not at the indicated location
405	Method Not Allowed	Often denotes an endpoint accessed with an incorrect operation (for example, a GET request where a PUT/POST is indicated)
500	Internal Server Error	Often denotes a syntax error in the request

Formats and representations

A `format` specifies how a particular resource should be represented. A format is used:

- In an operation to specify what representation should be returned to the client
- In a POST or PUT operation to specify the representation being sent to the server

In a **GET** operation the format can be specified in two ways.

There are two ways to specify the format for a GET operation. The first option uses the `Accept` header. For example, with the header set to `"Accept: text/xml"` the resource would be returned as XML. The second option of setting the format is via a file extension. For example, given a resource `foo`, to request a representation of `foo` as XML, the request URI would end with `/foo.xml`. To request a representation as JSON, the request URI would end with `/foo.json`. When no format is specified the server will use its own internal format, usually HTML. When the response format is specified both by the header and by the extension, the format specified by the extension takes precedence.

In a **POST** or **PUT** operation, the format of content being sent to the server is specified with the `Content-type` header. For example, to send a representation in XML, use `"Content-type: text/xml"` or `"Content-type: application/xml"`. As with GET requests, the representation of the content returned from the server is specified by the `Accept` header or by the format.

The following table defines the `Content-type` values for each format:

Format	Content-type
XML	`application/xml`
JSON	`application/json`
HTML	`application/html`
SLD	`application/vnd.ogc.sld+xml`
ZIP	`application/zip`

14.1.2 Global settings

Allows access to GeoServer's global settings.

`/settings[.<format>]`

Controls all global settings.

Method	Action	Status code	Formats	Default Format
GET	List all global settings	200	HTML, XML, JSON	HTML
POST		405		
PUT	Update global settings	200	XML, JSON	
DELETE		405		

`/settings/contact[.<format>]`

Controls global contact information only.

Method	Action	Status code	Formats	Default Format
GET	List global contact information	200	HTML, XML, JSON	HTML
POST		405		
PUT	Update global contact	200	XML, JSON	
DELETE		405		

14.1.3 Workspaces

A `workspace` is a grouping of data stores. Similar to a namespace, it is used to group data that is related in some way.

`/workspaces[.<format>]`

Controls all workspaces.

Method	Action	Status code	Formats	Default Format
GET	List all workspaces	200	HTML, XML, JSON	HTML
POST	Create a new workspace	201 with `Location` header	XML, JSON	
PUT		405		
DELETE		405		

`/workspaces/<ws>[.<format>]`

Controls a specific workspace.

Method	Action	Status code	Formats	Default Format	Parameters
GET	Return workspace `ws`	200	HTML, XML, JSON	HTML	
POST		405			
PUT	200	Modify workspace `ws`	XML, JSON		
DELETE	200	Delete workspace `ws`	XML, JSON		*recurse*

Exceptions

Exception	Status code
GET for a workspace that does not exist	404
PUT that changes name of workspace	403
DELETE against a workspace that is non-empty	403

Parameters

recurse The `recurse` parameter recursively deletes all layers referenced by the specified workspace, including data stores, coverage stores, feature types, and so on. Allowed values for this parameter are "true" or "false". The default value is "false".

/workspaces/default[.<format>]

Controls the default workspace.

Method	Action	Status code	Formats	Default Format
GET	Returns default workspace	200	HTML, XML, JSON	HTML
POST		405		
PUT	200	Set default workspace	XML, JSON	
DELETE		405		

/workspaces/<ws>/settings[.<format>]

Controls settings on a specific workspace.

Method	Action	Status code	Formats	Default Format
GET	Returns workspace settings	200	HTML, XML, JSON	HTML
POST		405		
PUT	Creates or updates workspace settings	200	XML, JSON	
DELETE	Deletes workspace settings	200	XML, JSON	

14.1.4 Namespaces

A `namespace` is a uniquely identifiable grouping of feature types. It is identified by a prefix and a URI.

/namespaces[.<format>]

Controls all namespaces.

Method	Action	Status code	Formats	Default Format
GET	List all namespaces	200	HTML, XML, JSON	HTML
POST	Create a new namespace	201 with `Location` header	XML, JSON	
PUT		405		
DELETE		405		

/namespaces/<ns>[.<format>]

Controls a particular namespace.

Method	Action	Status code	Formats	Default Format
GET	Return namespace `ns`	200	HTML, XML, JSON	HTML
POST		405		
PUT	200	Modify namespace `ns`	XML, JSON	
DELETE	200	Delete namespace `ns`	XML, JSON	

Exceptions

Exception	Status code
GET for a namespace that does not exist	404
PUT that changes prefix of namespace	403
DELETE against a namespace whose corresponding workspace is non-empty	403

`/namespaces/default[.<format>]`

Controls the default namespace.

Method	Action	Status code	Formats	Default Format
GET	Return default namespace	200	HTML, XML, JSON	HTML
POST		405		
PUT	200	Set default namespace	XML, JSON	
DELETE		405		

14.1.5 Data stores

A `data store` contains vector format spatial data. It can be a file (such as a shapefile), a database (such as PostGIS), or a server (such as a *remote Web Feature Service*).

`/workspaces/<ws>/datastores[.<format>]`

Controls all data stores in a given workspace.

Method	Action	Status code	Formats	Default Format
GET	List all data stores in workspace `ws`	200	HTML, XML, JSON	HTML
POST	Create a new data store	201 with `Location` header	XML, JSON	
PUT		405		
DELETE		405		

`/workspaces/<ws>/datastores/<ds>[.<format>]`

Controls a particular data store in a given workspace.

Method	Action	Status code	Formats	Default Format	Parameters
GET	Return data store `ds`	200	HTML, XML, JSON	HTML	
POST		405			
PUT	Modify data store `ds`				
DELETE	Delete data store `ds`				*recurse*

Exceptions

Exception	Status code
GET for a data store that does not exist	404
PUT that changes name of data store	403
PUT that changes workspace of data store	403
DELETE against a data store that contains configured feature types	403

Parameters

recurse The `recurse` parameter recursively deletes all layers referenced by the specified data store. Allowed values for this parameter are "true" or "false". The default value is "false".

/workspaces/<ws>/datastores/<ds>/[file|url|external][.<extension>]

These endpoints (`file`, `url`, and `external`) allow a file containing spatial data to be added (via a POST or PUT) into an existing data store, or will create a new data store if it doesn't already exist. The three endpoints are used to specify the method that is used to upload the file:

- `file`—Uploads a file from a local source. The body of the request is the file itself.
- `url`—Uploads a file from an remote source. The body of the request is a URL pointing to the file to upload. This URL must be visible from the server.
- `external`—Uses an existing file on the server. The body of the request is the absolute path to the existing file.

Method	Action	Status code	Formats	Default Format	Parameters
GET	*Deprecated*. Retrieve the underlying files for the data store as a zip file with MIME type `application/zip`	200			
POST		405			
PUT	Uploads files to the data store `ds`, creating it if necessary	200	*See note below*		*configure*, *target*, *update*, *charset*
DELETE		405			

Exceptions

Exception	Status code
GET for a data store that does not exist	404
GET for a data store that is not file based	404

Parameters

extension The `extension` parameter specifies the type of data being uploaded. The following extensions are supported:

Extension	Datastore
shp	Shapefile
properties	Property file
h2	H2 Database
spatialite	SpatiaLite Database

Note: A file can be PUT to a data store as a standalone or zipped archive file. Standalone files are only suitable for data stores that work with a single file such as a GML store. Data stores that work with multiple files, such as the shapefile store, must be sent as a zip archive.

When uploading a standalone file, set the `Content-type` appropriately based on the file type. If you are loading a zip archive, set the `Content-type` to `application/zip`.

configure The `configure` parameter controls how the data store is configured upon file upload. It can take one of the three values:

- `first`—(*Default*) Only setup the first feature type available in the data store.
- `none`—Do not configure any feature types.
- `all`—Configure all feature types.

target The `target` parameter determines what format or storage engine will be used when a new data store is created on the server for uploaded data. When importing data into an existing data store, it is ignored. The allowed values for this parameter are the same as for the *extension parameter*.

update The `update` parameter controls how existing data is handled when the file is PUT into a data store that already exists and already contains a schema that matches the content of the file. The parameter accepts one of the following values:

- `append`—Data being uploaded is appended to the existing data. This is the default.
- `overwrite`—Data being uploaded replaces any existing data.

charset The `charset` parameter specifies the character encoding of the file being uploaded (such as "ISO-8559-1").

14.1.6 Feature types

A `feature type` is a vector based spatial resource or data set that originates from a data store. In some cases, such as with a shapefile, a feature type has a one-to-one relationship with its data store. In other cases, such as PostGIS, the relationship of feature type to data store is many-to-one, feature types corresponding to a table in the database.

`/workspaces/<ws>/datastores/<ds>/featuretypes[.<format>]`

Controls all feature types in a given data store / workspace.

Method	Action	Status code	Formats	Default Format	Parameters
GET	List all feature types in data store `ds`	200	HTML, XML, JSON	HTML	*list*
POST	Create a new feature type, *see note below*	201 with `Location` header	XML, JSON		
PUT		405			
DELETE		405			

Note: When creating a new feature type via `POST`, if no underlying dataset with the specified name exists an attempt will be made to create it. This will work only in cases where the underlying data format supports the creation of new types (such as a database). When creating a feature type in this manner the client should include all attribute information in the feature type representation.

Exceptions

Exception	Status code
GET for a feature type that does not exist	404
PUT that changes name of feature type	403
PUT that changes data store of feature type	403

Parameters

`list` The `list` parameter is used to control the category of feature types that are returned. It can take one of the following values:

- `configured`—Only configured feature types are returned. This is the default value.
- `available`—Only feature types that haven't been configured but are available from the specified data store will be returned.
- `available_with_geom`—Same as `available` but only includes feature types that have a geometry attribute.
- `all`—The union of `configured` and `available`.

`/workspaces/<ws>/datastores/<ds>/featuretypes/<ft>[.<format>]`

Controls a particular feature type in a given data store and workspace.

Method	Action	Status code	Formats	Default Format	Parameters
GET	Return feature type `ft`	200	HTML, XML, JSON	HTML	
POST		405			
PUT	Modify feature type `ft`	200	XML,JSON		*recalculate*
DELETE	Delete feature type `ft`	200			*recurse*

Exceptions

Exception	Status code
GET for a feature type that does not exist	404
PUT that changes name of feature type	403
PUT that changes data store of feature type	403

Parameters

`recurse` The `recurse` parameter recursively deletes all layers referenced by the specified featuretype. Allowed values for this parameter are "true" or "false". The default value is "false". A DELETE request with `recurse=false` will fail if any layers reference the featuretype.

`recalculate` The `recalculate` parameter specifies whether to recalculate any bounding boxes for a feature type. Some properties of feature types are automatically recalculated when necessary. In particular, the native bounding box is recalculated when the projection or projection policy are changed, and the lat/long bounding box is recalculated when the native bounding box is recalculated, or when a new native bounding box is explicitly provided in the request. (The native and lat/long bounding boxes are not automatically recalculated when they are explicitly included in the request.) In addition, the client may

explicitly request a fixed set of fields to calculate, by including a comma-separated list of their names in the `recalculate` parameter. For example:

- `recalculate=` (empty parameter): Do not calculate any fields, regardless of the projection, projection policy, etc. This might be useful to avoid slow recalculation when operating against large datasets.
- `recalculate=nativebbox`: Recalculate the native bounding box, but do not recalculate the lat/long bounding box.
- `recalculate=nativebbox,latlonbbox`: Recalculate both the native bounding box and the lat/long bounding box.

14.1.7 Coverage stores

A `coverage store` contains raster format spatial data.

`/workspaces/<ws>/coveragestores[.<format>]`

Controls all coverage stores in a given workspace.

Method	Action	Status code	Formats	Default Format
GET	List all coverage stores in workspace `ws`	200	HTML, XML, JSON	HTML
POST	Create a new coverage store	201 with `Location` header	XML, JSON	
PUT		405		
DELETE		405		

`/workspaces/<ws>/coveragestores/<cs>[.<format>]`

Controls a particular coverage store in a given workspace.

Method	Action	Status code	Formats	Default Format	Parameters
GET	Return coverage store `cs`	200	HTML, XML, JSON	HTML	
POST		405			
PUT	Modify coverage store `cs`				
DELETE	Delete coverage store `cs`				*recurse*

Exceptions

Exception	Status code
GET for a coverage store that does not exist	404
PUT that changes name of coverage store	403
PUT that changes workspace of coverage store	403
DELETE against a coverage store that contains configured coverage	403

Parameters

`recurse` The `recurse` parameter recursively deletes all layers referenced by the coverage store. Allowed values for this parameter are "true" or "false". The default value is "false".

`/workspaces/<ws>/coveragestores/<cs>/file[.<extension>]`

This end point allows a file containing spatial data to be added (via a POST or PUT) into an existing coverage store, or will create a new coverage store if it doesn't already exist. In case of coverage stores containing multiple coverages (e.g., mosaic of NetCDF files) all the coverages will be configured unless `configure=false` is specified as a parameter.

Method	Action	Status code	Formats	Default Format	Parameters
GET	*Deprecated*. Get the underlying files for the coverage store as a zip file with MIME type `application/zip`.	200			
POST	If the coverage store is a simple one (e.g. GeoTiff) it will return a 405, if the coverage store is a structured one (e.g., mosaic) it will harvest the specified files into it, which in turn will integrate the files into the store. Harvest meaning is store dependent, for mosaic the new files will be added as new granules of the mosaic, and existing files will get their attribute updated, other stores might have a different behavior.	405 if the coverage store is a simple one, 200 if structured and the harvest operation succeded			*recalculate*
PUT	Creates or overwrites the files for coverage store `cs`	200	*See note below*	:set spell spell-lang=en_us	*configure*, *coverageName*
DELETE		405			

Note: A file can be PUT to a coverage store as a standalone or zipped archive file. Standalone files are only suitable for coverage stores that work with a single file such as GeoTIFF store. Coverage stores that work with multiple files, such as the ImageMosaic store, must be sent as a zip archive.

When uploading a standalone file, set the `Content-type` appropriately based on the file type. If you are loading a zip archive, set the `Content-type` to `application/zip`.

Exceptions

Exception	Status code
GET for a data store that does not exist	404
GET for a data store that is not file based	404

Parameters

`extension` The `extension` parameter specifies the type of coverage store. The following extensions are supported:

Extension	Coverage store
geotiff	GeoTIFF
worldimage	Georeferenced image (JPEG, PNG, TIFF)
imagemosaic	Image mosaic

configure The `configure` parameter controls how the coverage store is configured upon file upload. It can take one of the three values:

- `first`—(*Default*) Only setup the first feature type available in the coverage store.
- `none`—Do not configure any feature types.
- `all`—Configure all feature types.

coverageName The `coverageName` parameter specifies the name of the coverage within the coverage store. This parameter is only relevant if the `configure` parameter is not equal to "none". If not specified the resulting coverage will receive the same name as its containing coverage store.

Note: At present a one-to-one relationship exists between a coverage store and a coverage. However, there are plans to support multidimensional coverages, so this parameter may change.

---

recalculate The `recalculate` parameter specifies whether to recalculate any bounding boxes for a coverage. Some properties of coverages are automatically recalculated when necessary. In particular, the native bounding box is recalculated when the projection or projection policy is changed. The lat/long bounding box is recalculated when the native bounding box is recalculated or when a new native bounding box is explicitly provided in the request. (The native and lat/long bounding boxes are not automatically recalculated when they are explicitly included in the request.) In addition, the client may explicitly request a fixed set of fields to calculate by including a comma-separated list of their names in the `recalculate` parameter. For example:

- `recalculate=` (empty parameter)—Do not calculate any fields, regardless of the projection, projection policy, etc. This might be useful to avoid slow recalculation when operating against large datasets.
- `recalculate=nativebbox`—Recalculate the native bounding box, but do not recalculate the lat/long bounding box.
- `recalculate=nativebbox,latlonbbox`—Recalculate both the native bounding box and the lat/long bounding box.

14.1.8 Coverages

A `coverage` is a raster data set which originates from a coverage store.

`/workspaces/<ws>/coveragestores/<cs>/coverages[.<format>]`

Controls all coverages in a given coverage store and workspace.

Method	Action	Status code	Formats	Default Format
GET	List all coverages in coverage store `cs`	200	HTML, XML, JSON	HTML
POST	Create a new coverage	201 with `Location` header	XML, JSON	
PUT		405		
DELETE		405		

/workspaces/<ws>/coveragestores/<cs>/coverages/<c>[.<format>]

Controls a particular coverage in a given coverage store and workspace.

Method	Action	Status code	Formats	Default Format	Parameters
GET	Return coverage `c`	200	HTML, XML, JSON	HTML	
POST		405			
PUT	Modify coverage `c`	200	XML,JSON		
DELETE	Delete coverage `c`	200			*recurse*

Exceptions

Exception	Status code
GET for a coverage that does not exist	404
PUT that changes name of coverage	403
PUT that changes coverage store of coverage	403

Parameters

`recurse` The `recurse` parameter recursively deletes all layers referenced by the specified coverage. Permitted values for this parameter are "true" or "false". The default value is "false".

14.1.9 Structured coverages

Structured coverages are the ones whose content is made of granules, normally associated to attributes, often used to represent time, elevation and other custom dimensions attached to the granules themselves. Image mosaic is an example of a writable structured coverage reader, in which each of the mosaic granules is associated with attributes. NetCDF is an example of a read only one, in which the multidimensional grid contained in the file is exposed as a set of 2D slices, each associated with a different set of variable values.

The following API applies exclusively to structured coverage readers.

/workspaces/<ws>/coveragestores/<cs>/coverages/<coverage>/index[.<format>]

Declares the set of attributes associated to the specified coverage, their name, type and min/max occurrences.

Method	Action	Status code	Formats	Default Format	Parameters
GET	Returns the attributes, their names and their types	200	XML, JSON	XML	
POST		405			
PUT		405			
DELETE		405			

/workspaces/<ws>/coveragestores/<cs>/coverages/<coverage>/index/granules.<format>

Returns the full list of granules, each with its attributes vales and geometry, and allows to selectively remove them

Method	Action	Status code	Formats	Default Format	Parameters
GET	Returns the list of granules and their attributes, either in GML (when XML is used) or GeoJSON (when JSON is used)	200	XML, JSON	XML	*offset*, *limit*, *filter*
POST		405			
PUT		405			
DELETE	Deletes the granules (all, or just the ones selected via the filter parameter)	200			*filter*

Parameters

`offset` The `offset` parameter instructs GeoServer to skip the specified number of first granules when returning the data.

`limit` The `limit` parameter instructs GeoServer to return at most the specified number of granules when returining the data.

`filter` The `filter` parameter is a CQL filter that allows to select which granules will be returned based on their attribute values.

`/workspaces/<ws>/coveragestores/<cs>/coverages/<mosaic>/index/granules/<granuleId>.<format>`

Returns a single granule and allows for its removal.

Method	Action	Status code	Formats	Default Format	Parameters
GET	Returns the specified of granules and its attributes, either in GML (when XML is used) or GeoJSON (when JSON is used)	200	XML, JSON	XML	
POST		405			
PUT		405			
DELETE	Deletes the granule	200			

14.1.10 Styles

A `style` describes how a resource (feature type or coverage) should be symbolized or rendered by the Web Map Service. In GeoServer styles are specified with *SLD*.

`/styles[.<format>]`

Controls all styles.

Method	Action	Status code	Formats	Default Format	Parameters
GET	Return all styles	200	HTML, XML, JSON	HTML	
POST	Create a new style	201 with `Location` header	SLD, XML, JSON *See note below*		*name*
PUT		405			
DELETE		405			*purge*

When executing a POST or PUT request with an SLD style, the `Content-type` header should be set to `application/vnd.ogc.sld+xml`.

Parameters

`name` The `name` parameter specifies the name to be given to the style. This option is most useful when executing a POST request with a style in SLD format, and an appropriate name can be not be inferred from the SLD itself.

`/styles/<s>[.<format>]`

Controls a given style.

Method	Action	Status code	Formats	Default Format
GET	Return style `s`	200	SLD, HTML, XML, JSON	HTML
POST		405		
PUT	Modify style `s`	200	SLD, XML, JSON, *See note above*	
DELETE	Delete style `s`	200		

Exceptions

Exception	Status code
GET for a style that does not exist	404
PUT that changes name of style	403
DELETE against style which is referenced by existing layers	403

Parameters

`purge` The `purge` parameter specifies whether the underlying SLD file for the style should be deleted on disk. Allowable values for this parameter are "true" or "false". When set to "true" the underlying file will be deleted.

`/workspaces/<ws>/styles[.<format>]`

Controls all styles in a given workspace.

Method	Action	Status code	Formats	Default Format	Parameters
GET	Return all styles within workspace `ws`	200	HTML, XML, JSON	HTML	
POST	Create a new style within workspace `ws`	201 with `Location` header	SLD, XML, JSON, *See note above*		*name*
PUT		405			
DELETE		405			*purge*

`/workspaces/<ws>/styles/<s>[.<format>]`

Controls a particular style in a given workspace.

Method	Action	Status code	Formats	Default Format
GET	Return style `s` within workspace `ws`	200	SLD, HTML, XML, JSON	HTML
POST		405		
PUT	Modify style `s` within workspace `ws`	200	SLD, XML, JSON *See note above*	
DELETE	Delete style `s` within workspace `ws`	200		

14.1.11 Layers

A `layer` is a *published* resource (feature type or coverage).

`/layers[.<format>]`

Controls all layers.

Method	Action	Status code	Formats	Default Format
GET	Return all layers	200	HTML, XML, JSON	HTML
POST		405		
PUT		405		
DELETE		405		

`/layers/<l>[.<format>]`

Controls a particular layer.

Method	Action	Status code	Formats	Default Format	Parameters
GET	Return layer `l`	200	HTML, XML, JSON	HTML	
POST		405			
PUT	Modify layer `l`	200	XML,JSON		
DELETE	Delete layer `l`	200			*recurse*

Exceptions

Exception	Status code
GET for a layer that does not exist	404
PUT that changes name of layer	403
PUT that changes resource of layer	403

Parameters

recurse The `recurse` parameter recursively deletes all styles referenced by the specified layer. Allowed values for this parameter are "true" or "false". The default value is "false".

/layers/<l>/styles[.<format>]

Controls all styles in a given layer.

Method	Action	Status code	Formats	Default Format
GET	Return all styles for layer `l`	200	SLD, HTML, XML, JSON	HTML
POST	Add a new style to layer `l`	201, with `Location` header	XML, JSON	
PUT		405		
DELETE		405		

14.1.12 Layer groups

A `layer group` is a grouping of layers and styles that can be accessed as a single layer in a WMS GetMap request. A layer group is sometimes referred to as a "base map".

/layergroups[.<format>]

Controls all layer groups.

Method	Action	Status code	Formats	Default Format
GET	Return all layer groups	200	HTML, XML, JSON	HTML
POST	Add a new layer group	201, with `Location` header	XML,JSON	
PUT		405		
DELETE		405		

/layergroups/<lg>[.<format>]

Controls a particular layer group.

Method	Action	Status code	Formats	Default Format
GET	Return layer group `lg`	200	HTML, XML, JSON	HTML
POST		405		
PUT	Modify layer group `lg`	200	XML,JSON	
DELETE	Delete layer group `lg`	200		

Exceptions

Exception	Status code
GET for a layer group that does not exist	404
POST that specifies layer group with no layers	400
PUT that changes name of layer group	403

/workspaces/<ws>/layergroups[.<format>]

Controls all layer groups in a given workspace.

Method	Action	Status code	Formats	Default Format
GET	Return all layer groups within workspace `ws`	200	HTML, XML, JSON	HTML
POST	Add a new layer group within workspace `ws`	201, with `Location` header	XML,JSON	
PUT		405		
DELETE		405		

/workspaces/<ws>/layergroups/<lg>[.<format>]

Controls a particular layer group in a given workspace.

Method	Action	Status code	Formats	Default Format
GET	Return layer group `lg` within workspace `ws`	200	HTML, XML, JSON	HTML
POST		405		
PUT	Modify layer group `lg` within workspace `ws`	200	XML,JSON	
DELETE	Delete layer group `lg` within workspace `ws`	200		

14.1.13 Fonts

This operation provides the list of `fonts` available in GeoServer. It can be useful to use this operation to verify if a `font` used in a SLD file is available before uploading the SLD.

/fonts[.<format>]

Method	Action	Status code	Formats	Default Format
GET	Return the fonts available in GeoServer	200	XML, JSON	XML
POST		405		
PUT		405		
DELETE		405		

14.1.14 Freemarker templates

Freemarker is a simple yet powerful template engine that GeoServer emplys for user customization of outputs.

It is possible to use the GeoServer REST API to manage Freemarker templates for catalog resources.

/templates/<template>.ftl

This endpoint manages a template that is global to the entire catalog.

Method	Action	Status Code	Formats	Default Format
GET	Return a template	200		
PUT	Insert or update a template	405		
DELETE	Delete a template	405		

Identical operations apply to the following endpoints:

- Workspace templates—`/workspaces/<ws>/templates/<template>.ftl`
- Vector store templates—`/workspaces/<ws>/datastores/<ds>/templates/<template>.ftl`
- Feature type templates—`/workspaces/<ws>/datastores/<ds>/featuretypes/<f>/templates/<templat`
- Raster store templates—`/workspaces/<ws>/coveragestores/<cs>/templates/<template>.ftl`
- Coverage templates—`/workspaces/<ws>/coveragestores/<cs>/coverages/<c>/templates/<template`

`/templates[.<format>]`

This endpoint manages all global templates.

Method	Action	Status Code	Formats	Default Format
GET	Return templates	200	HTML, XML, JSON	HTML

Identical operations apply to the following endpoints:

- Workspace templates—`/workspaces/<ws>/templates[.<format>]`
- Vector store templates—`/workspaces/<ws>/datastores/<ds>/templates[.<format>]`
- Feature type templates—`/workspaces/<ws>/datastores/<ds>/featuretypes/<f>/templates[.<format`
- Raster store templates—`/workspaces/<ws>/coveragestores/<cs>/templates[.<format>]`
- Coverage templates—`/workspaces/<ws>/coveragestores/<cs>/coverages/<c>/templates[.<format>`

14.1.15 OWS Services

GeoServer includes several types of OGC services like WCS, WFS and WMS, commonly referred to as "OWS" services. These services can be global for the whole GeoServer instance or local to a particular workspace. In this last case, they are called *virtual services*.

`/services/wcs/settings[.<format>]`

Controls Web Coverage Service settings.

Method	Action	Status code	Formats	Default Format
GET	Return global WCS settings	200	XML, JSON	HTML
POST		405		
PUT	Modify global WCS settings	200		
DELETE		405		

`/services/wcs/workspaces/<ws>/settings[.<format>]`

Controls Web Coverage Service settings for a given workspace.

Method	Action	Status code	Formats	Default Format
GET	Return WCS settings for workspace `ws`	200	HTML, XML, JSON	HTML
POST		405		
PUT	Create or modify WCS settings for workspace `ws`	200	XML,JSON	
DELETE	Delete WCS settings for workspace `ws`	200		

/services/wfs/settings[.<format>]

Controls Web Feature Service settings.

Method	Action	Status code	Formats	Default Format
GET	Return global WFS settings	200	HTML, XML, JSON	HTML
POST		405		
PUT	Modify global WFS settings	200	XML,JSON	
DELETE		405		

/services/wfs/workspaces/<ws>/settings[.<format>]

Controls Web Feature Service settings for a given workspace.

Method	Action	Status code	Formats	Default Format
GET	Return WFS settings for workspace `ws`	200	HTML, XML, JSON	HTML
POST		405		
PUT	Modify WFS settings for workspace `ws`	200	XML,JSON	
DELETE	Delete WFS settings for workspace `ws`	200		

/services/wms/settings[.<format>]

Controls Web Map Service settings.

Method	Action	Status code	Formats	Default Format
GET	Return global WMS settings	200	HTML, XML, JSON	HTML
POST		405		
PUT	Modify global WMS settings	200	XML,JSON	
DELETE		405		

/services/wms/workspaces/<ws>/settings[.<format>]

Controls Web Map Service settings for a given workspace.

Method	Action	Status code	Formats	Default Format
GET	Return WMS settings for workspace `ws`	200	HTML, XML, JSON	HTML
POST		405		
PUT	Modify WMS settings for workspace `ws`	200	XML,JSON	
DELETE	Delete WMS settings for workspace `ws`	200		

14.1.16 Reloading configuration

Reloads the GeoServer catalog and configuration from disk. This operation is used in cases where an external tool has modified the on-disk configuration. This operation will also force GeoServer to drop any internal caches and reconnect to all data stores.

/reload

Method	Action	Status code	Formats	Default Format
GET		405		
POST	Reload the configuration from disk	200		
PUT	Reload the configuration from disk	200		
DELETE		405		

14.1.17 Resource reset

Resets all store, raster, and schema caches. This operation is used to force GeoServer to drop all caches and store connections and reconnect to each of them the next time they are needed by a request. This is useful in case the stores themselves cache some information about the data structures they manage that may have changed in the meantime.

/reset

Method	Action	Status code	Formats	Default Format
GET		405		
POST	Reload the configuration from disk	200		
PUT	Reload the configuration from disk	200		
DELETE		405		

14.1.18 Manifests

GeoServer provides a REST service to expose a listing of all loaded JARs and resources on the running instance. This is useful for bug reports and to keep track of extensions deployed into the application. There are two endpoints for accessing this information:

- `about/manifest`—Retrieves details on all loaded JARs
- `about/version`—Retrieves details for the high-level components: GeoSever, GeoTools, and GeoWebCache

/about/manifest[.<format>]

This endpoint retrieves details on all loaded JARs.

All the GeoServer manifest JARs are marked with the property `GeoServerModule` and classified by type, so you can use filtering capabilities to search for a set of manifests using regular expressions (see the *manifest* parameter) or a type category (see the *key* and *value* parameter).

The available types are `core`, `extension`, or `community`. To filter modules by a particular type, append a request with `key=GeoServerModule&value=<type>`

Method	Action	Status Code	Formats	Default Format	Parameters
GET	List all manifests into the classpath	200	HTML, XML, JSON	HTML	*manifest*, *key*, *value*
POST		405			
PUT		405			
DELETE		405			

Usage

The model is very simple and is shared between the version and the resource requests to parse both requests.

```
<about>
  <resource name="{NAME}">
    <{KEY}>{VALUE}</{KEY}>
    ...
  </resource>
  ...
</about>
```

You can customize the results adding a properties file called `manifest.properties` into the root of the data directory. Below is the default implementation that is used when no custom properties file is present:

```
resourceNameRegex=.+/(.*).(jar|war)
resourceAttributeExclusions=Import-Package,Export-Package,Class-Path,Require-Bundle
versionAttributeInclusions=Project-Version:Version,Build-Timestamp,Git-Revision,
  Specification-Version:Version,Implementation-Version:Git-Revision
```

where:

- `resourceNameRegex`—Group(1) will be used to match the attribute name of the resource.
- `resourceAttributeExclusions`—Comma-separated list of properties to exclude (blacklist), used to exclude parameters that are too verbose such that the resource properties list is left open. Users can add their JARs (with custom properties) having the complete list of properties.
- `versionAttributeInclusions`—Comma-separated list of properties to include (whitelist). Also supports renaming properties (using `key:replace`) which is used to align the output of the `versions` request to the output of the web page. The model uses a map to store attributes, so the last attribute found in the manifest file will be used.

`manifest` The `manifest` parameter is used to filter over resulting resource (manifest) names attribute using Java regular expressions.

`key` The `key` parameter is used to filter over resulting resource (manifest) properties name. It can be combined with the `value` parameter.

`value` The `value` parameter is used to filter over resulting resource (manifest) properties value. It can be combined with the `key` parameter.

`/about/version[.<format>]`

This endpoint shows only the details for the high-level components: GeoServer, GeoTools, and GeoWebCache.

Method	Action	Status Code	Formats	Default Format	Parameters
GET	List GeoServer, GeoWebCache and GeoTools manifests	200	HTML, XML, JSON	HTML	*manifest*, *key*, *value*
POST		405			
PUT		405			
DELETE		405			

14.2 REST configuration examples

This section contains a number of examples which illustrate various uses of the *REST configuration API*. The examples are grouped by the language or environment used.

14.2.1 cURL

The examples in this section use cURL, a command line tool for executing HTTP requests and transferring files, to generate requests to GeoServer's REST interface. Although the examples are based on cURL, they could be adapted for any HTTP-capable tool or library.

Adding a new workspace

The following creates a new workspace named "acme" with a POST request:

Note: Each code block below contains a single command that may be extended over multiple lines.

```
curl -v -u admin:geoserver -XPOST -H "Content-type: text/xml"
  -d "<workspace><name>acme</name></workspace>"
  http://localhost:8080/geoserver/rest/workspaces
```

If executed correctly, the response should contain the following:

```
< HTTP/1.1 201 Created
...
< Location: http://localhost:8080/geoserver/rest/workspaces/acme
```

Note the `Location` response header, which specifies the location (URI) of the newly created workspace.

The workspace information can be retrieved as XML with a GET request:

```
curl -v -u admin:geoserver -XGET -H "Accept: text/xml"
  http://localhost:8080/geoserver/rest/workspaces/acme
```

The response should look like this:

```
<workspace>
  <name>acme</name>
  <dataStores>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate"
     href="http://localhost:8080/geoserver/rest/workspaces/acme/datastores.xml"
     type="application/xml"/>
  </dataStores>
  <coverageStores>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate"
     href="http://localhost:8080/geoserver/rest/workspaces/acme/coveragestores.xml"
     type="application/xml"/>
  </coverageStores>
  <wmsStores>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate"
     href="http://localhost:8080/geoserver/rest/workspaces/acme/wmsstores.xml"
     type="application/xml"/>
  </wmsStores>
</workspace>
```

This shows that the workspace can contain "dataStores" (for *vector data*), "coverageStores" (for *raster data*), and "wmsStores" (for *cascaded WMS servers*).

Note: The Accept header is optional. The following request omits the Accept header, but will return the same response as above.

```
curl -v -u admin:geoserver -XGET http://localhost:8080/geoserver/rest/workspaces/acme.xml
```

Uploading a shapefile

In this example a new store will be created by uploading a shapefile.

The following request uploads a zipped shapefile named roads.zip and creates a new store named roads.

Note: Each code block below contains a single command that may be extended over multiple lines.

```
curl -v -u admin:geoserver -XPUT -H "Content-type: application/zip"
  --data-binary @roads.zip
  http://localhost:8080/geoserver/rest/workspaces/acme/datastores/roads/file.shp
```

The roads identifier in the URI refers to the name of the store to be created. To create a store named somethingelse, the URI would be http://localhost:8080/geoserver/rest/workspaces/acme/datastores/somethingelse/file.shp

If executed correctly, the response should contain the following:

```
< HTTP/1.1 201 Created
```

The store information can be retrieved as XML with a GET request:

```
curl -v -u admin:geoserver -XGET
  http://localhost:8080/geoserver/rest/workspaces/acme/datastores/roads.xml
```

The response should look like this:

```
<dataStore>
  <name>roads</name>
  <type>Shapefile</type>
  <enabled>true</enabled>
  <workspace>
    <name>acme</name>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate"
     href="http://localhost:8080/geoserver/rest/workspaces/acme.xml" type="application/xml"/>
  </workspace>
  <connectionParameters>
    <entry key="url">file:/C:/path/to/data_dir/data/acme/roads/</entry>
    <entry key="namespace">http://acme</entry>
  </connectionParameters>
  <__default>false</__default>
  <featureTypes>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate"
     href="http://localhost:8080/geoserver/rest/workspaces/acme/datastores/roads/featuretypes.xml"
     type="application/xml"/>
  </featureTypes>
</dataStore>
```

By default when a shapefile is uploaded, a feature type is automatically created. The feature type information can be retrieved as XML with a GET request:

```
curl -v -u admin:geoserver -XGET
  http://localhost:8080/geoserver/rest/workspaces/acme/datastores/roads/featuretypes/roads.xml
```

If executed correctly, the response will be:

```
<featureType>
  <name>roads</name>
  <nativeName>roads</nativeName>
  <namespace>
    <name>acme</name>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate"
     href="http://localhost:8080/geoserver/rest/namespaces/acme.xml" type="application/xml"/>
  </namespace>
  ...
</featureType>
```

The remainder of the response consists of layer metadata and configuration information.

Adding an existing shapefile

In the previous example a shapefile was uploaded directly to GeoServer by sending a zip file in the body of a PUT request. This example shows how to publish a shapefile that already exists on the server.

Consider a directory on the server `/data/shapefiles/rivers` that contains the shapefile `rivers.shp`. The following adds a new store for the shapefile:

Note: Each code block below contains a single command that may be extended over multiple lines.

```
curl -v -u admin:geoserver -XPUT -H "Content-type: text/plain"
  -d "file:///data/shapefiles/rivers/rivers.shp"
  http://localhost:8080/geoserver/rest/workspaces/acme/datastores/rivers/external.shp
```

The `external.shp` part of the request URI indicates that the file is coming from outside the catalog.

If executed correctly, the response should contain the following:

```
< HTTP/1.1 201 Created
```

The shapefile will be added to the existing store and published as a layer.

To verify the contents of the store, execute a GET request. Since the XML response only provides details about the store itself without showing its contents, execute a GET request for HTML:

```
curl -v -u admin:geoserver -XGET
  http://localhost:8080/geoserver/rest/workspaces/acme/datastores/rivers.html
```

Adding a directory of existing shapefiles

This example shows how to load and create a store that contains a number of shapefiles, all with a single operation. This example is very similar to the example above of adding a single shapefile.

Consider a directory on the server `/data/shapefiles` that contains multiple shapefiles. The following adds a new store for the directory.

Note: Each code block below contains a single command that may be extended over multiple lines.

```
curl -v -u admin:geoserver -XPUT -H "Content-type: text/plain"
  -d "file:///data/shapefiles/"
  "http://localhost:8080/geoserver/rest/workspaces/acme/datastores/shapefiles/external.shp?configure=
```

Note the `configure=all` query string parameter, which sets each shapefile in the directory to be loaded and published.

If executed correctly, the response should contain the following:

```
< HTTP/1.1 201 Created
```

To verify the contents of the store, execute a GET request. Since the XML response only provides details about the store itself without showing its contents, execute a GET request for HTML:

```
curl -v -u admin:geoserver -XGET
http://localhost:8080/geoserver/rest/workspaces/acme/datastores/shapefiles.html
```

Creating a layer style

This example will create a new style on the server and populate it the contents of a local SLD file.

The following creates a new style named `roads_style`:

Note: Each code block below contains a single command that may be extended over multiple lines.

```
curl -v -u admin:geoserver -XPOST -H "Content-type: text/xml"
  -d "<style><name>roads_style</name><filename>roads.sld</filename></style>"
  http://localhost:8080/geoserver/rest/styles
```

If executed correctly, the response should contain the following:

```
< HTTP/1.1 201 Created
```

This request uploads a file called `roads.sld` file and populates the `roads_style` with its contents:

```
curl -v -u admin:geoserver -XPUT -H "Content-type: application/vnd.ogc.sld+xml"
  -d @roads.sld http://localhost:8080/geoserver/rest/styles/roads_style
```

If executed correctly, the response should contain the following:

```
< HTTP/1.1 200 OK
```

The SLD itself can be downloaded through a a GET request:

```
curl -v -u admin:geoserver -XGET
  http://localhost:8080/geoserver/rest/styles/roads_style.sld
```

Changing a layer style

This example will alter a layer style. Prior to making any changes, it is helpful to view the existing configuration for a given layer.

Note: Each code block below contains a single command that may be extended over multiple lines.

The following retrieves the "acme:roads" layer information as XML:

```
curl -v -u admin:geoserver -XGET "http://localhost:8080/geoserver/rest/layers/acme:roads.xml"
```

The response in this case would be:

```
<layer>
  <name>roads</name>
  <type>VECTOR</type>
  <defaultStyle>
    <name>line</name>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate"
     href="http://localhost:8080/geoserver/rest/styles/line.xml" type="application/xml"/>
  </defaultStyle>
  <resource class="featureType">
    <name>roads</name>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate"
     href="http://localhost:8080/geoserver/rest/workpaces/acme/datastores/roads/featuretypes/roads.xm
     type="application/xml"/>
  </resource>
  <enabled>true</enabled>
  <attribution>
    <logoWidth>0</logoWidth>
    <logoHeight>0</logoHeight>
  </attribution>
</layer>
```

When the layer is created, GeoServer assigns a default style to the layer that matches the geometry of the layer. In this case a style named `line` is assigned to the layer. This style can viewed with a WMS request:

```
http://localhost:8080/geoserver/wms/reflect?layers=acme:roads
```

In this next example a new style will be created called `roads_style` and assigned to the "acme:roads" layer:

```
curl -v -u admin:geoserver -XPUT -H "Content-type: text/xml"
  -d "<layer><defaultStyle><name>roads_style</name></defaultStyle></layer>"
  http://localhost:8080/geoserver/rest/layers/acme:roads
```

If executed correctly, the response should contain the following:

```
< HTTP/1.1 200 OK
```

The new style can be viewed with the same WMS request as above:

```
http://localhost:8080/geoserver/wms/reflect?layers=acme:roads
```

Note that if you want to upload the style in a workspace (ie, not making it a global style), and then assign this style to a layer in that workspace, you need first to create the style in the given workspace:

```
curl -u admin:geoserver -XPOST -H 'Content-type: text/xml' \
  -d '<style><name>roads_style</name><filename>roads.sld</filename></style>'
  http://localhost:8080/geoserver/rest/workspaces/acme/styles
```

Upload the file within the workspace:

```
curl -u admin:geoserver -XPUT -H 'Content-type: application/vnd.ogc.sld+xml' \
  -d @roads.sld http://localhost:8080/geoserver/rest/workspaces/acme/styles/roads_style
```

And finally apply that style to the layer. Note the use of the `<workspace>` tag in the XML:

```
curl -u admin:geoserver -XPUT -H 'Content-type: text/xml' \
  -d '<layer><defaultStyle><name>roads_style</name><workspace>acme</workspace></defaultStyle></layer>
  http://localhost:8080/geoserver/rest/layers/acme:roads
```

Adding a PostGIS database

In this example a PostGIS database named `nyc` will be added as a new store. This section assumes that a PostGIS database named `nyc` is present on the local system and is accessible by the user `bob`.

Create a new text file and add the following content to it. This will represent the new store. Save the file as `nycDataStore.xml`.

```
<dataStore>
  <name>nyc</name>
  <connectionParameters>
    <host>localhost</host>
    <port>5432</port>
    <database>nyc</database>
    <user>bob</user>
    <passwd>postgres</passwd>
    <dbtype>postgis</dbtype>
  </connectionParameters>
</dataStore>
```

The following will add the new PostGIS store to the GeoServer catalog:

Note: Each code block below contains a single command that may be extended over multiple lines.

```
curl -v -u admin:geoserver -XPOST -T nycDataStore.xml -H "Content-type: text/xml"
  http://localhost:8080/geoserver/rest/workspaces/acme/datastores
```

If executed correctly, the response should contain the following:

```
< HTTP/1.1 200 OK
```

The store information can be retrieved as XML with a GET request:

```
curl -v -u admin:geoserver -XGET http://localhost:8080/geoserver/rest/workspaces/acme/datastores/nyc
```

The response should look like the following:

```
<dataStore>
  <name>nyc</name>
  <type>PostGIS</type>
  <enabled>true</enabled>
  <workspace>
    <name>acme</name>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate"
     href="http://localhost:8080/geoserver/rest/workspaces/acme.xml" type="application/xml"/>
  </workspace>
  <connectionParameters>
    <entry key="port">5432</entry>
    <entry key="dbtype">postgis</entry>
    <entry key="host">localhost</entry>
    <entry key="user">bob</entry>
    <entry key="database">nyc</entry>
    <entry key="namespace">http://acme</entry>
  </connectionParameters>
```

```
  <__default>false</__default>
  <featureTypes>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate"
     href="http://localhost:8080/geoserver/rest/workspaces/acme/datastores/nyc/featuretypes.xml"
     type="application/xml"/>
  </featureTypes>
</dataStore>
```

Adding a PostGIS table

In this example a table from the PostGIS database created in the previous example will be added as a featuretypes. This example assumes the table has already been created.

The following adds the table `buildings` as a new feature type:

Note: Each code block below contains a single command that may be extended over multiple lines.

```
curl -v -u admin:geoserver -XPOST -H "Content-type: text/xml"
  -d "<featureType><name>buildings</name></featureType>"
  http://localhost:8080/geoserver/rest/workspaces/acme/datastores/nyc/featuretypes
```

The featuretype information can be retrieved as XML with a GET request:

```
curl -v -u admin:geoserver -XGET
  http://localhost:8080/geoserver/rest/workspaces/acme/datastores/nyc/featuretypes/buildings.xml
```

This layer can viewed with a WMS GetMap request:

```
http://localhost:8080/geoserver/wms/reflect?layers=acme:buildings
```

Creating a PostGIS table

In the previous example, a new feature type was added based on a PostGIS table that already existed in the database. The following example will not only create a new feature type in GeoServer, but will also create the PostGIS table itself.

Create a new text file and add the following content to it. This will represent the definition of the new feature type and table. Save the file as `annotations.xml`.

```
<featureType>
  <name>annotations</name>
  <nativeName>annotations</nativeName>
  <title>Annotations</title>
  <srs>EPSG:4326</srs>
  <attributes>
    <attribute>
      <name>the_geom</name>
      <binding>com.vividsolutions.jts.geom.Point</binding>
    </attribute>
    <attribute>
      <name>description</name>
      <binding>java.lang.String</binding>
    </attribute>
    <attribute>
      <name>timestamp</name>
      <binding>java.util.Date</binding>
```

```
    </attribute>
  </attributes>
</featureType>
```

This request will perform the feature type creation and add the new table:

Note: Each code block below contains a single command that may be extended over multiple lines.

```
curl -v -u admin:geoserver -XPOST -T annotations.xml -H "Content-type: text/xml"
  http://localhost:8080/geoserver/rest/workspaces/acme/datastores/nyc/featuretypes
```

The result is a new, empty table named "annotations" in the "nyc" database, fully configured as a feature type.

The featuretype information can be retrieved as XML with a GET request:

```
curl -v -u admin:geoserver -XGET
  http://localhost:8080/geoserver/rest/workspaces/acme/datastores/nyc/featuretypes/annotations.xml
```

Creating a layer group

In this example a layer group will be created, based on layers that already exist on the server.

Create a new text file and add the following content to it. This file will represent the definition of the new layer group. Save the file as `nycLayerGroup.xml`.

```
<layerGroup>
  <name>nyc</name>
  <layers>
    <layer>roads</layer>
    <layer>parks</layer>
    <layer>buildings</layer>
  </layers>
  <styles>
    <style>roads_style</style>
    <style>polygon</style>
    <style>polygon</style>
  </styles>
</layerGroup>
```

The following request creates the new layer group:

Note: Each code block below contains a single command that may be extended over multiple lines.

```
curl -v -u admin:geoserver -XPOST -d @nycLayerGroup.xml -H "Content-type: text/xml"
  http://localhost:8080/geoserver/rest/layergroups
```

Note: The argument `-d@filename.xml` in this example is used to send a file as the body of an HTTP request with a POST method. The argument `-T filename.xml` used in the previous example was used to send a file as the body of an HTTP request with a PUT method.

This layer group can be viewed with a WMS GetMap request:

```
http://localhost:8080/geoserver/wms/reflect?layers=nyc
```

Retrieving component versions

This example shows how to retrieve the versions of the main components: GeoServer, GeoTools, and GeoWebCache:

Note: The code block below contains a single command that is extended over multiple lines.

```
curl -v -u admin:geoserver -XGET -H "Accept: text/xml"
  http://localhost:8080/geoserver/rest/about/version.xml
```

The response will look something like this:

```
<about>
 <resource name="GeoServer">
  <Build-Timestamp>11-Dec-2012 17:55</Build-Timestamp>
  <Git-Revision>e66f8da85521f73d0fd00b71331069a5f49f7865</Git-Revision>
  <Version>2.3-SNAPSHOT</Version>
 </resource>
 <resource name="GeoTools">
  <Build-Timestamp>04-Dec-2012 02:31</Build-Timestamp>
  <Git-Revision>380a2b8545ee9221f1f2d38a4f10ef77a23bccae</Git-Revision>
  <Version>9-SNAPSHOT</Version>
 </resource>
 <resource name="GeoWebCache">
  <Git-Revision>2a534f91f6b99e5120a9eaa5db62df771dd01688</Git-Revision>
  <Version>1.3-SNAPSHOT</Version>
 </resource>
</about>
```

Retrieving manifests

This collection of examples shows how to retrieve the full manifest and subsets of the manifest as known to the ClassLoader.

Note: The code block below contains a single command that is extended over multiple lines.

```
curl -v -u admin:geoserver -XGET -H "Accept: text/xml"
  http://localhost:8080/geoserver/rest/about/manifest.xml
```

The result will be a very long list of manifest information. While this can be useful, it is often desirable to filter this list.

Filtering over resource name

It is possible to filter over resource names using regular expressions. This example will retrieve only resources where the `name` attribute matches `gwc-.*`:

Note: The code block below contains a single command that is extended over multiple lines.

```
curl -v -u admin:geoserver -XGET -H "Accept: text/xml"
  http://localhost:8080/geoserver/rest/about/manifest.xml?manifest=gwc-.*
```

The result will look something like this (edited for brevity):

```
<about>
  <resource name="gwc-2.3.0">
    ...
  </resource>
  <resource name="gwc-core-1.4.0">
    ...
  </resource>
  <resource name="gwc-diskquota-core-1.4.0">
    ...
  </resource>
  <resource name="gwc-diskquota-jdbc-1.4.0">
    ...
  </resource>
  <resource name="gwc-georss-1.4.0">
    ...
  </resource>
  <resource name="gwc-gmaps-1.4.0">
    ...
  </resource>
  <resource name="gwc-kml-1.4.0">
    ...
  </resource>
  <resource name="gwc-rest-1.4.0">
    ...
  </resource>
  <resource name="gwc-tms-1.4.0">
    ...
  </resource>
  <resource name="gwc-ve-1.4.0">
    ...
  </resource>
  <resource name="gwc-wms-1.4.0">
    ...
  </resource>
  <resource name="gwc-wmts-1.4.0">
    ...
  </resource>
</about>
```

Filtering over resource properties

Filtering is also available over resulting resource properties. This example will retrieve only resources with a property equal to `GeoServerModule`.

Note: The code blocks below contain a single command that is extended over multiple lines.

```
curl -u admin:geoserver -XGET -H "Accept: text/xml"
  http://localhost:8080/geoserver/rest/about/manifest.xml?key=GeoServerModule
```

The result will look something like this (edited for brevity):

```
<about>
 <resource name="control-flow-2.3.0">
  <GeoServerModule>extension</GeoServerModule>
  ...
 </resource>
```

```
 ...
 <resource name="wms-2.3.0">
  <GeoServerModule>core</GeoServerModule>
  ...
 </resource>
</about>
```

It is also possible to filter against both property and value. To retrieve only resources where a property named `GeoServerModule` has a value equal to `extension`, append the above request with `&value=extension`:

```
curl -u admin:geoserver -XGET -H "Accept: text/xml"
  http://localhost:8080/geoserver/rest/about/manifest.xml?key=GeoServerModule&value=extension
```

Uploading and modifying a image mosaic

The following command uploads a zip file containing the definition of a mosaic (along with at least one granule of the mosaic to initialize the resolutions, overviews and the like) and will configure all the coverages in it as new layers.

Note: The code blocks below contain a single command that is extended over multiple lines.

```
curl -u admin:geoserver -XPUT H "Contenttype:application/zip"--data-binary @polyphemus.zip
   http://localhost:8080/geoserver/rest/workspaces/topp/coveragestores/polyphemus/file.imagemosaic
```

The following instead instructs the mosaic to harvest (or re-harvest) a single file into the mosaic, collecting its properties and updating the mosaic index:

```
curl -v -u admin:geoserver -XPOST -H "Content-type: text/plain" -d "file:///path/to/the/file/polypher
   "http://localhost:8080/geoserver/rest/workspaces/topp/coveragestores/poly-incremental/external.ima
```

Harvesting can also be directed towards a whole directory, as follows:

```
curl -v -u admin:geoserver -XPOST -H "Content-type: text/plain" -d "file:///path/to/the/mosaic/folde
    "http://localhost:8080/geoserver/rest/workspaces/topp/coveragestores/poly-incremental/external.i
```

The image mosaic index structure can be retrieved using something like:

```
curl -v -u admin:geoserver -XGET "http://localhost:8080/geoserver/rest/workspaces/topp/coveragestores
```

which will result in the following:

```
   <Schema>
  <attributes>
    <Attribute>
      <name>the_geom</name>
      <minOccurs>0</minOccurs>
      <maxOccurs>1</maxOccurs>
      <nillable>true</nillable>
      <binding>com.vividsolutions.jts.geom.Polygon</binding>
    </Attribute>
    <Attribute>
      <name>location</name>
      <minOccurs>0</minOccurs>
      <maxOccurs>1</maxOccurs>
      <nillable>true</nillable>
      <binding>java.lang.String</binding>
    </Attribute>
```

```
    <Attribute>
      <name>imageindex</name>
      <minOccurs>0</minOccurs>
      <maxOccurs>1</maxOccurs>
      <nillable>true</nillable>
      <binding>java.lang.Integer</binding>
    </Attribute>
    <Attribute>
      <name>time</name>
      <minOccurs>0</minOccurs>
      <maxOccurs>1</maxOccurs>
      <nillable>true</nillable>
      <binding>java.sql.Timestamp</binding>
    </Attribute>
    <Attribute>
      <name>elevation</name>
      <minOccurs>0</minOccurs>
      <maxOccurs>1</maxOccurs>
      <nillable>true</nillable>
      <binding>java.lang.Double</binding>
    </Attribute>
    <Attribute>
      <name>fileDate</name>
      <minOccurs>0</minOccurs>
      <maxOccurs>1</maxOccurs>
      <nillable>true</nillable>
      <binding>java.sql.Timestamp</binding>
    </Attribute>
    <Attribute>
      <name>updated</name>
      <minOccurs>0</minOccurs>
      <maxOccurs>1</maxOccurs>
      <nillable>true</nillable>
      <binding>java.sql.Timestamp</binding>
    </Attribute>
  </attributes>
  <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" href="http://localhost:8080/geo
</Schema>
```

Listing the existing granules can be performed as follows:

```
curl -v -u admin:geoserver -XGET "http://localhost:8080/geoserver/rest/workspaces/topp/coveragestores
```

This will result in a GML description of the granules, as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<wfs:FeatureCollection xmlns:gf="http://www.geoserver.org/rest/granules" xmlns:ogc="http://www.openg
  <gml:boundedBy>
    <gml:Box srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
      <gml:coord>
        <gml:X>5.0</gml:X>
        <gml:Y>45.0</gml:Y>
      </gml:coord>
      <gml:coord>
        <gml:X>14.875</gml:X>
        <gml:Y>50.9375</gml:Y>
      </gml:coord>
    </gml:Box>
  </gml:boundedBy>
```

```
  <gml:featureMember>
    <gf:NO2 fid="NO2.1">
      <gf:the_geom>
        <gml:Polygon>
          <gml:outerBoundaryIs>
            <gml:LinearRing>
              <gml:coordinates>5.0,45.0 5.0,50.9375 14.875,50.9375 14.875,45.0 5.0,45.0</gml:coordina
            </gml:LinearRing>
          </gml:outerBoundaryIs>
        </gml:Polygon>
      </gf:the_geom>
      <gf:location>polyphemus_20130301.nc</gf:location>
      <gf:imageindex>336</gf:imageindex>
      <gf:time>2013-03-01T00:00:00Z</gf:time>
      <gf:elevation>10.0</gf:elevation>
      <gf:fileDate>2013-03-01T00:00:00Z</gf:fileDate>
      <gf:updated>2013-04-11T10:54:31Z</gf:updated>
    </gf:NO2>
  </gml:featureMember>
  <gml:featureMember>
    <gf:NO2 fid="NO2.2">
      <gf:the_geom>
        <gml:Polygon>
          <gml:outerBoundaryIs>
            <gml:LinearRing>
              <gml:coordinates>5.0,45.0 5.0,50.9375 14.875,50.9375 14.875,45.0 5.0,45.0</gml:coordina
            </gml:LinearRing>
          </gml:outerBoundaryIs>
        </gml:Polygon>
      </gf:the_geom>
      <gf:location>polyphemus_20130301.nc</gf:location>
      <gf:imageindex>337</gf:imageindex>
      <gf:time>2013-03-01T00:00:00Z</gf:time>
      <gf:elevation>35.0</gf:elevation>
      <gf:fileDate>2013-03-01T00:00:00Z</gf:fileDate>
      <gf:updated>2013-04-11T10:54:31Z</gf:updated>
    </gf:NO2>
  </gml:featureMember>
</wfs:FeatureCollection>
```

Removing all the granules originating from a particular file (a NetCDF file can contain many) can be done as follows:

```
curl -v -u admin:geoserver -XDELETE "http://localhost:8080/geoserver/rest/workspaces/topp/coveragesto
```

Creating an empty mosaic and harvest granules

The following command uploads a zip file containing the definition of a mosaic (no granules in this case)

Note: The code blocks below contain a single command that is extended over multiple lines.

Note: the indexer file should contain a CanBeEmpty=true property

Note: The configure=none parameter allows for future configuration after harvesting

```
curl -u admin:geoserver -XPUT -H "Content-type:application/zip" --data-binary @empty.zip
   http://localhost:8080/geoserver/rest/workspaces/topp/coveragestores/empty/file.imagemosaic?configu
```

The following instead instructs the mosaic to harvest a single file into the mosaic, collecting its properties and updating the mosaic index:

```
curl -v -u admin:geoserver -XPOST -H "Content-type: text/plain" -d "file:///path/to/the/file/polypher
   "http://localhost:8080/geoserver/rest/workspaces/topp/coveragestores/empty/external.imagemosaic"
```

Once done you can get the list of coverages/granules available on that store.

```
curl -v -u admin:geoserver -XGET
    "http://localhost:8080/geoserver/rest/workspaces/topp/coveragestores/empty/coverages.xml?list=all
```

which will result in the following:

```
<list>
  <coverageName>NO2</coverageName>
  <coverageName>O3</coverageName>
  <coverageName>V</coverageName>
</list>
```

Next step is configuring ONCE for coverage (as an instance NO2), an available coverage.

```
curl -v -u admin:geoserver -XPOST -H "Content-type: text/xm" -d @"/path/to/coverageconfig.xml" "http
```

Where coverageconfig.xml may look like this

```
<coverage>
  <name>NO2</name>
</coverage>
```

Note: When specifying only the coverage name, the coverage will be automatically configured

14.2.2 PHP

The examples in this section use the server-side scripting language PHP, a popular language for dynamic webpages. PHP has cURL functions , as well as XML functions, making it a convenient method for performing batch processing through the Geoserver REST interface. The following scripts execute single requests, but can be easily modified with looping structures to perform batch processing.

POST with PHP/cURL

The following script attempts to add a new workspace.

```
<?php
    // Open log file
    $logfh = fopen("GeoserverPHP.log", 'w') or die("can't open log file");

    // Initiate cURL session
    $service = "http://localhost:8080/geoserver/"; // replace with your URL
    $request = "rest/workspaces"; // to add a new workspace
    $url = $service . $request;
    $ch = curl_init($url);

    // Optional settings for debugging
```

```
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, true); //option to return string
    curl_setopt($ch, CURLOPT_VERBOSE, true);
    curl_setopt($ch, CURLOPT_STDERR, $logfh); // logs curl messages

    //Required POST request settings
    curl_setopt($ch, CURLOPT_POST, True);
    $passwordStr = "admin:geoserver"; // replace with your username:password
    curl_setopt($ch, CURLOPT_USERPWD, $passwordStr);

    //POST data
    curl_setopt($ch, CURLOPT_HTTPHEADER,
              array("Content-type: application/xml"));
    $xmlStr = "<workspace><name>test_ws</name></workspace>";
    curl_setopt($ch, CURLOPT_POSTFIELDS, $xmlStr);

    //POST return code
    $successCode = 201;

    $buffer = curl_exec($ch); // Execute the curl request

    // Check for errors and process results
    $info = curl_getinfo($ch);
    if ($info['http_code'] != $successCode) {
      $msgStr = "# Unsuccessful cURL request to ";
      $msgStr .= $url." [". $info['http_code']. "]\n";
      fwrite($logfh, $msgStr);
    } else {
      $msgStr = "# Successful cURL request to ".$url."\n";
      fwrite($logfh, $msgStr);
    }
    fwrite($logfh, $buffer."\n");

    curl_close($ch); // free resources if curl handle will not be reused
    fclose($logfh);  // close logfile

?>
```

The logfile should look something like:

```
* About to connect() to www.example.com port 80 (#0)
*   Trying 123.456.78.90... * connected
* Connected to www.example.com (123.456.78.90) port 80 (#0)
* Server auth using Basic with user 'admin'
> POST /geoserver/rest/workspaces HTTP/1.1
Authorization: Basic sDsdfjkLDFOIedlsdkfj
Host: www.example.com
Accept: */*
Content-type: application/xml
Content-Length: 43

< HTTP/1.1 201 Created
< Date: Fri, 21 May 2010 15:44:47 GMT
< Server: Apache-Coyote/1.1
< Location: http://www.example.com/geoserver/rest/workspaces/test_ws
< Content-Length: 0
< Content-Type: text/plain
<
* Connection #0 to host www.example.com left intact
# Successful cURL request to http://www.example.com/geoserver/rest/workspaces
```

```
* Closing connection #0
```

If the cURL request fails, a code other than 201 will be returned. Here are some possible values:

Code	Meaning
0	Couldn't resolve host; possibly a typo in host name
201	Successful POST
30x	Redirect; possibly a typo in the URL
401	Invalid username or password
405	Method not Allowed: check request syntax
500	Geoserver is unable to process the request, e.g. the workspace already exists, the xml is malformed, ...

For other codes see cURL Error Codes and HTTP Codes.

GET with PHP/cURL

The script above can be modified to perform a GET request to obtain the names of all workspaces by replacing the code blocks for required settings, data and return code with the following:

```
<?php
    // Required GET request settings
    // curl_setopt($ch, CURLOPT_GET, True); // CURLOPT_GET is True by default

    //GET data
    curl_setopt($ch, CURLOPT_HTTPHEADER, array("Accept: application/xml"));

    //GET return code
    $successCode = 200;
?>
```

The logfile should now include lines like:

```
> GET /geoserver/rest/workspaces HTTP/1.1

< HTTP/1.1 200 OK
```

DELETE with PHP/cURL

To delete the (empty) workspace we just created, the script is modified as follows:

```
<?php
    $request = "rest/workspaces/test_ws"; // to delete this workspace
?>

<?php
    //Required DELETE request settings
    curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "DELETE");
    $passwordStr = "admin:geoserver"; // replace with your username:password
    curl_setopt($ch, CURLOPT_USERPWD, $passwordStr);

    //DELETE data
    curl_setopt($ch, CURLOPT_HTTPHEADER,
              array("Content-type: application/atom+xml"));

    //DELETE return code
```

```
    $successCode = 200;
?>
```

The log file will include lines like:

```
> DELETE /geoserver/rest/workspaces/test_ws HTTP/1.1

< HTTP/1.1 200 OK
```

14.2.3 Python

We are looking for volunteers to flesh out this section with examples.

In the meantime, anyone looking to do python scripting of the GeoServer REST config API should use gsconfig.py. It is quite capable, and is used in production as part of GeoNode, so examples can be found in that codebase. It just currently lacks documentation and examples.

14.2.4 Java

We are looking for volunteers to flesh out this section with examples.

In the meantime, anyone looking to do Java scripting of the GeoServer REST API should use GeoServer Manager, a REST client library with minimal dependencies on external libraries.

Another option is gsrcj. This project is a GeoServer REST client written in Java with no extra dependencies on GeoServer/GeoTools, unlike the standard GeoServer REST module. The project has minimal documentation, but does include a Quickstart.

14.2.5 Ruby

The examples in this section use rest-client, a REST client for Ruby. There is also a project to create a GeoServer-specific REST client in Ruby: RGeoServer.

Once installed on a system, `rest-client` can be included in a Ruby script by adding `require 'rest-client'`.

GET and PUT Settings

This example shows how to read the settings using GET, make a change and then use PUT to write the change to the server.

```
require 'json'
require 'rest-client'

url = 'http://admin:geoserver@localhost:8080/geoserver/rest/'

# get the settings and parse the JSON into a Hash
json_text = RestClient.get(url + 'settings.json')
settings = JSON.parse(json_text)

# settings can be found with the appropriate keys
global_settings = settings["global"]
jai_settings = global_settings["jai"]
```

```
# change a value
jai_settings["allowInterpolation"] = true

# put changes back to the server
RestClient.put(url + 'settings, settings.to_json, :content_type => :json)
```

CHAPTER 15

Advanced GeoServer Configuration

GeoServer provides a variety of options to customize your service for different situations. While none of the configuration options discussed in this section are required for a basic GeoServer installation, they allow you to adapt your GeoServer to your own needs, beyond the options exposed in OGC standard services.

15.1 Coordinate Reference System Handling

This section describes how coordinate reference systems (CRS) are handled in GeoServer, as well as what can be done to extend GeoServer's CRS handling abilities.

15.1.1 Coordinate Reference System Configuration

When adding data, GeoServer tries to inspect data headers looking for an EPSG code:

- If the data has a CRS with an explicit EPSG code and the full CRS definition behind the code matches the one in GeoServer, the CRS will be already set for the data.
- If the data has a CRS but no EPSG code, you can use the *Find* option on the *Layers* page to make GeoServer perform a lookup operation where the data CRS is compared against every other known CRS. If this succeeds, an EPSG code will be selected. The common case for a CRS that has no EPSG code is shapefiles whose .PRJ file contains a valid WKT string without the EPSG identifiers (as these are optional).

If an EPSG code cannot be found, then either the data has no CRS or it is unknown to GeoServer. In this case, there are a few options:

- Force the declared CRS, ignoring the native one. This is the best solution if the native CRS is known to be wrong.
- Reproject from the native to the declared CRS. This is the best solution if the native CRS is correct, but cannot be matched to an EPSG number. (An alternative is to add a custom EPSG code that matches exactly the native SRS. See the section on *Custom CRS Definitions* for more information.)

If your data has no native CRS information, the only option is to specify/force an EPSG code.

15.1.2 Custom CRS Definitions

Add a custom CRS

This example shows how to add a custom projection in GeoServer.

1. The projection parameters need to be provided as a WKT (well known text) definition. The code sample below is just an example:

```
PROJCS["NAD83 / Austin",
  GEOGCS["NAD83",
    DATUM["North_American_Datum_1983",
      SPHEROID["GRS 1980", 6378137.0, 298.257222101],
      TOWGS84[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]],
    PRIMEM["Greenwich", 0.0],
    UNIT["degree", 0.017453292519943295],
    AXIS["Lon", EAST],
    AXIS["Lat", NORTH]],
  PROJECTION["Lambert_Conformal_Conic_2SP"],
  PARAMETER["central_meridian", -100.333333333333],
  PARAMETER["latitude_of_origin", 29.6666666666667],
  PARAMETER["standard_parallel_1", 31.883333333333297],
  PARAMETER["false_easting", 2296583.333333],
  PARAMETER["false_northing", 9842500.0],
  PARAMETER["standard_parallel_2", 30.1166666666667],
  UNIT["m", 1.0],
  AXIS["x", EAST],
  AXIS["y", NORTH],
  AUTHORITY["EPSG","100002"]]
```

Note: This code sample has been formatted for readability. The information will need to be provided on a single line instead, or with backslash characters at the end of every line (except the last one).

2. Go into the `user_projections` directory inside your data directory, and open the `epsg.properties` file. If this file doesn't exist, you can create it.

3. Insert the code WKT for the projection at the end of the file (on a single line or with backslash characters):

```
100002=PROJCS["NAD83 / Austin", \
  GEOGCS["NAD83", \
    DATUM["North_American_Datum_1983", \
      SPHEROID["GRS 1980", 6378137.0, 298.257222101], \
      TOWGS84[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]], \
    PRIMEM["Greenwich", 0.0], \
    UNIT["degree", 0.017453292519943295], \
    AXIS["Lon", EAST], \
    AXIS["Lat", NORTH]], \
  PROJECTION["Lambert_Conformal_Conic_2SP"], \
  PARAMETER["central_meridian", -100.333333333333], \
  PARAMETER["latitude_of_origin", 29.6666666666667], \
  PARAMETER["standard_parallel_1", 31.883333333333297], \
  PARAMETER["false_easting", 2296583.333333], \
  PARAMETER["false_northing", 9842500.0], \
  PARAMETER["standard_parallel_2", 30.1166666666667], \
  UNIT["m", 1.0], \
  AXIS["x", EAST], \
  AXIS["y", NORTH], \
  AUTHORITY["EPSG","100002"]]
```

Note: Note the number that precedes the WKT. This will determine the EPSG code. So in this example, the EPSG code is 100002.

1. Save the file.

2. Restart GeoServer.
3. Verify that the CRS has been properly parsed by navigating to the *SRS* page in the *Web Administration Interface*.
4. If the projection wasn't listed, examine the logs for any errors.

Override an official EPSG code

In some situations it is necessary to override an official EPSG code with a custom definition. A common case is the need to change the TOWGS84 parameters in order to get better reprojection accuracy in specific areas.

The GeoServer referencing subsystem checks the existence of another property file, `epsg_overrides.properties`, whose format is the same as `epsg.properties`. Any definition contained in `epsg_overrides.properties` will **override** the EPSG code, while definitions stored in `epsg.proeprties` can only **add** to the database.

Special care must be taken when overriding the Datum parameters, in particular the **TOWGS84** parameters. To make sure the override parameters are actually used the code of the Datum must be removed, otherwise the referencing subsystem will keep on reading the official database in search of the best Datum shift method (grid, 7 or 5 parameters transformation, plain affine transform).

For example, if you need to override the official **TOWGS84** parameters of EPSG:23031:

```
PROJCS["ED50 / UTM zone 31N",
  GEOGCS["ED50",
    DATUM["European Datum 1950",
      SPHEROID["International 1924", 6378388.0, 297.0, AUTHORITY["EPSG","7022"]],
      TOWGS84[-157.89, -17.16, -78.41, 2.118, 2.697, -1.434, -1.1097046576093785],
      AUTHORITY["EPSG","6230"]],
    PRIMEM["Greenwich", 0.0, AUTHORITY["EPSG","8901"]],
    UNIT["degree", 0.017453292519943295],
    AXIS["Geodetic longitude", EAST],
    AXIS["Geodetic latitude", NORTH],
    AUTHORITY["EPSG","4230"]],
 PROJECTION["Transverse_Mercator"],
  PARAMETER["central_meridian", 3.0],
  PARAMETER["latitude_of_origin", 0.0],
  PARAMETER["scale_factor", 0.9996],
  PARAMETER["false_easting", 500000.0],
  PARAMETER["false_northing", 0.0],
  UNIT["m", 1.0],
  AXIS["Easting", EAST],
  AXIS["Northing", NORTH],
  AUTHORITY["EPSG","23031"]]
```

You should write the following (in a single line, here it's reported formatted over multiple lines for readability):

```
23031=
 PROJCS["ED50 / UTM zone 31N",
   GEOGCS["ED50",
     DATUM["European Datum 1950",
       SPHEROID["International 1924", 6378388.0, 297.0, AUTHORITY["EPSG","7022"]],
       TOWGS84[-136.65549, -141.4658, -167.29848, 2.093088, 0.001405, 0.107709, 11.54611],
       AUTHORITY["EPSG","6230"]],
     PRIMEM["Greenwich", 0.0, AUTHORITY["EPSG","8901"]],
     UNIT["degree", 0.017453292519943295],
```

```
    AXIS["Geodetic longitude", EAST],
    AXIS["Geodetic latitude", NORTH]],
  PROJECTION["Transverse_Mercator"],
   PARAMETER["central_meridian", 3.0],
   PARAMETER["latitude_of_origin", 0.0],
   PARAMETER["scale_factor", 0.9996],
   PARAMETER["false_easting", 500000.0],
   PARAMETER["false_northing", 0.0],
   UNIT["m", 1.0],
   AXIS["Easting", EAST],
   AXIS["Northing", NORTH],
   AUTHORITY["EPSG","23031"]]
```

The definition has been changed in two places, the **TOWGS84** paramerers, and the Datum code, `AUTHORITY["EPSG","4230"]`, has been removed.

15.1.3 Coordinate Operations

Coordinate operations are used to convert coordinates from a *source CRS* to a *target CRS*.

If source and target CRSs are refered to a different datum, a datum transform has to be applied. Datum transforms are not exact, they are determined empirically. For the same pair of CRS, there can be many datum transforms and versions, each one with its own domain of validity and an associated transform error. Given a CRS pair, GeoServer will automatically pick the most accurate datum transform from the EPSG database, unless a custom operation is declared.

- Coordinate operations can be queried and tested using the Reprojection Console.
- To enable higher accuracy Grid Shift transforms, see Add Grid Shift Transform files.
- See Define a custom Coordinate Operation to declare new operations. Custom operations will take precedence over the EPSG ones.

Reprojection Console

The reprojection console (in *Demos => Reprojection console*) lets quickly test coordinate operations. Use it to convert a single coordinate or WKT geometry, and to see the operation details GeoServer is using. It is also useful to learn by example when you have to Define a custom Coordinate Operation.

Add Grid Shift Transform files

GeoServer supports NTv2 and NADCON grid shift transforms. Grid files are not shipped out with GeoServer. They need to be downloaded, usually from yor National Mapping Agency website.

> **Warning:** Grid Shift files are only valid in the specific geographic domain for which they where made; trying to transform coordinates outside this domain will result in no trasformation at all. Make sure that the Grid Shift files are valid in the area you want to transform.

1. Search for the *Grid File Name(s)* int the tables below, which are extracted from EPSG version 7.9.0. If you need to use a Grid Shift transform not declared in EPSG, you will need to Define a custom Coordinate Operation.
2. Get the Grid File(s) from your National Mapping Agency (NTv2) or the US National Geodetic Survey (NADCON).

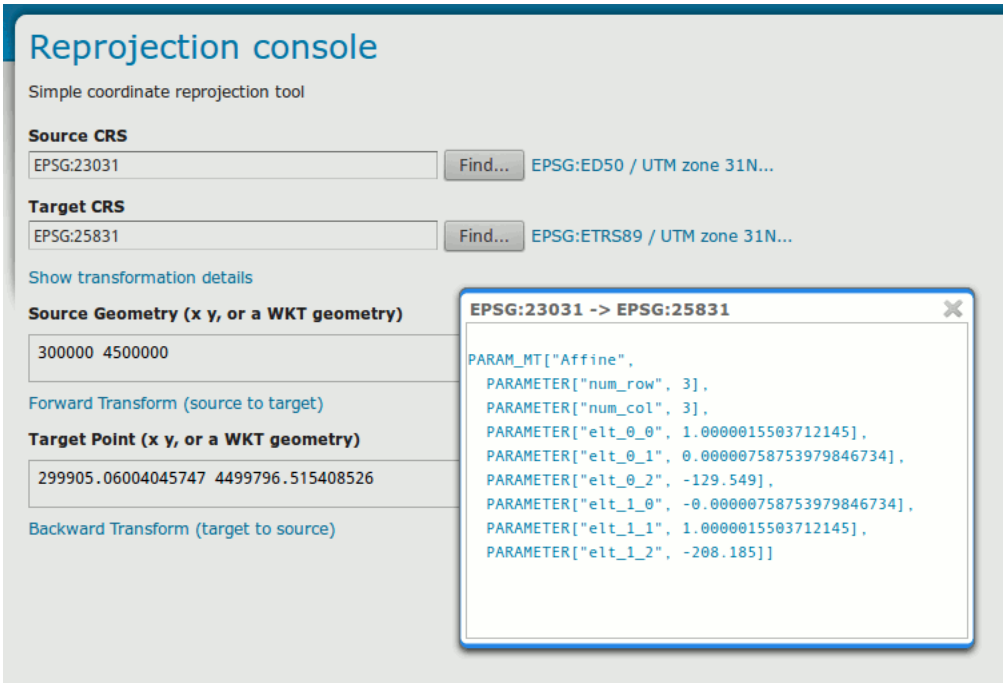


Figure 15.1: Reprojection console showing operation details and a transformed coordinate pair.

3. Copy the Grid File(s) in the `user_projections` directory inside your data directory.
4. Use the Reprojection Console to test the new transform.

List of available Grid Shift transforms

The list of Grid Shift transforms declared in EPSG version 7.9.0 is:

Source CRS	Target CRS	Grid File Name	Source Info
4122	4326	NB7783v2.gsb	OGP
4122	4326	NS778301.gsb	OGP
4122	4326	PE7783V2.gsb	OGP
4122	4617	NB7783v2.gsb	New Brunswick Geographic Information Corporation land a
4122	4617	NS778301.gsb	Nova Scotia Geomatics Centre - Contact aflemmin@linux1.n
4122	4617	PE7783V2.gsb	PEI Department of Transportation & Public Works
4149	4150	CHENYX06.gsb	Bundesamt für Landestopographie; www.swisstopo.ch
4171	4275	rgf93_ntf.gsb	ESRI
4202	4283	A66 National (13.09.01).gsb	GDA Technical Manual. http://www.icsm.gov.au/gda
4202	4283	SEAust_21_06_00.gsb	Office of Surveyor General Victoria; http://www.land.vic.g
4202	4283	nt_0599.gsb	GDA Technical Manual. http://www.icsm.gov.au/gda
4202	4283	tas_1098.gsb	http://www.delm.tas.gov.au/osg/Geodetic_transform.htm
4202	4283	vic_0799.gsb	Office of Surveyor General Victoria; http://www.land.vic.g
4202	4326	A66 National (13.09.01).gsb	OGP
4203	4283	National 84 (02.07.01).gsb	GDA Technical Manual. http://www.icsm.gov.au/gda
4203	4283	wa_0400.gsb	http://www.dola.wa.gov.au/lotl/survey_geodesy/gda199
4203	4283	wa_0700.gsb	Department of Land Information, Government of Western A
4203	4326	National 84 (02.07.01).gsb	OGP

Continued on next page

Table 15.1 – continued from previous page

Source CRS	Target CRS	Grid File Name	Source Info
4225	4326	CA7072_003.gsb	OGP
4225	4674	CA7072_003.gsb	IBGE.
4230	4258	SPED2ETV2.gsb	Instituto Geográfico Nacional, www.cnig.es
4230	4258	sped2et.gsb	Instituto Geográfico Nacional, www.cnig.es
4230	4326	SPED2ETV2.gsb	OGP
4230	4326	sped2et.gsb	OGP
4258	4275	rgf93_ntf.gsb	OGP
4267	4269	NTv2_0.gsb	http://www.geod.nrcan.gc.ca/products/html-public/GSD
4267	4269	QUE27-83.gsb	Geodetic Service of Quebec. Contact alain.bernard@mrn.gou
4267	4326	NTv2_0.gsb	OGP
4267	4326	QUE27-98.gsb	OGP
4267	4326	SK27-98.gsb	OGP
4267	4617	QUE27-98.gsb	Geodetic Service of Quebec. Contact alain.bernard@mrn.gou
4267	4617	SK27-98.gsb	Dir Geodetic Surveys; SaskGeomatics Div.; Saskatchewan Pr
4269	4326	AB_CSRS.DAC	OGP
4269	4326	NAD83-98.gsb	OGP
4269	4326	SK83-98.gsb	OGP
4269	4617	AB_CSRS.DAC	Geodetic Control Section; Land and Forest Svc; Alberta Envi
4269	4617	NAD83-98.gsb	Geodetic Service of Quebec. Contact alain.bernard@mrn.gou
4269	4617	SK83-98.gsb	Dir Geodetic Surveys; SaskGeomatics Div.; Saskatchewan Pr
4272	4167	nzgd2kgrid0005.gsb	Land Information New Zealand: LINZS25000 Standard for N
4272	4326	nzgd2kgrid0005.gsb	OGP
4277	4258	OSTN02_NTv2.gsb	Ordnance Survey of Great Britain, http://www.gps
4277	4326	OSTN02_NTv2.gsb	OGP
4314	4258	BETA2007.gsb	BKG via EuroGeographics http://crs.bkg.bund.de/
4314	4326	BETA2007.gsb	OGP
4326	4275	rgf93_ntf.gsb	OGP
4608	4269	May76v20.gsb	Geodetic Survey of Canada http://www.geod.nrcan
4608	4326	May76v20.gsb	OGP
4609	4269	CGQ77-83.gsb	Geodetic Service of Quebec. Contact alain.bernard@
4609	4326	CGQ77-98.gsb	OGP
4609	4617	CGQ77-98.gsb	Geodetic Service of Quebec. Contact alain.bernard@
4618	4326	SAD69_003.gsb	OGP
4618	4674	SAD69_003.gsb	IBGE.
4745	4326	BETA2007.gsb	OGP
4746	4326	BETA2007.gsb	OGP
4749	4644	RGNC1991_NEA74Noumea.gsb	ESRI
4749	4662	RGNC1991_IGN72GrandeTerre.gsb	ESRI
5524	4326	CA61_003.gsb	OGP
5524	4674	CA61_003.gsb	IBGE.
5527	4326	SAD96_003.gsb	OGP
5527	4674	SAD96_003.gsb	IBGE.

NTv2

Source CRS	Target CRS	Version	Latitude shift file	Longitude shift file
4135	4269	NGS-Usa HI	hawaii.las	hawaii.los
4136	4269	NGS-Usa AK StL	stlrnc.las	stlrnc.los
4137	4269	NGS-Usa AK StP	stpaul.las	stpaul.los

Continued on next page

Table 15.2 – continued from previous page

Source CRS	Target CRS	Version	Latitude shift file	Longitude shift file
4138	4269	NGS-Usa AK StG	stgeorge.las	stgeorge.los
4139	4269	NGS-PRVI	prvi.las	prvi.los
4169	4152	NGS-Asm E	eshpgn.las	eshpgn.los
4169	4152	NGS-Asm W	wshpgn.las	wshpgn.los
4267	4269	NGS-Usa AK	alaska.las	alaska.los
4267	4269	NGS-Usa Conus	conus.las	conus.los
4269	4152	NGS-Usa AL	alhpgn.las	alhpgn.los
4269	4152	NGS-Usa AR	arhpgn.las	arhpgn.los
4269	4152	NGS-Usa AZ	azhpgn.las	azhpgn.los
4269	4152	NGS-Usa CA n	cnhpgn.las	cnhpgn.los
4269	4152	NGS-Usa CO	cohpgn.las	cohpgn.los
4269	4152	NGS-Usa CA s	cshpgn.las	cshpgn.los
4269	4152	NGS-Usa ID MT e	emhpgn.las	emhpgn.los
4269	4152	NGS-Usa TX e	ethpgn.las	ethpgn.los
4269	4152	NGS-Usa FL	flhpgn.las	flhpgn.los
4269	4152	NGS-Usa GA	gahpgn.las	gahpgn.los
4269	4152	NGS-Usa HI	hihpgn.las	hihpgn.los
4269	4152	NGS-Usa IA	iahpgn.las	iahpgn.los
4269	4152	NGS-Usa IL	ilhpgn.las	ilhpgn.los
4269	4152	NGS-Usa IN	inhpgn.las	inhpgn.los
4269	4152	NGS-Usa KS	kshpgn.las	kshpgn.los
4269	4152	NGS-Usa KY	kyhpgn.las	kyhpgn.los
4269	4152	NGS-Usa LA	lahpgn.las	lahpgn.los
4269	4152	NGS-Usa DE MD	mdhpgn.las	mdhpgn.los
4269	4152	NGS-Usa ME	mehpgn.las	mehpgn.los
4269	4152	NGS-Usa MI	mihpgn.las	mihpgn.los
4269	4152	NGS-Usa MN	mnhpgn.las	mnhpgn.los
4269	4152	NGS-Usa MO	mohpgn.las	mohpgn.los
4269	4152	NGS-Usa MS	mshpgn.las	mshpgn.los
4269	4152	NGS-Usa NE	nbhpgn.las	nbhpgn.los
4269	4152	NGS-Usa NC	nchpgn.las	nchpgn.los
4269	4152	NGS-Usa ND	ndhpgn.las	ndhpgn.los
4269	4152	NGS-Usa NewEng	nehpgn.las	nehpgn.los
4269	4152	NGS-Usa NJ	njhpgn.las	njhpgn.los
4269	4152	NGS-Usa NM	nmhpgn.las	nmhpgn.los
4269	4152	NGS-Usa NV	nvhpgn.las	nvhpgn.los
4269	4152	NGS-Usa NY	nyhpgn.las	nyhpgn.los
4269	4152	NGS-Usa OH	ohhpgn.las	ohhpgn.los
4269	4152	NGS-Usa OK	okhpgn.las	okhpgn.los
4269	4152	NGS-Usa PA	pahpgn.las	pahpgn.los
4269	4152	NGS-PRVI	pvhpgn.las	pvhpgn.los
4269	4152	NGS-Usa SC	schpgn.las	schpgn.los
4269	4152	NGS-Usa SD	sdhpgn.las	sdhpgn.los
4269	4152	NGS-Usa TN	tnhpgn.las	tnhpgn.los
4269	4152	NGS-Usa UT	uthpgn.las	uthpgn.los
4269	4152	NGS-Usa VA	vahpgn.las	vahpgn.los
4269	4152	NGS-Usa WI	wihpgn.las	wihpgn.los
4269	4152	NGS-Usa ID MT w	wmhpgn.las	wmhpgn.los
4269	4152	NGS-Usa OR WA	wohpgn.las	wohpgn.los
4269	4152	NGS-Usa TX w	wthpgn.las	wthpgn.los
4269	4152	NGS-Usa WV	wvhpgn.las	wvhpgn.los

Continued on next page

Table 15.2 – continued from previous page

Source CRS	Target CRS	Version	Latitude shift file	Longitude shift file
4269	4152	NGS-Usa WY	wyhpgn.las	wyhpgn.los
4675	4152	NGS-Gum	guhpgn.las	guhpgn.los

NADCON

Define a custom Coordinate Operation

Custom Coordinate Operations are defined in `epsg_operations.properties` file. This file has to be placed into the `user_projections` directory, inside your data directory (create it if it doesn't exist).

Each line in `epsg_operations.properties` will describe a coordinate operation consisting of a *source CRS*, a *target CRS*, and a math transform with its parameter values. Use the following syntax:

```
<source crs code>,<target crs code>=<WKT math transform>
```

Math transform is described in Well-Known Text syntax. Parameter names and value ranges are described in the EPSG Geodetic Parameter Registry.

Note: Use the Reprojection Console to learn from example and to test your custom definitions.

Examples

Custom NTv2 file:

```
4230,4258=PARAM_MT["NTv2", \
  PARAMETER["Latitude and longitude difference file", "100800401.gsb"]]
```

Geocentric transformation, preceded by an ellipsoid to geocentric conversion, and back geocentric to ellipsoid. The results is a concatenation of three math transforms:

```
4230,4258=CONCAT_MT[PARAM_MT["Ellipsoid_To_Geocentric", \
  PARAMETER["dim", 2], \
  PARAMETER["semi_major", 6378388.0], \
  PARAMETER["semi_minor", 6356911.9461279465]], \
PARAM_MT["Position Vector transformation (geog2D domain)", \
  PARAMETER["dx", -116.641], \
  PARAMETER["dy", -56.931], \
  PARAMETER["dz", -110.559], \
  PARAMETER["ex", 0.8925078166311858], \
  PARAMETER["ey", 0.9207660950870382], \
  PARAMETER["ez", -0.9166407989620964], \
  PARAMETER["ppm", -3.5200000000346066]], \
PARAM_MT["Geocentric_To_Ellipsoid", \
  PARAMETER["dim", 2], \
  PARAMETER["semi_major", 6378137.0], \
  PARAMETER["semi_minor", 6356752.314140356]]]
```

Affine 2D transform operating directly in projected coordinates:

```
23031,25831=PARAM_MT["Affine", \
  PARAMETER["num_row", 3], \
  PARAMETER["num_col", 3], \
  PARAMETER["elt_0_0", 1.0000015503712145], \
```

```
PARAMETER["elt_0_1", 0.00000758753979846734], \
PARAMETER["elt_0_2", -129.549], \
PARAMETER["elt_1_0", -0.00000758753979846734], \
PARAMETER["elt_1_1", 1.0000015503712145], \
PARAMETER["elt_1_2", -208.185]]
```

Each operation can be described in a single line, or can be split in several lines for readability, adding a backslash "\" at the end of each line, as in the former examples.

15.1.4 Manually editing the EPSG database

Warning: These instructions are very advanced, and are here mainly for the curious who want to know details about the EPSG database subsystem.

To define a custom projection, edit the EPSG.sql file, which is used to create the cached EPSG database.

1. Navigate to the `WEB-INF/lib` directory
2. Uncompress the `gt2-epsg-h.jar` file. On Linux, the command is:

   ```
   jar xvf gt2-epsg-h.jar
   ```

3. Open `org/geotools/referencing/factory/epsg/EPSG.sql` with a text editor. To add a custom projection, these entries are essential:
   (a) An entry in the EPSG_COORDINATEREFERENCESYSTEM table:

   ```
   (41111,'WGC 84 / WRF Lambert',1324,'projected',4400,NULL,4326,20000,NULL,NULL,'US Nat. scale
   ```

   where:

   - **1324** is the EPSG_AREA code that describes the area covered by my projection
   - **4400** is the EPSG_COORDINATESYSTEM code for my projection
   - **20000** is the EPSG_COORDOPERATIONPARAMVALUE key for the array that contains my projection parameters

   (b) An entry in the EPSG_COORDOPERATIONPARAMVALUE table:

   ```
   (20000,9802,8821,40,'',9102),    //latitude of origin
   (20000,9802,8822,-97.0,'',9102), //central meridian
   (20000,9802,8823,33,'',9110),    //st parallel 1
   (20000,9802,8824,45,'',9110),    //st parallel 2
   (20000,9802,8826,0.0,'',9001),   //false easting
   (20000,9802,8827,0.0,'',9001)    //false northing
   ```

   where:

   - **9802** is the EPSG_COORDOPERATIONMETHOD key for the Lambert Conic Conformal (2SP) formula

   (c) An entry in the EPSG_COORDOPERATION table:

   (20000,'WRF Lambert','conversion',NULL,NULL,'',NULL,1324,'Used for weather forecasting.',0.0,9802,NULL,NULL,'Used with the WRF-Chem model for weather forecasting','Firelab in Missoula, MT','EPSG','2005-11-23','2005.01',1,0)

   where:

- **1324** is the EPSG_AREA code that describes the area covered by my projection
- **9802** is the EPSG_COORDOPERATIONMETHOD key for the Lambert Conic Conformal (2SP) formula

---

Note: Observe the commas. If you enter a line that is at the end of an INSERT statement, the comma is omitted (make sure the row before that has a comma at the end). Otherwise, add a comma at the end of your entry.

---

1. After all edits, save the file and exit.
2. Compress the gt2-epsg-h.jar file. On Linux, the command is:

   ```
   jar -Mcvf gt2-epsg-h.jar META-INF org
   ```

3. Remove the cached copy of the EPSG database, so that can be recreated. On Linux, the command is:

   ```
   rm -rf /tmp/Geotools/Databases/HSQL
   ```

4. Restart GeoServer.

The new projection will be successfully parsed. Verify that the CRS has been properly parsed by navigating to the *SRS* page in the *Web Administration Interface*.

15.2 Advanced log configuration

GeoServer logging subsystem is based on Java logging, which is in turn by default redirected to Log4J and controlled by the current logging configuration set in the *Global Settings*.

The standard configuration can be overridden in a number of ways to create custom logging profiles or to force GeoServer to use another logging library altogheter.

15.2.1 Custom logging profiles

Anyone can write a new logging profile by adding a Log4J configuration file to the list of files already available in the `$GEOSERVER_DATA_DIR/logs` folder. The name of the file will become the configuration name displayed in the admin console and the contents will drive the specific behavior of the logger.

Here is an example, taken from the `GEOTOOLS_DEVELOPER_LOGGING` configuration, which enables the geotools log messages to appear in the logs:

```
log4j.rootLogger=WARN, geoserverlogfile, stdout

log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{dd MMM HH:mm:ss} %p [%c] - %m%n

log4j.category.log4j=FATAL

log4j.appender.geoserverlogfile=org.apache.log4j.RollingFileAppender
# Keep three backup files.
log4j.appender.geoserverlogfile.MaxBackupIndex=3
# Pattern to output: date priority [category] - message
log4j.appender.geoserverlogfile.layout=org.apache.log4j.PatternLayout
log4j.appender.geoserverlogfile.layout.ConversionPattern=%d %p [%c] - %m%n
```

```
log4j.category.org.geotools=TRACE
# Some more geotools loggers you may be interest in tweaking
log4j.category.org.geotools.factory=TRACE
log4j.category.org.geotools.renderer=DEBUG
log4j.category.org.geotools.data=TRACE
log4j.category.org.geotools.feature=TRACE
log4j.category.org.geotools.filter=TRACE
log4j.category.org.geotools.factory=TRACE

log4j.category.org.geoserver=INFO
log4j.category.org.vfny.geoserver=INFO

log4j.category.org.springframework=WARN
```

Any custom configuration can be setup to enable specific packages to emit logs at the desired logging level. There are however a few rules to follow:

- the configuration should always include a `geoserverlogfile` appender that GeoServer will configure to work against the location configured in the *Global Settings*
- a logger writing to the standard output should be called `stdout` and again GeoServer will enable/disable it according to the configuration set in the *Global Settings*
- it is advisable, but not require, to setup log rolling for the `geoserverlogfile` appender

15.2.2 Overriding the log location setup in the GeoServer configuration

When setting up a cluster of GeoServer machines it is common to share a single data directory among all the cluster nodes. There is however a gotcha, all nodes would end up writing the logs in the same file, which would cause various kinds of troubles depending on the operating system file locking rules (a single server might be able to write, or all togheter in an uncontrolled manner resulting in an unreadable log file).

In this case it is convenient to set a separate log location for each GeoServer node by setting the following parameter among the JVM system variables, enviroment variables, or servlet context parameters:

```
GEOSERVER_LOG_LOCATION=<the location of the file>
```

A common choice could be to use the machine name as a distinction, setting values such as `logs/geoserver_node1.log`, `logs/geoserver_node2.log` and so on: in this case all the log files would still be contained in the data directory and properly rotated, but each server would have its own separate log file to write on.

15.2.3 Forcing GeoServer to relinquish Log4J control

GeoServer internally overrides the Log4J configuration by using the current logging configuration as a template and appling the log location and standard output settings configured by the administrator.

If you wish GeoServer not to override the normal Log4J behavior you can set the following parameter among the JVM system variables, enviroment variables, or servlet context parameters:

```
RELINQUISH_LOG4J_CONTROL=true
```

15.2.4 Forcing GeoServer to use an alternate logging redirection

GeoServer uses the GeoTools logging framework, which in turn is based on Java Logging, but allowing to redirect all message to an alternate framework of users choice.

By default GeoServer setups a Log4J redirection, but it is possible to configure GeoServer to use plain Java Logging or Commons Logging instead (support for other loggers is also possible by using some extra programming).

If you wish to force GeoServer to use a different logging mechanism set the following parameters among the JVM system variables, enviroment variables, or servlet context parameters:

```
GT2_LOGGING_REDIRECTION=[JavaLogging,CommonsLogging,Log4J]
RELINQUISH_LOG4J_CONTROL=true
```

As noted in the example you'll also have to demand that GeoServer does not exert control over the Log4J configuration

15.3 WMS Decorations

WMS Decorations provide a framework for visually annotating images from WMS with absolute, rather than spatial, positioning. Examples of decorations include compasses, legends, and watermarks.

15.3.1 Configuration

To use decorations in a *GetMap* request, the administrator must first configure a decoration layout. These layouts are stored in a subdirectory called `layouts` in the *GeoServer Data Directory* as XML files, one file per layout. Each layout file must have the extension `.xml`. Once a layout `foo.xml` is defined, users can request it by adding `&format_options=layout:foo` to the request parameters.

Layout files follow a very simple XML structure; a root node named layout containing any number of decoration elements. Each decoration element has several attributes:

Attribute	Meaning
`type`	the type of decoration to use (see *Decoration Types*)
`affinity`	the region of the map image to which the decoration is anchored
`offset`	how far from the anchor point the decoration is drawn
`size`	the maximum size to render the decoration. Note that some decorations may dynamically resize themselves.

Each decoration element may also contain an arbitrary number of option elements providing a parameter name and value:

```
<option name="foo" value="bar"/>
```

Option interpretation depends on the type of decoration in use.

15.3.2 Decoration Types

While GeoServer allows for decorations to be added via extension, there is a core set of decorations included in the default installation. These decorations include:

The **image** decoration (`type="image"`) overlays a static image file onto the document. If height and width are specified, the image will be scaled to fit, otherwise the image is displayed at full size.

Option Name	Meaning
url	provides the URL or file path to the image to draw (relative to the GeoServer data directory)
opacity	a number from 0 to 100 indicating how opaque the image should be.

The **scaleratio** decoration (`type="scaleratio"`) overlays a text description of the map's scale ratio onto the document.

Option Name	Meaning
bgcolor	the background color for the text. supports RGB or RGBA colors specified as hex values.
fgcolor	the color for the text and border. follows the color specification from bgcolor.

The **scaleline** decoration (`type="scaleline"`) overlays a graphic showing the scale of the map in world units.

Option Name	Meaning
bgcolor	the background color, as used in scaleratio
fgcolor	the foreground color, as used in scaleratio
fontsize	the size of the font to use
transparent	if set to true, the background and border won't be painted (false by default)

The **legend** decoration (`type="legend"`) overlays a graphic containing legends for the layers in the map.

The **text** decoration (`type="text"`) overlays a parametric, single line text message on top of the map. The parameter values can be fed via the the `env` request parameter, just like SLD enviroment parameters.

Option Name	Meaning
message	the message to be displayed, as plain text or Freemarker template that can use the `env` map contents to expand variables
font-family	the name of the font used to display the message, e.g., `Arial`, defaults to `Serif`
font-size	the size of the font to use (can have decimals), defaults to 12
font-italic	if `true` the font will be italic, defaults to `false`
font-bold	if `true` the font will be bold, defaults to `false`
font-color	the color of the message, in `#RRGGBB` or `#RRGGBBAA` format, defaults to black
halo-radius	the radius of a halo around the message, can have decimals, defaults to 0
halo-color	the halo fill color, in `#RRGGBB` or `#RRGGBBAA` format, defaults to white

15.3.3 Example

A layout configuration file might look like this:

```
<layout>
    <decoration type="image" affinity="bottom,right" offset="6,6" size="80,31">
        <option name="url" value="pbGS_80x31glow.png"/>
    </decoration>

    <decoration type="scaleline" affinity="bottom,left" offset="36,6"/>

    <decoration type="legend" affinity="top,left" offset="6,6" size="auto"/>
</layout>
```

Used against the states layer from the default GeoServer data, this layout produces an image like the following.

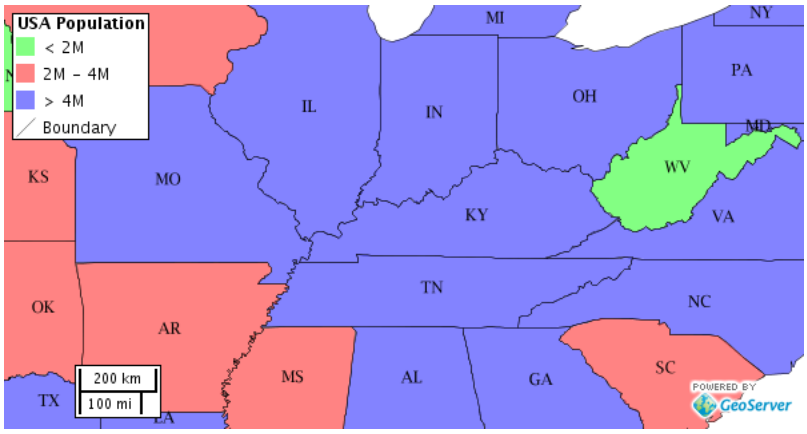


Figure 15.2: The default states layer, drawn with the decoration layout above.

CHAPTER 16

Security

This section details the security subsystem in GeoServer, which is based on Spring Security. For web-based configuration, please see the section on *Security* in the *Web Administration Interface*.

As of GeoServer 2.2.0, the security subsystem has been completely re-engineered, providing a more secure and flexible authentication framework. This rework is largely based on a Christian Müeller's masters thesis entitled Flexible Authentication for Stateless Web Services. It is good reading to help understanding many of the new concepts introduced.

16.1 Role system

Security in GeoServer is based on a **role-based system**, with roles created to serve particular functions. Examples of roles sporting a particular function are those accessing the Web Feature Service (WFS), administering the *Web Administration Interface*, and reading a specific layer. Roles are assigned to users and groups of users, and determine what actions those users or groups are permitted to do. A user is authorized through *Authentication*.

16.1.1 Users and Groups

The definition of a GeoServer **user** is similar to most security systems. Although the correct Java term is **principle**—a principle being a human being, computer, software system, and so on—the term **user** is adopted throughout the GeoServer documentation. For each user the following information is maintained:

- User name
- *Password* (optionally stored *encrypted*)
- A flag indicating if the user is enabled (this is the default). A disabled user is prevented from logging on. Existing user sessions are not affected.
- Set of key/value pairs

Key/value pairs are implementation-specific and may be configured by the *user/group service* the user or group belongs to. For example, a user/group service that maintains information about a user such as Name, Email address, and so on, may wish to associate those attributes with the user object.

A GeoServer **group** is simply a set of users. For each group the following information is maintained:

- Group name
- A flag indicating if the group is enabled (this is the default). A disabled group does not contribute to the role calculation for all users contained in this group.

- List of users who belong to the group

16.1.2 User/group services

A **user/group service** provides the following information for users and groups:

- Listing of users
- Listing of groups, including users affiliated with each group
- User passwords

Many authentication providers will make use of a user/group service to perform authentication. In this case, the user/group service would be the database against which users and passwords are authenticated. Depending on how the *Authentication chain* is configured, there may be zero, one, or multiple user/group services active at any given time.

A user/group service may be read-only, providing access to user information but not allowing new users and groups to be added or altered. This may occur if a user/group service was configured to delegate to an external service for the users and groups database. An example of this would be an external LDAP server.

By default, GeoServer support two types of user/group services:

- XML—*(Default)* User/group service persisted as XML
- JDBC—User/group service persisted in database via JDBC

XML user/group service

The XML user/group service persists the user/group database in an XML file. This is the default behavior in GeoServer. This service represents the user database as XML, and corresponds to this `XML schema`.

Note: The XML user/group file, `users.xml`, is located in the GeoServer data directory, `security/usergroup/<name>/users.xml`, where `<name>` is the name of the user/group service.

The following is the contents of `users.xml` that ships with the default GeoServer configuration:

```
<userRegistry version="1.0" xmlns="http://www.geoserver.org/security/users">
  <users>
    <user enabled="true" name="admin" password="crypt1:5WK8hBrtrte9wtImg5i5fjnd8VeqCjDB"/>
  </users>
  <groups/>
</userRegistry>
```

This particular configuration defines a single user, `admin`, and no groups. By default, stored user passwords are encrypted using the *weak PBE* method.

For further information, please refer to *configuring a user/group service* in the *Web Administration Interface*.

JDBC user/group service

The JDBC user/group service persists the user/group database via JDBC, managing the user information in multiple tables. The user/group database schema is as follows:

Table 16.1: Table: users

Field	Type	Null	Key
name	varchar(128)	NO	PRI
password	varchar(254)	YES	
enabled	char(1)	NO	

Table 16.2: Table: user_props

Field	Type	Null	Key
username	varchar(128)	NO	PRI
propname	varchar(64)	NO	PRI
propvalue	varchar(2048)	YES	

Table 16.3: Table: groups

Field	Type	Null	Key
name	varchar(128)	NO	PRI
enabled	char(1)	NO	

Table 16.4: Table: group_members

Field	Type	Null	Key
groupname	varchar(128)	NO	PRI
username	varchar(128)	NO	PRI

The `users` table is the primary table and contains the list of users with associated passwords. The `user_props` table maps additional properties to a user. (See *Users and Groups* for more details.) The `groups` table lists all available groups, and the `group_members` table maps which users belong to which groups.

The default GeoServer security configuration is:

Table 16.5: Table: users

name	password	enabled
Empty	*Empty*	*Empty*

Table 16.6: Table: user_props

username	propname	propvalue
Empty	*Empty*	*Empty*

Table 16.7: Table: groups

name	enabled
Empty	*Empty*

Table 16.8: Table: group_members

groupname	username
Empty	*Empty*

For further information, please refer to *configuring a user/group service* in the *Web Administration Interface*.

16.1.3 Roles

GeoServer **roles** are keys associated with performing certain tasks or accessing particular resources. Roles are assigned to users and groups, authorizing them to perform the actions associated with the role. A

GeoServer role includes the following:

- Role name
- Parent role
- Set of key/value pairs

GeoServer roles support inheritance—a child role inherits all the access granted to the parent role. For example, suppose you have one role named `ROLE_SECRET` and another role, `ROLE_VERY_SECRET`, that extends `ROLE_SECRET`. `ROLE_VERY_SECRET` can access everything `ROLE_SECRET` can access, but not vice versa.

Key/value pairs are implementation-specific and may be configured by the *role service* the user or group belongs to. For example, a role service that assigns roles based on employee organization may wish to associate additional information with the role such as Department Name.

Geoserver has a number of system roles, the names of which are reserved. Adding a new GeoServer role with reserved name is not permitted.

- `ROLE_ADMINISTRATOR`—Provides access to all operations and resources
- `ROLE_GROUP_ADMIN`—Special role for administrating user groups
- `ROLE_AUTHENTICATED`—Assigned to every user authenticating successfully
- `ROLE_ANONYMOUS`—Assigned if anonymous authentication is enabled and user does not log on

16.1.4 Role services

A **role service** provides the following information for roles:

- List of roles
- Calculation of role assignments for a given user
- Mapping of a role to the system role `ROLE_ADMINISTRATOR`
- Mapping of a role to the system role `ROLE_GROUP_ADMIN`

When a user/group service loads information about a user or a group, it delegates to the role service to determine which roles should be assigned to the user or group. Unlike *User/group services*, only one role service is active at any given time.

By default, GeoServer supports two types of role services:

- XML—*(Default)* role service persisted as XML
- JDBC—Role service persisted in a database via JDBC

Mapping roles to system roles

To assign the system role `ROLE_ADMINISTRATOR` to a user or to a group, a new role with a different name must be created and mapped to the `ROLE_ADMINISTRATOR` role. The same holds true for the system role `ROLE_GROUP_ADMIN`. The mapping is stored in the service's `config.xml` file.

```
<roleService>
  <id>471ed59f:13915c479bc:-7ffc</id>
  <name>default</name>
  <className>org.geoserver.security.xml.XMLRoleService</className>
  <fileName>roles.xml</fileName>
  <checkInterval>10000</checkInterval>
```

```
  <validating>true</validating>
  <adminRoleName>ADMIN</adminRoleName>
  <groupAdminRoleName>GROUP_ADMIN</groupAdminRoleName>
</roleService>
```

In this example, a user or a group assigned to the role `ADMIN` is also assigned to the system role `ROLE_ADMINISTRATOR`. The same holds true for `GROUP_ADMIN` and `ROLE_GROUP_ADMIN`.

XML role service

The XML role service persists the role database in an XML file. This is the default role service for GeoServer. This service represents the user database as XML, and corresponds to this `XML schema`.

Note: The XML role file, `roles.xml`, is located in the GeoServer data directory, `security/role/<name>/roles.xml`, where `<name>` is the name of the role service.

The service is configured to map the local role `ADMIN` to the system role `ROLE_ADMINISTRATOR`. Additionally, `GROUP_ADMIN` is mapped to `ROLE_GROUP_ADMIN`. The mapping is stored the `config.xml` file of each role service.

The following provides an illustration of the `roles.xml` that ships with the default GeoServer configuration:

```
<roleRegistry version="1.0" xmlns="http://www.geoserver.org/security/roles">
  <roleList>
    <role id="ADMIN"/>
    <role id="GROUP_ADMIN"/>
  </roleList>
  <userList>
    <userRoles username="admin">
      <roleRef roleID="ADMIN"/>
    </userRoles>
  </userList>
  <groupList/>
</roleRegistry>
```

This configuration contains two roles named `ADMIN` and `GROUP_ADMIN`. The role `ADMIN` is assigned to the `admin` user. Since the `ADMIN` role is mapped to the system role `ROLE_ADMINISTRATOR`, the role calculation assigns both roles to the `admin` user.

For further information, please refer to *configuring a role service* in the *Web Administration Interface*.

J2EE role service

The J2EE role service parses roles from the `WEB-INF/web.xml` file. As a consequence, this service is a read only role service. Roles are extracted from the following XML elements:

```
<security-role>

    <security-role>
       <role-name>role1</role-name>
    </security-role>
    <security-role>
       <role-name>role2</role-name>
```

```
</security-role>
<security-role>
   <role-name>employee</role-name>
</security-role>
```

Roles retrieved:

- role1
- role2
- employee

```
<security-constraint>
```

```
 <security-constraint>
    <web-resource-collection>
        <web-resource-name>Protected Area</web-resource-name>
        <url-pattern>/jsp/security/protected/*</url-pattern>
        <http-method>PUT</http-method>
        <http-method>DELETE</http-method>
        <http-method>GET</http-method>
        <http-method>POST</http-method>
    </web-resource-collection>
    <auth-constraint>
        <role-name>role1</role-name>
        <role-name>employee</role-name>
    </auth-constraint>
</security-constraint>
```

Roles retrieved:

- role1
- employee

```
<security-role-ref>
```

```
<security-role-ref>
    <role-name>MGR</role-name>
    <!-- role name used in code -->
    <role-link>employee</role-link>
  </security-role-ref>
```

Roles retrieved:

- MGR

JDBC role service

The JDBC role service persists the role database via JDBC, managing the role information in multiple tables. The role database schema is as follows:

Table 16.9: Table: roles

Field	Type	Null	Key
name	varchar(64)	NO	PRI
parent	varchar(64)	YES	

Table 16.10: Table: role_props

Field	Type	Null	Key
rolename	varchar(64)	NO	PRI
propname	varchar(64)	NO	PRI
propvalue	varchar(2048)	YES	

Table 16.11: Table: user_roles

Field	Type	Null	Key
username	varchar(128)	NO	PRI
rolename	varchar(64)	NO	PRI

Table 16.12: Table: group_roles

Field	Type	Null	Key
groupname	varchar(128)	NO	PRI
rolename	varchar(64)	NO	PRI

The `roles` table is the primary table and contains the list of roles. Roles in GeoServer support inheritance, so a role may optionally have a link to a parent role. The `role_props` table maps additional properties to a role. (See the section on *Roles* for more details.) The `user_roles` table maps users to the roles they are assigned. Similarly the `group_roles` table maps which groups have been assigned to which roles.

The default GeoServer security configuration is:

Table 16.13: Table: roles

name	parent
Empty	*Empty*

Table 16.14: Table: role_props

rolename	propname	propvalue
Empty	*Empty*	*Empty*

Table 16.15: Table: user_roles

username	rolename
Empty	*Empty*

Table 16.16: Table: group_roles

groupname	rolename
Empty	*Empty*

For further information, please refer to *configuring a role service* in the *Web Administration Interface*.

LDAP role service

The LDAP role service is a read only role service that maps groups from an LDAP repository to GeoServer roles.

Groups are extracted from a specific LDAP node, configured as the `Groups search base`. A role is mapped for every matching group. The role will have a name that is built taking the Group common name (cn attribute), transformed to upper case and with a `ROLE_` prefix applied.

It is possible to filter extracted groups using an `All groups filter` (defaults to `cn=*` that basically extracts all nodes from the configured base). It is also possible to configure the filter for `users to roles membership` (defaults to member={0}).

A specific group can be assigned to the `ROLE_ADMINISTRATOR` and/or the `ROLE_GROUP_ADMIN` administrative roles.

Groups extraction can be done anonymously or using a given username/password if the LDAP repository requires it.

An example of configuration file (config.xml) for this type of role service is the following:

```
<org.geoserver.security.ldap.LDAPRoleServiceConfig>
  <id>-36dfbd50:1424687f3e0:-8000</id>
  <name>ldapacme</name>
  <className>org.geoserver.security.ldap.LDAPRoleService</className>
  <serverURL>ldap://127.0.0.1:10389/dc=acme,dc=org</serverURL>
  <groupSearchBase>ou=groups</groupSearchBase>
  <groupSearchFilter>member=uid={0},ou=people,dc=acme,dc=org</groupSearchFilter>
  <useTLS>false</useTLS>
  <bindBeforeGroupSearch>true</bindBeforeGroupSearch>
  <adminGroup>ROLE_ADMIN</adminGroup>
  <groupAdminGroup>ROLE_ADMIN</groupAdminGroup>
  <user>uid=bill,ou=people,dc=acme,dc=org</user>
  <password>hello</password>
  <allGroupsSearchFilter>cn=*</allGroupsSearchFilter>
</org.geoserver.security.ldap.LDAPRoleServiceConfig>
```

For further information, please refer to *configuring a role service* in the *Web Administration Interface*.

16.1.5 Role source and role calculation

Different authentication mechanisms provide different possibilities where to look for the roles of a principal/user. The role source is the base for the calculation of the roles assigned to the authenticated principal.

Using a user/group Service

During configuration of an authentication mechanism, the name of a user group service has to be specified. The used role service is always the role service configured as active role service. The role calculation itself is described here *Interaction between user/group and role services*

Using a role service directly

During configuration of an authentication mechanism, the name of a role service has to be specified. The calculation of the roles works as follows:

1. Fetch all roles for the user.
2. For each role in the result set, fetch all ancestor roles and add those roles to the result set.
3. If the result set contains the local admin role, add the role `ROLE_ADMINISTRATOR`.
4. If the result set contains the local group admin role, add the role `ROLE_GROUP_ADMIN`.

This algorithm does not offer the possibility to have personalized roles and it does not consider group memberships.

Using an HTTP header attribute

The roles for a principal are sent by the client in an HTTP header attribute (Proxy authentication). GeoServer itself does no role calculation and extracts the roles from the header attribute. During configuration, the name of the header attribute must be specified. An example with a header attribute named "roles":

```
roles: role_a;role_b;role_c
```

An example for roles with role parameters:

```
roles: role_a;role_b(pnr=123,nick=max);role_c
```

The default syntax is

- roles are delimited by **;**
- a role parameter list starts with **(** and ends with **)**
- a role parameter is a key value pair delimited by **=**
- role parameters are delimited by **,**

16.1.6 Interaction between user/group and role services

The following section describes the interaction between the *User/group services* and the *Role services*.

Calculating the roles of a user

The diagram below illustrates how a user/group service and a role service interact to calculate user roles.

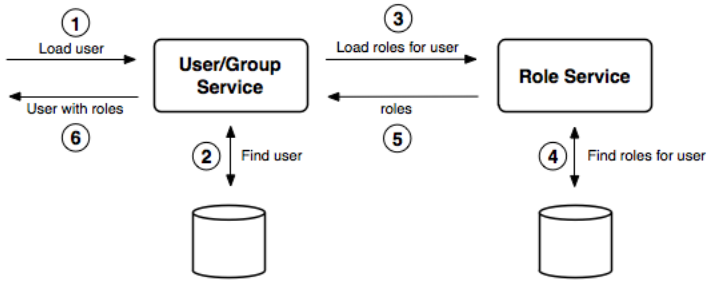


Figure 16.1: *User/group and role service interacting for role calculation*

On fetching an enabled user from a user/group service, the roles(s) assigned to that user must be identified. The identification procedure is:

1. Fetch all enabled groups for the user. If a group is disabled, it is discarded.
2. Fetch all roles associated with the user and add the roles to the result set.
3. For each enabled group the user is a member of, fetch all roles associated with the group and add the roles to the result set.
4. For each role in the result set, fetch all ancestor roles and add those roles to the result set.

5. Personalize each role in the result set as required.
6. If the result set contains the local admin role, add the role `ROLE_ADMINISTRATOR`.
7. If the result set contains the local group admin role, add the role `ROLE_GROUP_ADMIN`.

Note: Role personalization looks for role parameters (key/value pairs) for each role and checks if the user properties (key/value pairs) contain an identical key. If any matches are found, the value of the role parameter is replaced by the value of the user property.

Authentication of user credentials

A user/group service is primarily used during authentication. An authentication provider in the *Authentication chain* may use a user/group service to authenticate user credentials.

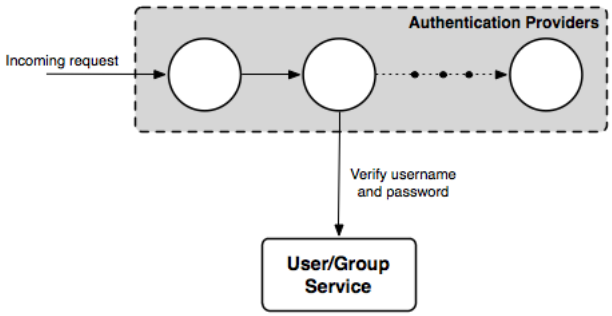


Figure 16.2: *Using a a user/group service for authentication*

GeoServer defaults

The following diagram illustrates the default user/group service, role service, and authentication provider in GeoServer:

Two authentication providers are configured—the *Root* provider and the *Username/password* provider. The *Root* provider authenticates for the GeoServer *Root account* and does not use a user/group service. The *Username/password* provider is the default provider and relays username and password credentials to a user/group service.

A single user/group service, which persist the user database as XML, is present. The database contains a single user named `admin` and no groups. Similarly, the role server persists the role database as XML. By default, this contains a single role named `ADMIN`, which is associated with the `admin` user. The `ADMIN` role is mapped to the `ROLE_ADMINISTRATOR` role and as a result, the `admin` user is associated with system administrator role during role calculation.

16.2 Authentication

There are three sets of GeoServer resources involved in authentication:

- The *Web Administration Interface* (also known as web admin)
- *OWS* services (such as WFS and WMS)

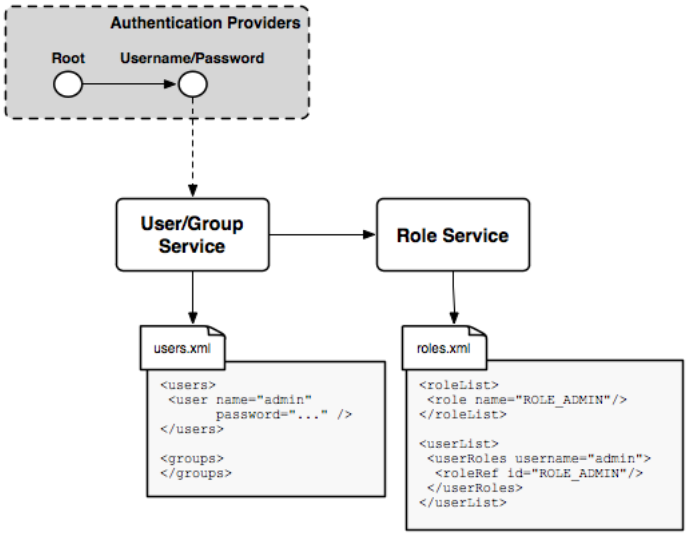


Figure 16.3: *Default GeoServer security configuration*

- *REST* services

The following sections describe how each set of GeoServer resources administers authentication. To configure the authentication settings and providers, please see the section on *Authentication* in the *Web Administration Interface*.

16.2.1 Authentication chain

Understanding the **authentication chain** helps explain how GeoServer authentication works. The authentication chain processes requests and applies certain authentication mechanisms. Examples of authentication mechanisms include:

- **Username/password**—Performs authentication by looking up user information in an external user database
- **Browser cookie**—Performs authentication by recognizing previously sent browser cookies (also known as "Remember Me")
- **LDAP**—Performs authentication against an LDAP database
- **Anonymous**—Essentially performs no authentication and allows a request to proceed without any credentials

Multiple authentication mechanisms may be active within GeoServer at a given time. The following figure illustrates the flow of a generic request.

Before dispatching a request to the appropriate service or handler, GeoServer first filters the request through the authentication chain. The request is passed to each mechanism in the chain in order, and each is given the chance to authenticate the request. If one of the mechanisms in the chain is able to successfully authenticate, the request moves to normal processing. Otherwise the request is not routed any further and an authorization error (usually a HTTP 401) is returned to the user.

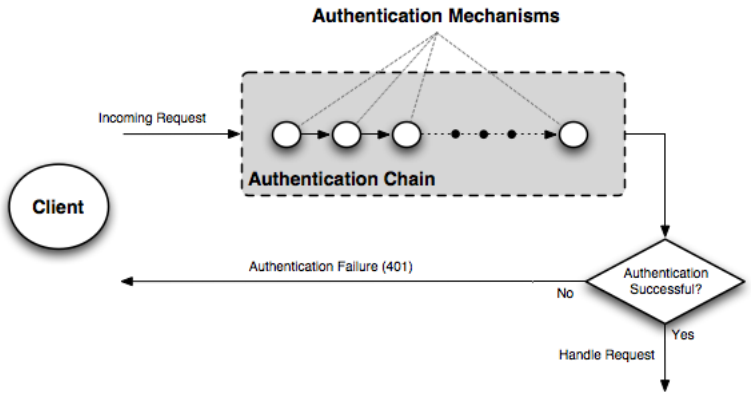


Figure 16.4: *Flow of a request through the authentication system*

Filter chain and provider chain

In the case of GeoServer, the authentication chain is actually made up of two chains: a **filter chain**, which determine if further authentication of a request is required, and a **provider chain**, which performs the actual authentication.

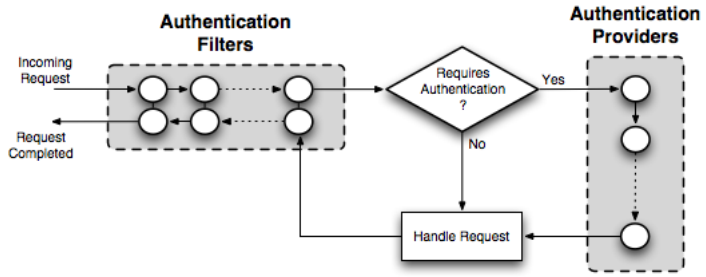


Figure 16.5: *Detail of authentication chain, showing filter chain and provider chain*

The filter chain performs a variety of tasks, including:

- Gathering user credentials from a request, for example from Basic and Digest Authentication headers
- Handling events such as ending the session (logging out), or setting the "Remember Me" browser cookie
- Performing session integration, detecting existing sessions and creating new sessions if necessary
- Invoking the authentication provider chain to perform actual authentication

The filter chain is actually processed twice, before and after the request is handled.

The provider chain is concerned solely with performing the underlying authentication of a request. It is invoked by the filter chain when a filter determines that authentication is required.

16.2.2 Authenticating to the Web Admin Interface

The method of authenticating to the *Web Administration Interface* application is typical of most web applications that provide login capabilities. The application is based primarily on form-based authentication, in

which a user authenticates through a form in a web browser. Upon successful authentication a session is created on the server, eliminating the need for a user to repeat the login process for each page they wish to access. An optional "Remember Me" setting is also supported which will store authentication information in a client-side cookie to allow the user to bypass the form-based authentication after the initial session has timed out.

The typical process of authentication is as follows:

1. User visits the home page of the web admin for the very first time, so neither a session or "Remember Me" cookie is present. In this case, the user is anonymously authenticated.
2. User accesses a secured page and is presented with a login form.
3. Upon successful login a session is created. Depending on the privileges of the account used to log in, the user will either be directed to the requested page or be redirected back to the home page.
4. Upon subsequent requests to secured pages, the user is authenticated via browser session until the session expires or the user logs out.

Examples

The following shows the default configuration of the authentication chain for the web admin.

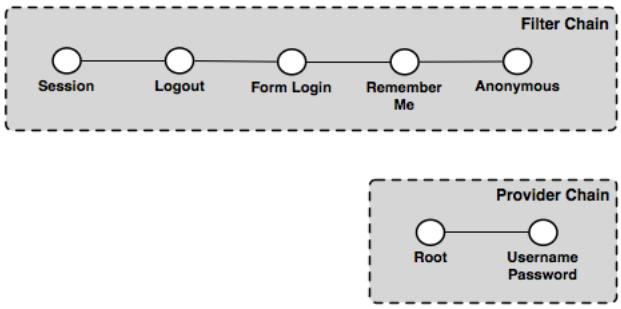


Figure 16.6: *GeoServer authentication chain, with filter and provider chains*

In this example the filter chain is made up of the following filters:

- **Session**—Handles session integration, recognizing existing sessions and creating new sessions on demand
- **Logout**—Handles ending sessions (user logout)
- **Form login**—Handles form logins
- **Remember Me**—Handles "Remember Me" authentication, reading when the flag is set on a form login, creating the appropriate cookie, and recognizing the cookie on future requests
- **Anonymous**—Handles anonymous access

The provider chain is made up of two providers:

- **Root**—The *Root account* has a special "super user" provider. As this account is rarely used, this provider is rarely invoked.
- **Username/password**—Performs username/password authentication against a user database.

To following example requests illustrate how the elements of the various chains work.

First time visit

This example describes the process when a user visits the home page of the web admin for the first time.

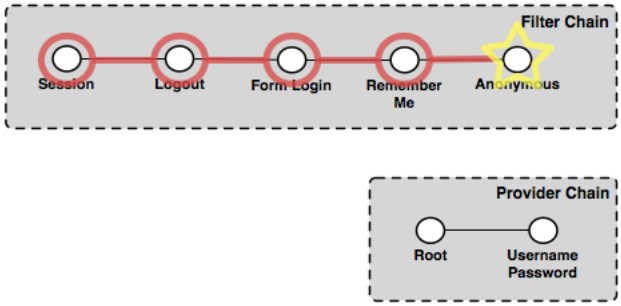


Figure 16.7: *Authentication chain for a first time visit from a user*

The first filter to execute is the *Session* filter. It checks for an existing session, but finds none, so processing continues to the next filter in the chain. The *Logout* filter checks for the case of a user logging out, which also is not the case, so processing continues. The *Form login* filter checks for a form login, and also finds none. The *Remember Me* filter determines if this request can be authenticated from a previous session cookie, but in this case it cannot. The final filter to execute is the *Anonymous* filter which checks if the user specified any credentials. In this case the user has not provided any credentials, so the request is authenticated anonymously. Since no authentication is required to view the home page, the provider chain is not invoked.

The last response to the request directs the user to the home page.

User logs on

This examples describes the process invoked when a user logs on to the web admin via the login form.

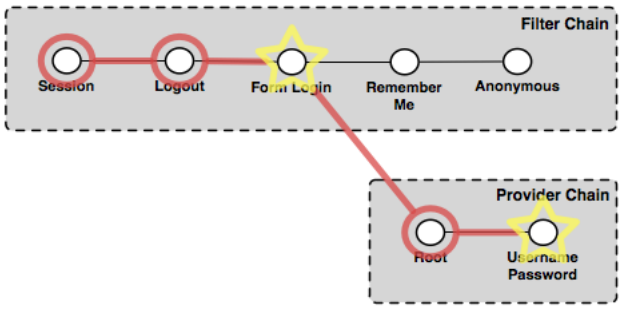


Figure 16.8: *Authentication chain for a user logging in*

The *Session* filter finds no existing session, and processing continues. The *Logout* filter checks for a logout request, finds none, and continues. The *Form login* filter recognizes the request as a form login and begins the authentication process. It extracts the username and password from the request and invokes the provider chain.

In the provider chain, the *Root* provider checks for the root account login, but doesn't find it so processing continues to the next provider. The *Username/password* provider checks if the supplied credentials are valid. If they are valid the authentication succeeds, user is redirected to the home page and is considered to be

logged on. During the post-processing step the *Session* filter recognizes that a successful authentication has taken place and creates a new session.

If the credentials are invalid, the user will be returned to the login form page and asked to try again.

User visits another page

This example describes the process invoked when a user who is already logged on visits another page in the web admin.

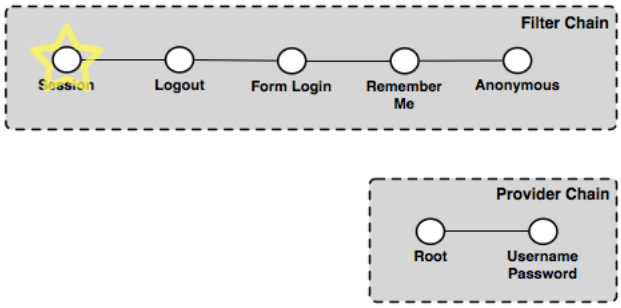


Figure 16.9: *Authentication chain for a user visiting another page after logging in*

The *Session* filter executes and finds an existing session that is still valid. The session contains the authentication details and no further chain processing is required. The response is the page requested by the user.

User returns after session time out

This example describes the process invoked when a user returns to the web admin after the previously created session has timed out.

A session will time out after a certain period of time. When the user returns to the web admin, this becomes essentially the same chain of events as the user visiting the web app for the first time (as described previously). The chain proceeds to the *Anonymous* filter that authenticates anonymously. Since the page requested is likely to be a page that requires authentication, the user is redirected to the home page and is not logged on.

User logs on with "Remember Me" flag set

This example describes the process for logging on with the "Remember Me" flag set.

The chain of events for logging on with "Remember Me" set is identical to the process for when the flag is not set, except that after the successful authentication the *Form login* filter recognizes the "Remember Me" flag and triggers the creation of the browser cookie used to persist the authentication information. The user is now logged on and is directed to the home page.

User returns after session time out (with "Remember Me")

This example describes the process invoked when the user returns to the web admin after a period of inactivity, while the "Remember Me" flag is set.

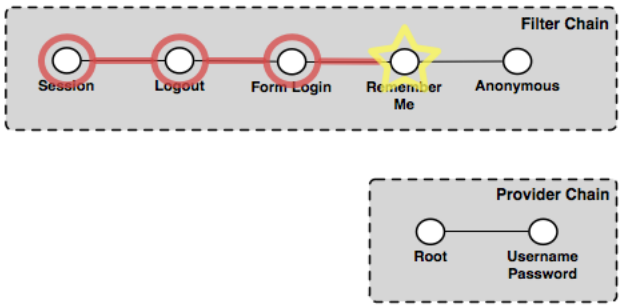


Figure 16.10: *Authentication chain for a user returning after session time out with the "Remember Me" flag*

Even though the "Remember Me" flag is set, the user's session on the server will still time out as normal. As such, the chain proceeds accordingly through the filters, starting with the *Session* filter, which finds no valid session. The *Logout* and *Form login* filters do not apply here. The *Remember Me* filter recognizes the browser cookie and is able to authenticate the request. The user is directed to whatever page was accessed and remains logged on.

16.2.3 Authentication to OWS and REST services

OWS and REST services are stateless and have no inherent awareness of "session", so the authentication scheme for these services requires the client to supply credentials on every request. That said, "session integration" is supported, meaning that if a session already exists on the server (from a concurrent *authenticated web admin session*) it will be used for authentication. This scheme allows GeoServer to avoid the overhead of session creation for OWS and REST services.

The default GeoServer configuration ships with support for HTTP Basic authentication for services.

The typical process of authentication is as follows:

1. User makes a service request without supplying any credentials
2. If the user is accessing an unsecured resource, the request is handled normally
3. If the user is accessing a secured resource:

- An HTTP 401 status code is sent back to the client, typically forcing the client to prompt for credentials.
- The service request is then repeated with the appropriate credentials included, usually in the HTTP header as with Basic Authentication.
- If the user has sufficient privileges to access the resource the request is handled normally, otherwise, a HTTP 404 status code is returned to the client.

4. Subsequent requests should include the original user credentials

Examples

The following describes the authentication chain for an OWS service:

In this example the filter chain consists of three filters:

- **Session**—Handles "session integration", recognizing existing sessions (but *not* creating new sessions)
- **Basic Auth**—Extracts Basic Authentication credentials from request HTTP header
- **Anonymous**—Handles anonymous access

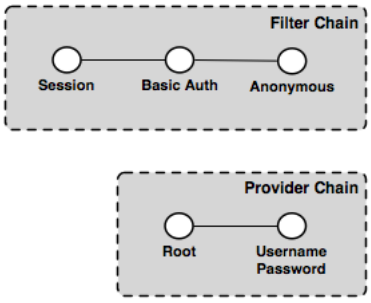


Figure 16.11: *The OWS service authentication chain*

The provider chain is made up of two providers:

- **Root**—*Root account* has a special "super user" provider. As this account is rarely used, this provider is rarely invoked.
- **Username/password**—Performs username/password authentication against a user database

To illustrate how the elements of the various chains work, here are some example OWS requests.

Anonymous WMS GetCapabilities request

This example shows the process for when a WMS client makes an anonymous GetCapabilities request.

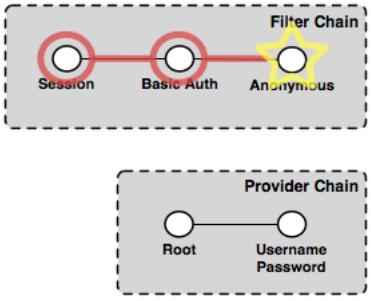


Figure 16.12: *Authentication chain for a WMS client making an anonymous GetCapabilities request*

The *Session* filter looks for an existing session, but finds none, so processing continues. The *Basic Auth* filter looks for the Basic Authorization header in the request, but as the request is anonymous, the filter finds none. Finally, the *Anonymous* filter executes and authenticates the request anonymously. Since GetCapabilities is a "discovery" operation it is typically not locked down, even on a secure server. Assuming this is the case here, the anonymous request succeeds, returning the capabilities response to the client. The provider chain is not invoked.

Anonymous WMS GetMap request for a secured layer

This example shows the process invoked when a WMS client makes an anonymous GetMap request for a secured layer

The chain executes exactly as described above. The *Session* filter looks for an existing session, but finds none, so processing continues. The *Basic Auth* filter looks for the Basic Authorization header in the request, but as

the request is anonymous, the filter finds none. Finally, the *Anonymous* filter executes and authenticates the request anonymously. However, in this case the layer being accessed is a secured resource, so the handling of the GetMap request fails. The server returns an exception accompanied with a HTTP 401 status code, which usually triggers the client presenting the user with a login dialog.

WMS GetMap request with user-supplied credentials

This example shows the process invoked when a WMS client gathers credentials from the user and reissues the previous request for a secured layer.

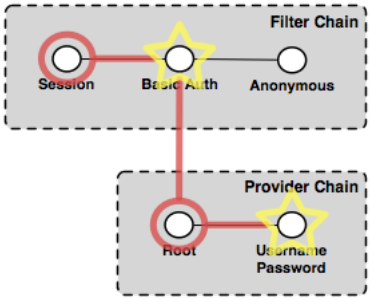


Figure 16.13: *Authentication chain for a WMS client making a GetMap request with user-supplied credentials*

The *Session* filter executes as described above, and does nothing. The *Basic Auth* filter finds the authorization header in the request, extracts the credentials for it, and invokes the provider chain. Processing moves to the *Username/password* provider that does the actual authentication. If the credentials have the necessary privileges to access the layer, the processing of the request continues normally and the GetMap request succeeds, returning the map response. If the credentials are not sufficient, the HTTP 401 status code will be supplied instead, which may again trigger the login dialog on the client side.

16.2.4 Authentication providers

The following authentication providers are available in GeoServer:

- Authentication of a username/password against a *user/group service*
- Authentication against an LDAP server
- Authentication by connecting to a database through JDBC

Username/password authentication

Username and password authentication is the default authentication provider. It uses a *user/group service* to authenticate.

The provider simply takes the username/password from an incoming request (such as a Basic Authentication request), then loads the user information from the user/group service and verifies the credentials.

LDAP authentication

The LDAP authentication provider allows for authentication against a Lightweight Directory Access Protocol (LDAP) server. The provider takes the username/password from the incoming request and attempts to connect to the LDAP server with those credentials.

Note: Currently only LDAP Bind authentication is supported.

Role assignment

The LDAP provider offers two options for role assignment for authenticated users:

- Convert the user's LDAP groups into roles
- Employ a user/group service

The following LDAP database will illustrate the first option:

```
dn: ou=people,dc=acme,dc=com
objectclass: organizationalUnit
ou: people

dn: uid=bob,ou=people,dc=acme,dc=com
objectclass: person
uid: bob

dn: ou=groups,dc=acme,dc=com
objectclass: organizationalUnit
ou: groups

dn: cn=workers,ou=groups,dc=acme,dc=com
objectclass: groupOfNames
cn: users
member: uid=bob,ou=people,dc=acme,dc=com
```

The above scenario defines a user with the `uid` of `bob`, and a `group` named `workers` of which `bob` is a member. After authentication, `bob` will be assigned the role `ROLE_WORKERS`. The role name is generated by concatenating `ROLE_` with the name of the group in upper case.

Note: When the LDAP server doesn't allow searching in an anoymous context, the bindBeforeGroupSearch option should be enabled to avoid errors.

In the case of using a *user/group service*, the user/group service is queried for the user following authentication, and the role assignment is performed by both the user/group service and the active *role service*. When using this option, any password defined for the user in the user/group service database is ignored.

Secure LDAP connections

There are two ways to create a secure LDAP connection with the server. The first is to directly specify a secure connection by using the **ldaps** protocol as part of the *Server URL*. This typically requires changing the connection port to **port 636** rather than 389.

The second method involves using **STARTTLS** (Transport Layer Security) to negotiate a secure connection over a non-secure one. The negotiation takes place over the non-secure URL using the "ldap" protocol on port 389. To use this option, the *Use TLS* flag must be set.

Warning: Using TLS for connections will prevent GeoServer from being able to pool LDAP connections. This means a new LDAP connection will be created and destroyed for each authentication, resulting in loss of performance.

JDBC authentication

The JDBC authentication provider authenticates by connecting to a database over JDBC.

The provider takes the username/password from the incoming request and attempts to create a database connection using those credentials. Optionally the provider may use a *user/group service* to load user information after a successful authentication. In this context the user/group service will not be used for password verification, only for role assignment.

Note: To use the user/group service for password verification, please see the section on *Username/password authentication*.

16.3 Passwords

Passwords are a central aspect of any security system. This section describes how GeoServer handles passwords.

16.3.1 Password encryption

A GeoServer configuration stores two types of passwords:

- Passwords for **user accounts** to access GeoServer resources
- Passwords used internally for **accessing external services** such as databases and cascading OGC services

As these passwords are typically stored on disk it is strongly recommended that they be encrypted and not stored as human-readable text. GeoServer security provides four schemes for encrypting passwords: **empty**, **plain text**, **Digest**, and **Password-based encryption (PBE)**.

The password encryption scheme is specified as a global setting that affects the encryption of passwords used for external resources, and as an encryption scheme for each *user/group service*. The encryption scheme for external resources has to be be *reversible*, while the user/group services can use any scheme.

Empty

The scheme is not reversible. Any password is encoded as an empty string, and as a consequence it is not possible to recalculate the plain text password. This scheme is used for user/group services in combination with an authentication mechanism using a back end system. Examples are user name/password authentication against a LDAP server or a JDBC database. In these scenarios, storing passwords locally to Geoserver does not make sense.

Plain text

Note: Prior to version 2.2.0, plain text encryption was the only available method used by GeoServer for storing passwords.

Plain text passwords provide no encryption at all. In this case, passwords are human-readable by anyone who has access to the file system. For obvious reasons, this is not recommended for any but the most basic test server. A password `mypassword` is encoded as `plain:mypassword`, the prefix uniquely describing the algorithm used for encoding/decoding.

Digest

Digest encryption is not reversible. It applies, 100,000 times through an iterative process, a SHA-256 cryptographic hash function to passwords. This scheme is "one-way" in that it is virtually impossible to reverse and obtain the original password from its hashed representation. Please see the section on *Reversible encryption* for more information on reversibility.

To protect from well known attacks, a random value called a salt is added to the password when generating the key. For each digesting, a separate random salt is used. Digesting the same password twice results in different hashed representations.

As an example, the password `geoserver` is digested to `digest1:YgaweuS60t+mJNobGlf9hzUC6g7gGTtPEu0TlnUxFl`
`digest1` indicates the usage of digesting. The hashed representation and the salt are base 64 encoded.

Password-based encryption

Password-based encryption (PBE) normally employs a user-supplied password to generate an encryption key. This scheme is reversible. A random salt described in the previous section is used.

Note: The system never uses passwords specified by users because these passwords tend to be weak. Passwords used for encryption are generated using a secure random generator and stored in the GeoServer key store. The number of possible passwords is 2^260 .

GeoServer supports two forms of PBE. **Weak PBE** (the GeoServer default) uses a basic encryption method that is relatively easy to crack. The encryption key is derived from the password using MD5 1000 times iteratively. The encryption algorithm itself is DES (Data Encryption Standard). DES has an effective key length of 56 bits, which is not really a challenge for computer systems in these days.

Strong PBE uses a much stronger encryption method based on an AES 256-bit algorithm with CBC. The key length is 256 bit and is derived using SHA-256 instead of MD5. Using Strong PBE is highly recommended.

As an example, the password `geoserver` is encrypted to `crypt1:KWhO7jrTz/Gi0oTQRKsVeCmWIZY5VZaD`.
`crypt1` indicates the usage of Weak PBE. The prefix for Strong PBE is `crypt2`. The ciphertext and the salt are base 64 encoded.

Note: Strong PBE is not natively available on all Java virtual machines and may require the installation of some additional JCE Unlimited Strength Jurisdiction policy files:

- Oracle JCE policy jars for Oracle JVM
- IBM JCE policy jars for IBM JVM

Reversible encryption

Password encryption methods can be **reversible**, meaning that it is possible (and desirable) to obtain the plain-text password from its encrypted version. Reversible passwords are necessary for database connections or external OGC services such as *cascading WMS* and *cascading WFS*, since GeoServer must be able to decode the encrypted password and pass it to the external service. Plain text and PBE passwords are reversible.

Non-reversible passwords provide the highest level of security, and therefore should be used for user accounts and wherever else possible. Using password digesting is highly recommended, the installation of the unrestricted policy files is not required.

16.3.2 Secret keys and the keystore

For a reversible password to provide a meaningful level of security, access to the password must be restricted in some way. In GeoServer, encrypting and decrypting passwords involves the generation of secret shared keys, stored in a typical Java *keystore*. GeoServer uses its own keystore for this purpose named `geoserver.jceks` which is located in the `security` directory in the GeoServer data directory. This file is stored in the JCEKS format rather than the default JKS. JKS does not support storing shared keys.

The GeoServer keystore is password protected with a *Master password*. It is possible to access the contents of the keystore with external tools such as keytool. For example, this following command would prompt for the master password and list the contents of the keystore:

```
$ keytools -list -keystore geoserver.jceks -storetype "JCEKS"
```

16.3.3 Master password

It is also possible to set a **master password** for GeoServer. This password serves two purposes:

- Protect access to the *keystore*
- Protect access to the GeoServer *Root account*

By default, the master password is generated and stored in a file named `security/masterpw.info` using plain text. When upgrading from an existing GeoServer data directory (versions 2.1.x and lower), the algorithm attempts to figure out the password of a user with the role `ROLE_ADMINISTRATOR`. If such a password is found and the password length is 8 characters at minimum, GeoServer uses this password as master password. Again, the name of the chosen user is found in `security/masterpw.info`.

> **Warning:** The file `security/masterpw.info` is a security risk. The administrator should read this file and verify the master password by logging on GeoServer as the `root` user. On success, this file should be removed.

Refer to *Active master password provider* for information on how to change the master password.

16.3.4 Password policies

A password policy defines constraints on passwords such as password length, case, and required mix of character classes. Password policies are specified when adding *User/group services* and are used to constrain passwords when creating new users and when changing passwords of existing users.

Each user/group service uses a password policy to enforce these rules. The default GeoServer password policy implementation supports the following optional constraints:

- Passwords must contain at least one number
- Passwords must contain at least one upper case letter
- Passwords must contain at least one lower case letter
- Password minimum length
- Password maximum length

16.4 Root account

The highly configurable nature of GeoServer security may result in an administrator inadvertently disrupting normal authentication, essentially disabling all users including administrative accounts. For this reason, the GeoServer security subsystem contains a **root account** that is always active, regardless of the state of the security configuration. Much like its UNIX-style counterpart, this account provides "super user" status, and is meant to provide an alternative access method for fixing configuration issues.

The user name for the root account is `root`. Its name cannot be changed and the password for the root account is the *Master password*.

16.5 Service Security

GeoServer supports access control at the service level, allowing for the locking down of service operations to only authenticated users who have been granted a particular role. There are two main categories of services in GeoServer. The first is *OWS services* such as WMS and WFS. The second are RESTful services, such as the GeoServer *REST configuration*.

Note: Service-level security and *Layer security* cannot be combined. For example, it is not possible to specify access to a specific OWS service only for one specific layer.

16.5.1 OWS services

OWS services support setting security access constraints globally for a particular service, or to a specific operation within that service. A few examples include:

- Securing the entire WFS service so only authenticated users have access to all WFS operations.
- Allowing anonymous access to read-only WFS operations such as GetCapabilities, but securing write operations such as Transaction.
- Disabling the WFS service in effect by securing all operations and not applying the appropriate roles to any users.

OWS service security access rules are specified in a file named `services.properties`, located in the `security` directory in the GeoServer data directory. The file contains a list of rules that map service operations to defined roles. The syntax for specifying rules is as follows:

```
<service>.<operation|*>=<role>[,<role2>,...]
```

The parameters include:

- `[]`—Denotes optional parameters
- `|`—Denotes "or"
- `service`—Identifier of an OGC service, such as `wfs`, `wms`, or `wcs`
- `operation`—Any operation supported by the service, examples include `GetFeature` for WFS, `GetMap` for WMS, `*` for all operations
- `role[,role2,...]`—List of predefined role names

Note: It is important that roles specified are actually linked to a user, otherwise the whole service/operation will be accessible to no one except for the *Root account*. However in some cases this may be the desired effect.

The default service security configuration in GeoServer contains no rules and allows any anonymous user to access any operation of any service. The following are some examples of desired security restrictions and the corresponding rules.

Securing the entire WFS service

This rule grants access to any WFS operation only to authenticated users that have been granted the `ROLE_WFS` role:

```
wfs.*=ROLE_WFS
```

Anonymous WFS access only for read-only operations

This rule grants anonymous access to all WFS operations (such as GetCapabilities and GetFeature) but restricts Transaction requests to authenticated users that have been granted the `ROLE_WFS_WRITE` role:

```
wfs.Transaction=ROLE_WFS_WRITE
```

Securing data-accessing WFS operations and write operations

Used in conjunction, these two rules grant anonymous access to GetCapabilities and DescribeFeatureType, forcing the user to authenticate for the GetFeature operation (must be granted the `ROLE_WFS_READ` role), and to authenticate to perform transactions (must be granted the `ROLE_WFS_WRITE` role:

```
wfs.GetFeature=ROLE_WFS_READ
wfs.Transaction=ROLE_WFS_WRITE
```

Note this example does not specify whether a user accessing Transactions would also have access to GetFeature.

16.5.2 REST services

In addition to providing the ability to secure OWS services, GeoServer also allows for the securing of RESTful services.

REST service security access rules are specified in a file named `rest.properties`, located in the `security` directory of the GeoServer data directory. This file contains a list of rules mapping request URIs to defined roles. The rule syntax is as follows:

```
<uriPattern>;<method>[,<method>,...]=<role>[,<role>,...]
```

The parameters include:

- `[]`—Denote optional parameters
- `uriPattern`—The *ant pattern* that matches a set of request URIs
- `method`—HTTP request method, one of `GET`, `POST`, `PUT`, `POST`, `DELETE`, or `HEAD`
- `role`—Name of a predefined role. The wildcard `*` is used to indicate all users, including anonymous users.

Note:

- URI patterns should account for the first component of the rest path, usually `rest` or `api`
- `method` and `role` lists should **not** contain any spaces

Ant patterns

Ant patterns are commonly used for pattern matching directory and file paths. The following examples provide some basic instructions. The Apache ant user manual contains more sophisticated use cases.

These examples are specific to GeoServer *REST configuration*, but any RESTful GeoServer service could be configured in the same manner.

Disabling anonymous access to services

The most secure of configurations is one that forces any request, REST or otherwise, to be authenticated. The following will lock down access to all requests to users that are granted the `ROLE_ADMINISTRATOR` role:

```
/**;GET,POST,PUT,DELETE=ROLE_ADMINISTRATOR
```

A less restricting configuration locks down access to operations under the path `/rest` to users granted the `ROLE_ADMINISTRATOR` role, but will allow anonymous access to requests that fall under other paths (for example `/api`):

```
/rest/**;GET,POST,PUT,DELETE=ROLE_ADMINISTRATOR
```

Allowing anonymous read-only access

The following configuration grants anonymous access when the `GET` method is used, but forces authentication for a `POST`, `PUT`, or `DELETE` method:

```
/**;GET=IS_AUTHENTICATED_ANONYMOUSLY
/**;POST,PUT,DELETE=TRUSTED_ROLE
```

Securing a specific resource

The following configuration forces authentication for access to a particular resource (in this case the `states` feature type):

```
/rest/**/states*;GET=TRUSTED_ROLE
/rest/**;POST,PUT,DELETE=TRUSTED_ROLE
```

The following secures access to a set of resources (in this case all data stores).:

```
/rest/**/datastores/*;GET=TRUSTED_ROLE
/rest/**/datastores/*.*;GET=TRUSTED_ROLE
/rest/**;POST,PUT,DELETE=TRUSTED_ROLE
```

Note the trailing wildcards `/*` and `/*.*`.

16.6 Layer security

GeoServer allows access to be determined on a per-layer basis.

Note: Layer security and *Service Security* cannot be combined. For example, it is not possible to specify access to a specific OWS service, only for one specific layer.

Providing access to layers is linked to *roles*. Layers and roles are linked in a file called `layers.properties`, which is located in the `security` directory in your GeoServer data directory. The file contains the rules that control access to workspaces and layers.

16.6.1 Rules

The syntax for a layer security rule is as follows (`[]` denotes optional parameters):

```
workspace.layer.permission=role[,role2,...]
```

The parameters include:

```
* ''workspace''--Name of the workspace. The wildcard ''*'' is used to indicate all workspaces.
* ''layer''--Name of a resource (featuretype/coverage/etc...). The wildcard ''*'' is used to indicate
* ''permission''--Type of access permission/mode.
```

  - `r`—Read access
  - `w`—Write access
  - `a`—Admin access

  See *Access modes* for more details.

- `role[,role2,...]` is the name(s) of predefined roles. The wildcard `*` is used to indicate the permission is applied to all users, including anonymous users.

Note: If a workspace or layer name is supposed to contain dots, they can be escaped using double backslashes (`\\`). For example, if a layer is named `layer.with.dots` the following syntax for a rule may be used:

```
topp.layer\\.with\\.dots.r=role[,role2,...]
```

Each entry must have a unique combination of workspace, layer, and permission values. If a permission at the global level is not specified, global permissions are assumed to allow read/write access. If a permission for a workspace is not specified, it inherits permissions from the global specification. If a permission for a layer is not specified, it inherits permissions from its workspace specification. If a user belongs to multiple roles, the **least restrictive** permission they inherit will apply.

16.6.2 Catalog Mode

The `layers.properties` file may contain a further directive that specifies how GeoServer will advertise secured layers and behave when a secured layer is accessed without the necessary privileges. The parameter is `mode` and is commonly referred to as the "catalog mode".

The syntax is:

```
mode=option
```

`option` may be one of three values:

Option	Description
`hide`	*(Default)* Hides layers that the user does not have read access to, and behaves as if a layer is read only if the user does not have write permissions. The capabilities documents will not contain the layers the current user cannot access. This is the highest security mode. As a result, it may not work very well with clients such as uDig or Google Earth.
`challenge`	Allows free access to metadata, but any attempt at accessing actual data is met by a HTTP 401 code (which forces most clients to show an authentication dialog). The capabilities documents contain the full list of layers. DescribeFeatureType and DescribeCoverage operations work successfully. This mode works fine with clients such as uDig or Google Earth.
`mixed`	Hides the layers the user cannot read from the capabilities documents, but triggers authentication for any other attempt to access the data or the metadata. This option is useful if you don't want the world to see the existence of some of your data, but you still want selected people to who have data access links to get the data after authentication.

16.6.3 Access modes

The access mode defines what level of access should be granted on a specific workspace/layer to a particular role. There are three types of access mode:

- `r`—**Read mode** (read data from a workspace/layer)
- `w`—**Write mode** (write data to a workspace/layer)
- `a`—**Admin mode** (access and modify the configuration of a workspace/layer)

Some notes on the above access modes:

- Write does not imply Read, but Admin implies both Write *and* Read.
- Read and Write apply to the data of a layer, while Admin applies to the configuration of a layer.
- As Admin mode only refers to the configuration of the layer, it is not required for any OGC service request.

Note: Currently, it is possible to assign Admin permission only to an entire workspace, and not to specific layers.

16.6.4 Examples

The following examples illustrate some possible layer restrictions and the corresponding rules.

Protecting a single workspace and a single layer

The following example demonstrates how to configure GeoServer as a primarily a read-only server:

```
*.*.r=*
*.*.w=NO_ONE
private.*.r=TRUSTED_ROLE
private.*.w=TRUSTED_ROLE
topp.congress_district.w=STATE_LEGISLATORS
```

The mapping of roles to permissions is as follows:

Role	private.*	topp.*	topp.congress_district	(all other workspaces)
NO_ONE	(none)	w	(none)	w
TRUSTED_ROLE	r/w	r	r	r
STATE_LEGISLATURES	(none)	r	r/w	r
(All other users)	r	r	r	r

Locking down GeoServer

The following example demonstrates how to lock down GeoServer:

```
*.*.r=TRUSTED_ROLE
*.*.w=TRUSTED_ROLE
topp.*.r=*
army.*.r=MILITARY_ROLE,TRUSTED_ROLE
army.*.w=MILITARY_ROLE,TRUSTED_ROLE
```

The mapping of roles to permissions is as follows:

Role	topp.*	army.*	(All other workspaces)
TRUSTED_ROLE	r/w	r/w	r/w
MILITARY_ROLE	r	r/w	(none)
(All other users)	r	(none)	(none)

Providing restricted administrative access

The following provides administrative access on a single workspace to a specific role, in additional to the full administrator role:

```
*.*.a=ROLE_ADMINISTRATOR
topp.*.a=ROLE_TOPP_ADMIN,ROLE_ADMINISTRATOR
```

Managing multi-level permissions

The following example demonstrates how to configure GeoServer with global-, workspace–, and layer-level permissions:

```
*.*.r=TRUSTED_ROLE
*.*.w=NO_ONE
topp.*.r=*
topp.states.r=USA_CITIZEN_ROLE,LAND_MANAGER_ROLE,TRUSTED_ROLE
topp.states.w=NO_ONE
topp.poly_landmarks.w=LAND_MANAGER_ROLE
topp.military_bases.r=MILITARY_ROLE
topp.military_bases.w=MILITARY_ROLE
```

The mapping of roles to permissions is as follows:

Role	topp.states	topp.poly_landmarks	topp.military_bases	topp.(all other layers)	(All other workspaces)
NO_ONE	w	r	(none)	w	w
TRUSTED_ROLE	r	r	(none)	r	r
MILITARY_ROLE	(none)	r	r/w	r	(none)
USA_CITIZEN_ROLE	r	r	(none)	r	(none)
LAND_MANAGER_ROLE	r	r/w	(none)	r	(none)
(All other users)	(none)	r	(none)	r	(none)

Note: The entry `topp.states.w=NO_ONE` is not required because this permission would be inherited from the global level (the entry `*.*.w=NO_ONE`).

Invalid configuration

The following examples are invalid because the workspace, layer, and permission combinations are not unique:

```
topp.state.rw=ROLE1
topp.state.rw=ROLE2,ROLE3
```

16.7 REST Security

In addition to providing the ability to secure OWS style services, GeoServer also supports securing RESTful services.

As with layer and service security, RESTful security configuration is based on `sec_roles`. The mapping of request URI to role is defined in a file named `rest.properties`, located in the `security` directory of the GeoServer data directory.

16.7.1 Syntax

The following syntax defines access control rules for RESTful services (parameters in brackets [] are optional):

```
uriPattern;method[,method,...]=role[,role,...]
```

The parameters are:

- **uriPattern**—*ant pattern* that matches a set of request URIs
- **method**—HTTP request method, one of `GET`, `POST`, `PUT`, `POST`, `DELETE`, or `HEAD`
- **role**—Name of a predefined role. The wildcard '* is used to indicate the permission is applied to all users, including anonymous users.

Note:

- URI patterns should account for the first component of the rest path, usually `rest` or `api`
- Method and role lists should **not** contain any spaces

Ant patterns

Ant patterns are commonly used for pattern matching directory and file paths. The *examples* section contains some basic instructions. The Apache ant user manual contains more sophisticated use cases.

16.7.2 Examples

Most of the examples in this section are specific to the GeoServer *REST configuration* but any RESTful GeoServer service may be configured in the same manner.

Allowing only authenticated access

The most secure configuration is one that forces any request to be authenticated. The following example locks down access to all requests:

```
/**;GET,POST,PUT,DELETE=ROLE_ADMINISTRATOR
```

A less restricting configuration locks down access to operations under the path `/rest`, but will allow anonymous access to requests that fall under other paths (for example `/api`):

```
/rest/**;GET,POST,PUT,DELETE=ROLE_ADMINISTRATOR
```

The following configuration is similar to the previous one except it grants access to a specific role rather than the administrator:

```
/**;GET,POST,PUT,DELETE=ROLE_TRUSTED
```

`ROLE_TRUSTED` is a role defined in `users.properties`.

Providing anonymous read-only access

The following configuration allows anonymous access when the `GET` (read) method is used but forces authentication for a `POST`, `PUT`, or `DELETE` (write):

```
/**;GET=IS_AUTHENTICATED_ANONYMOUSLY
/**;POST,PUT,DELETE=TRUSTED_ROLE
```

Securing a specific resource

The following configuration forces authentication for access to a particular resource (in this case a feature type):

```
/rest/**/states*;GET=TRUSTED_ROLE
/rest/**;POST,PUT,DELETE=TRUSTED_ROLE
```

The following secures access to a set of resources (in this case all data stores):

```
/rest/**/datastores/*;GET=TRUSTED_ROLE
/rest/**/datastores/*.*;GET=TRUSTED_ROLE
/rest/**;POST,PUT,DELETE=TRUSTED_ROLE
```

16.8 Disabling security

If you are using an external security subsystem, you may want to disable the built-in security to prevent conflicts. Disabling security is possible for each security filter chain individually. The security filter chains are listed on the GeoServer authentication page.

Warning: Disabling security for a filter chain results in administrator privileges for each HTTP request matching this chain. As an example, disabling security on the **web** chain gives administrative access to each user accessing the *Web Administration Interface* interface.

16.9 Tutorials

16.9.1 Authentication with LDAP

This tutorial introduces GeoServer LDAP support and walks through the process of setting up authentication aganist an LDAP server. It is recommended that the *LDAP authentication* section be read before proceeding.

LDAP server setup

A mock LDAP server will be used for this tutorial. Download and run the acme-ldap jar:

```
java -jar acme-ldap.jar
```

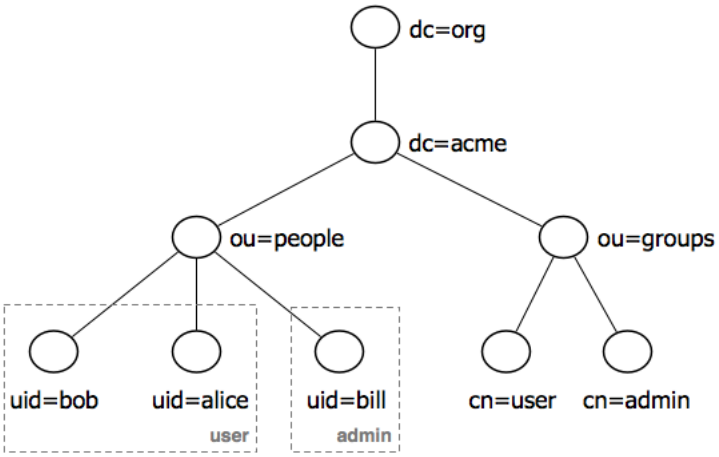
The output of which should look like the following:

```
Directory contents:
  ou=people,dc=acme,dc=org
    uid=bob,ou=people,dc=acme,dc=org
    uid=alice,ou=people,dc=acme,dc=org
    uid=bill,ou=people,dc=acme,dc=org
  ou=groups,dc=acme,dc=org
  cn=users,ou=groups,dc=acme,dc=org
    member: uid=bob,ou=people,dc=acme,dc=org
    member: uid=alice,ou=people,dc=acme,dc=org
  cn=admins,ou=groups,dc=acme,dc=org
    member: uid=bill,ou=people,dc=acme,dc=org

  Server running on port 10389
```

The following diagram illustrates the hierachy of the LDAP datatabse:

The LDAP tree consists of:

- The root domain component, `dc=acme,dc=org`
- Two organizational units (groups) named `user` and `admin`
- Two users named `bob` and `alice` who are members of the `user` group
- One user named `bill` who is a member of the `admin` group



Configure the LDAP authentication provider

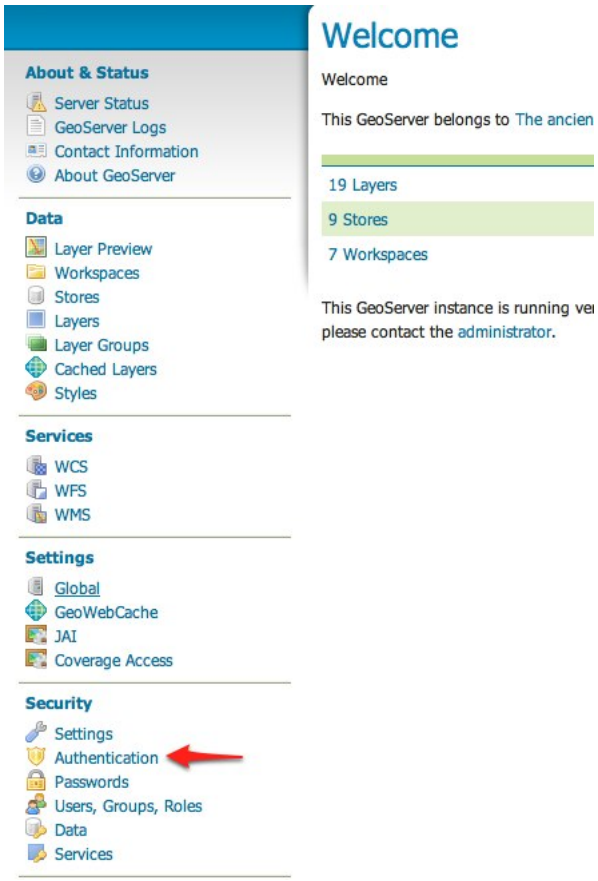
1. Start GeoServer and login to the web admin interface as the `admin` user.
2. Click the `Authentication` link located under the `Security` section of the navigation sidebar.
3. Scroll down to the `Authentication Providers` panel and click the `Add new` link.
4. Click the `LDAP` link.
5. Fill in the fields of the settings form as follows:
   - Set `Name` to "acme-ldap"
   - Set `Server URL` to "ldap://localhost:10389/dc=acme,dc=org"
   - Set `User lookup pattern` to "uid={0},ou=people"
6. Test the LDAP connection by entering the username "bob" and password "secret" in the connection test form located on the right and click the `Test Connection` button.

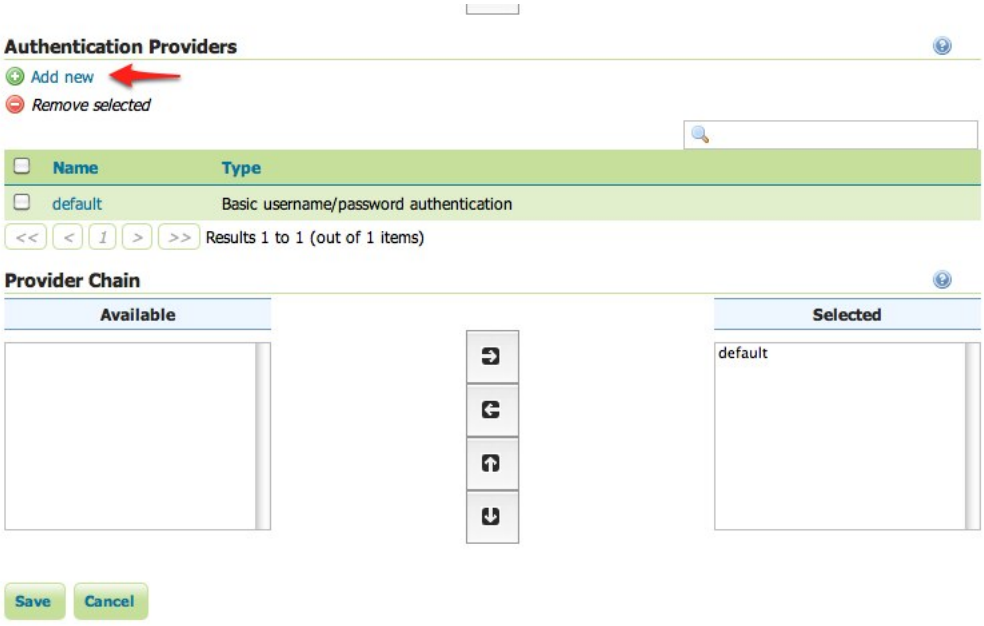
   A successful connection should be reported at the top of the page.
7. Save.
8. Back on the authentication page scroll down to the `Provider Chain` panel and move the `acme-ldap` provider from `Available` to `Selected`.
9. Save.

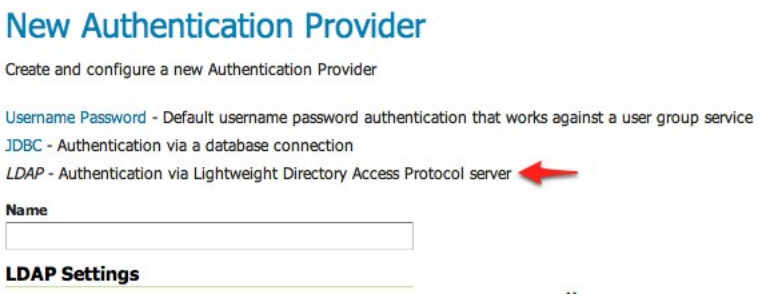
Test a LDAP login

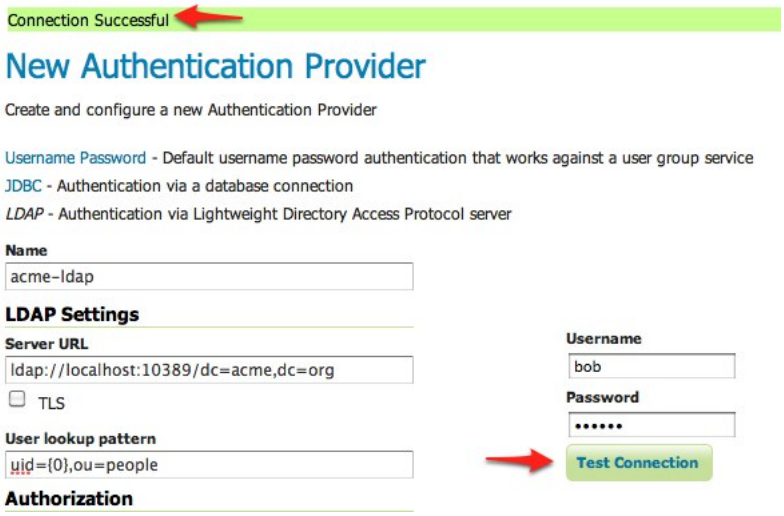
1. Navigate to the GeoServer home page and log out of the admin account.
2. Login as the user "bob" with the with the password "secret".

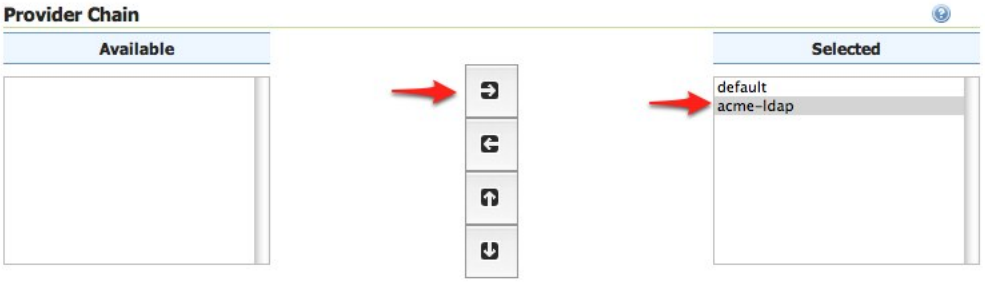
Logging in as bob doesn't yield any administrative functionality because the bobaccount has not been mapped to the administrator role. In the next section GeoServer will be configured to map groups from the LDAP database to roles.

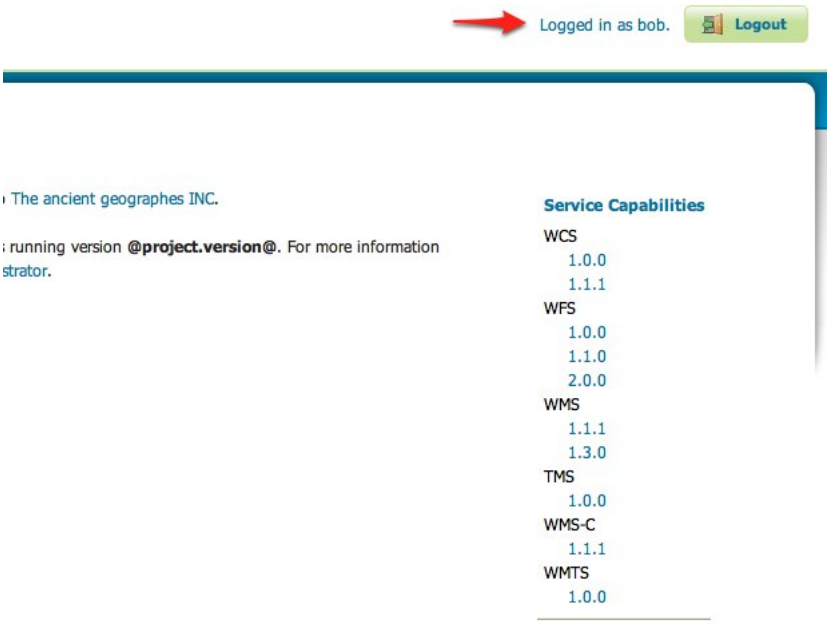

Welcome
About & Status
Server Status
GeoServer Logs
Contact Information
About GeoServer
Data
Layer Preview
Workspaces
Stores
Layers
Layer Groups
Cached Layers
Styles
Services
WCS
WFS
WMS
Settings
Global
GeoWebCache
JAI
Coverage Access
Security
Settings
Authentication
Passwords
Users, Groups, Roles
Data
Services
Welcome
This GeoServer belongs to The ancien
19 Layers
9 Stores
7 Workspaces
This GeoServer instance is running ver
please contact the administrator.


Authentication Providers
Add new
Remove selected
Name
Type
default
Basic username/password authentication
<< < 1 > >> Results 1 to 1 (out of 1 items)
Provider Chain
Available
Selected
default
Save
Cancel


New Authentication Provider
Create and configure a new Authentication Provider
Username Password - Default username password authentication that works against a user group service
JDBC - Authentication via a database connection
LDAP - Authentication via Lightweight Directory Access Protocol server
Name
LDAP Settings


Connection Successful
New Authentication Provider
Create and configure a new Authentication Provider
Username Password - Default username password authentication that works against a user group service
JDBC - Authentication via a database connection
LDAP - Authentication via Lightweight Directory Access Protocol server
Name
acme-ldap
LDAP Settings
Server URL
ldap://localhost:10389/dc=acme,dc=org
TLS
User lookup pattern
uid={0},ou=people
Authorization
Username
bob
Password
Test Connection

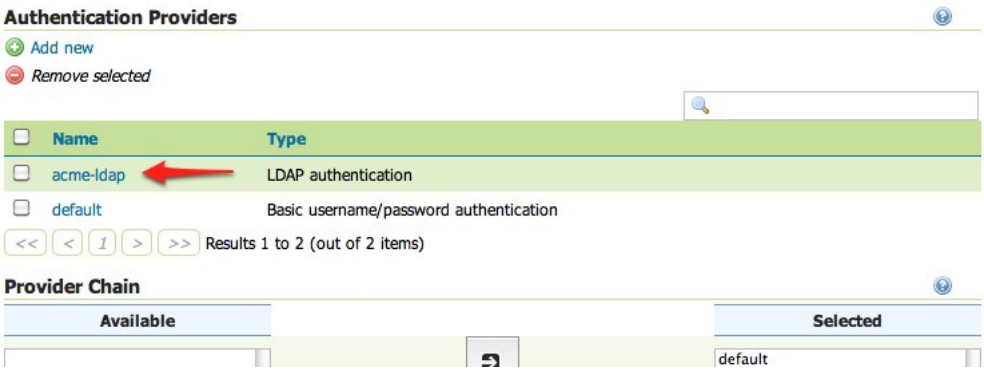

Provider Chain
Available
Selected
default
acme-ldap



Map LDAP groups to GeoServer roles

When using LDAP for authentication GeoServer maps LDAP groups to GeoServer roles by prefixing the group name with `ROLE_` and converting the result to uppercase. For example bob and alice are members of the `user` group so after authentication they would be assigned a role named `ROLE_USER`. Similarily bill is a member of the `admin` group so he would be assigned a role named `ROLE_ADMIN`.

1. Log out of the web admin and log back in as the admin user.
2. Navigate to the `Authentication` page.
3. Scroll to the `Authentication Providers` panel and click the `acme-ldap` link.



4. On the settings page fill in the following form fields:
   - Set `Group search base` to "ou=groups"
   - Set `Group search filter` to "member={0}"

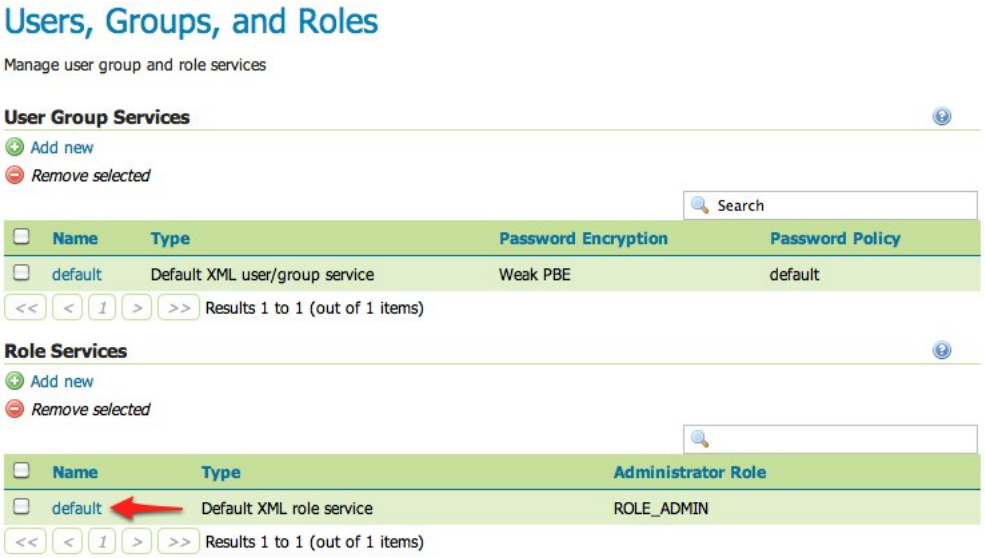
   The first field specifies the node of the LDAP directory tree at which groups are located. In this case the organizational unit named `groups`. The second field specifies the LDAP query filter to use in order to locate those groups that a specific user is a member of. The `{0}` is a placeholder which is replaced with the `uid` of the user.

5. Save.

At this point the LDAP provider will populate an authenticated user with roles based on the groups the user is a member of. But the GeoServer administrative role is named `ROLE_ADMINISTRATOR`. Therefore even bill who is assigned the role `ROLE_ADMIN` will not be granted administrative rights. To remedy this the GeoServer role service will be reconfigured to treat `ROLE_ADMIN` as an adminstrative role.

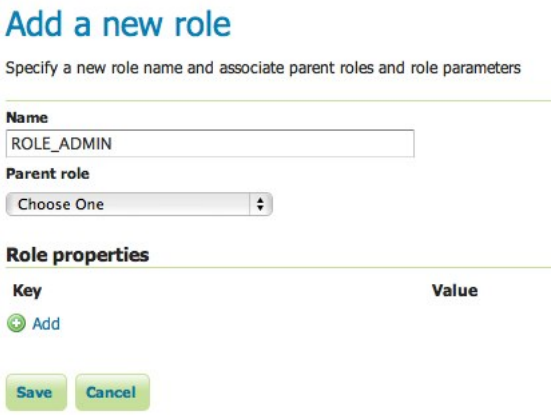
1. Click the `Users,Group,Roles` link located under the `Security` section of the navigation sidebar.

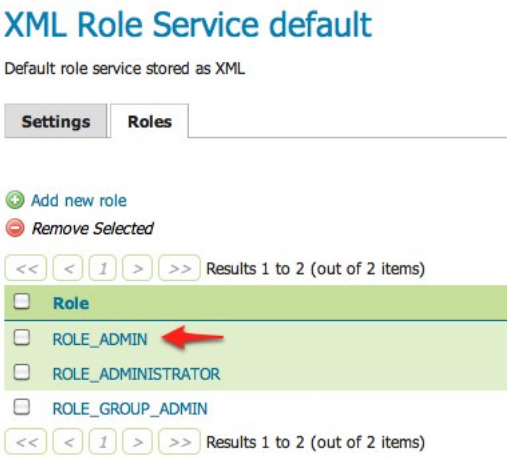
2. Scroll to the `Role Services` panel and click the `default` link.

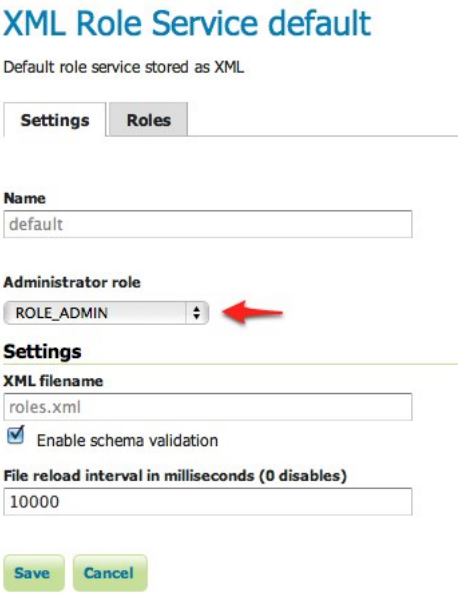


3. Switch to the `Roles` tab.
4. Add a new role named `ROLE_ADMIN`.
5. Save.
6. Switch to the `Settings` tab.
7. Select `ROLE_ADMIN` from the `Administrator role` drop down.
8. Save.

At this point members of the `admin` LDAP group should be given full administrative privileges once authenticated. Log out of the admin account and log in as "bill" with the password "hello". Once logged in full administrative functionality should be available.


Add a new role
Specify a new role name and associate parent roles and role parameters
Name
ROLE_ADMIN
Parent role
Choose One
Role properties
Key
Value
Add
Save
Cancel

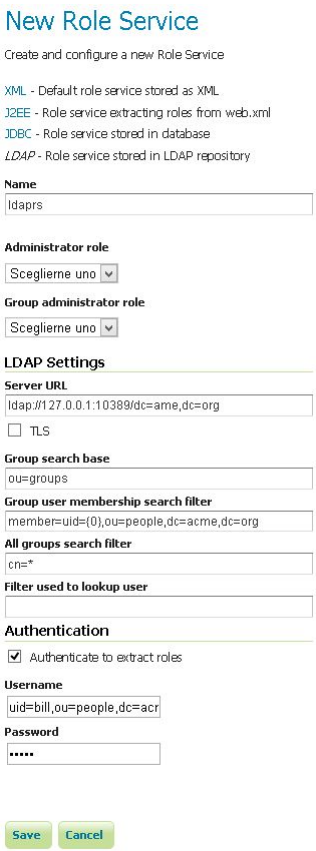

XML Role Service default
Default role service stored as XML
Settings
Roles
Add new role
Remove Selected
<< < 1 > >> Results 1 to 2 (out of 2 items)
Role
ROLE_ADMIN
ROLE_ADMINISTRATOR
ROLE_GROUP_ADMIN
<< < 1 > >> Results 1 to 2 (out of 2 items)


XML Role Service default
Default role service stored as XML
Settings
Roles
Name
default
Administrator role
ROLE_ADMIN
Settings
XML filename
roles.xml
Enable schema validation
File reload interval in milliseconds (0 disables)
10000
Save
Cancel

Configure the LDAP role service

An additional step permits to configure a role service to get GeoServer roles from the LDAP repository and allow access rights to be assigned to those roles.

1. Click the `Users,Group,Roles` link located under the `Security` section of the navigation sidebar.
2. Click the `Add new link` under the `Role Services` section.
3. Click the `LDAP` option under the `New Role Service` section.



4. Enter `ldaprs` in the `Name` text field.
5. Enter `ldap://127.0.0.1:10389/dc=acme,dc=org` in the `Server URL` text field.
6. Enter `ou=groups` in the `Group search base` text field.
7. Enter `member=uid={0},ou=people,dc=acme,dc=org` in the `Group user membership search filter` text field.
8. Enter `cn=*` in the `All groups search filter` text field.

Then we need to a choose a user to authenticate on the server (many LDAP server don't allow anonymous data lookup).

1. Check the `Authenticate to extract roles` checkbox.
2. Enter `uid=bill,ou=people,dc=acme,dc=org` in the `Username` text field.
3. Enter `hello` in the `Password` text field.
4. Save.

5. Click the `ldaprs` role service item under the `Role Services` section.
6. Select `ROLE_ADMIN` from the `Administrator role` combobox.
7. Select `ROLE_ADMIN` from the `Group administrator role` combobox.
8. Save again.

You should now be able to see and assign the new `ROLE_ADMIN` and `ROLE_USER` roles wherever an `Available Roles` list is shown (for example in the `Data` and `Services` rules sections.

16.9.2 Authentication with LDAP against ActiveDirectory

This tutorial explains how to use GeoServer LDAP support to connect to a Windows Domain using ActiveDirectory as an LDAP server. It is recommended that the *LDAP authentication* section be read before proceeding.

Windows Server and ActiveDirectory

Active Directory is just another LDAP server implementation, but has some features that we must know to successfully use it with GeoServer LDAP authentication. In this tutorial we will assume to have a Windows Server Domain Controller with ActiveDirectory named `domain-controller` for a domain named `ad.local`. If your environment uses different names (and it surely will) use your real names where needed.

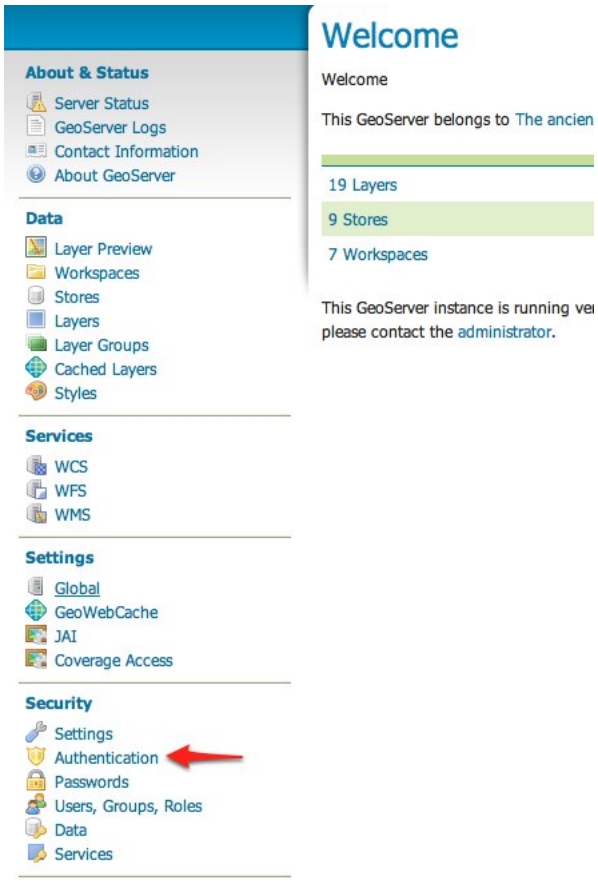
We will also assume that:

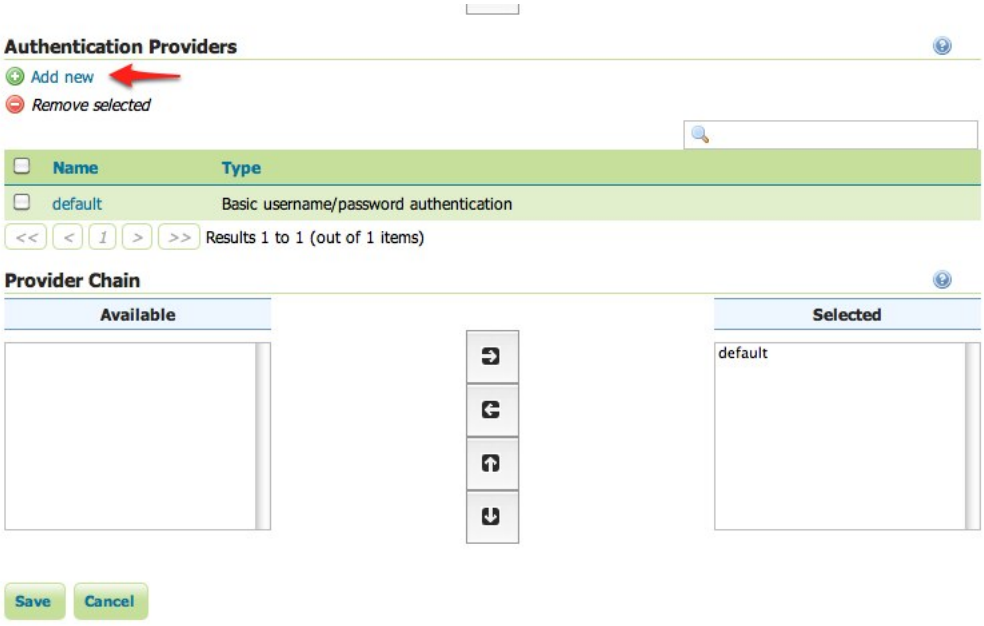
- a group named `GISADMINGROUP` exists.
- a user named `GISADMIN` exists, has password `secret`, and belongs to the `GISADMINGROUP` group.
- a user named `GISUSER` exists, has password `secret`, and does NOT belong to the `GISADMINGROUP` group.

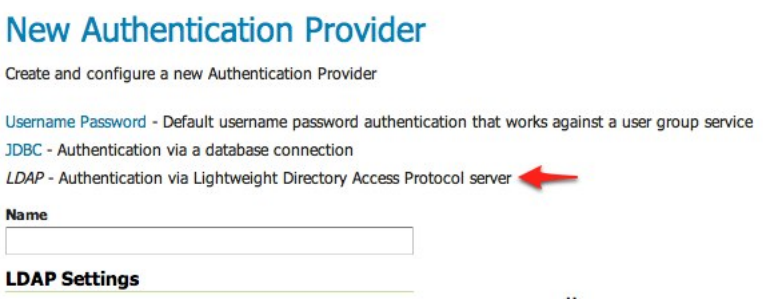
Note: ADMINISTRATOR cannot be generally used as the admin group name with ActiveDirectory, because Administrator is the master user name in Windows environment.

Configure the LDAP authentication provider

1. Start GeoServer and login to the web admin interface as the `admin` user.
2. Click the `Authentication` link located under the `Security` section of the navigation sidebar.
3. Scroll down to the `Authentication Providers` panel and click the `Add new` link.
4. Click the `LDAP` link.
5. Fill in the fields of the settings form as follows:
   - Set `Name` to "ad-ldap"
   - Set `Server URL` to "ldap://domain-controller/dc=ad,dc=local"
   - Set `Filter used to lookup user` to "(|(userPrincipalName={0})(sAMAccountName={1}))"

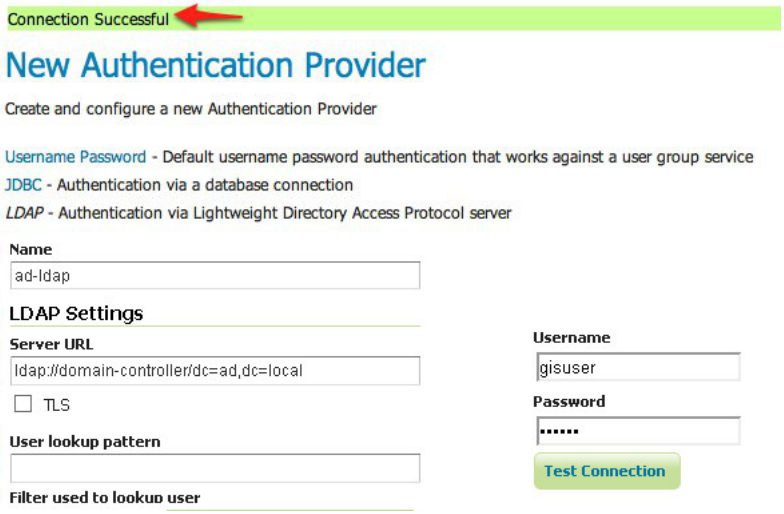

Welcome
Welcome
This GeoServer belongs to The ancien
19 Layers
9 Stores
7 Workspaces
This GeoServer instance is running ver
please contact the administrator.
About & Status
Server Status
GeoServer Logs
Contact Information
About GeoServer
Data
Layer Preview
Workspaces
Stores
Layers
Layer Groups
Cached Layers
Styles
Services
WCS
WFS
WMS
Settings
Global
GeoWebCache
JAI
Coverage Access
Security
Settings
Authentication
Passwords
Users, Groups, Roles
Data
Services


Authentication Providers
Add new
Remove selected
Name
Type
default
Basic username/password authentication
<< < 1 > >> Results 1 to 1 (out of 1 items)
Provider Chain
Available
Selected
default
Save
Cancel



- Set `Format used for user login name` to "{0}@ad.local"
- Check `Use LDAP groups for authorization`
- Check `Bind user before searching for groups`
- Set `Group to use as ADMIN` to "GISADMINGROUP"
- Set `Group search base` to "cn=Users"
- Set `Group search filter` to "member={0}"

6. Test the LDAP connection by entering the username "GISADMIN" and password "secret" in the connection test form located on the right and click the `Test Connection` button.

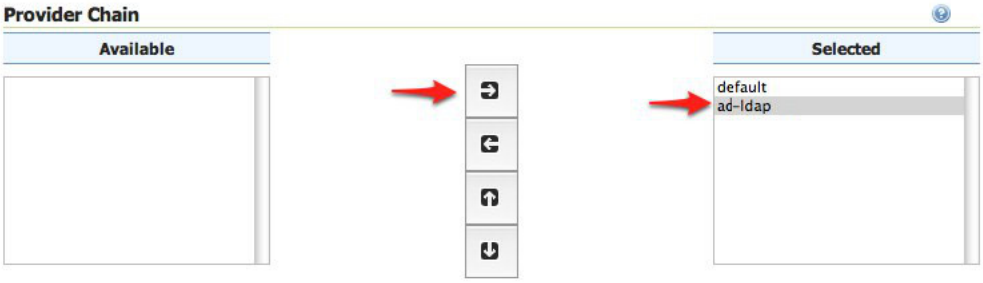


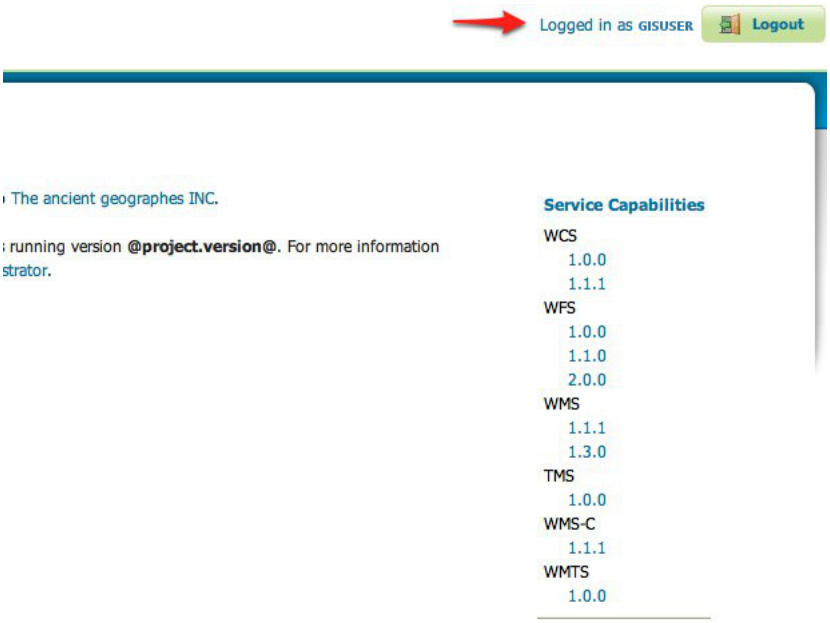
A successful connection should be reported at the top of the page.

7. Save.
8. Back on the authentication page scroll down to the `Provider Chain` panel and move the `ad-ldap` provider from `Available` to `Selected`.
9. Save.

Test a LDAP login

1. Navigate to the GeoServer home page and log out of the admin account.
2. Login as the user "GISUSER" with the with the password "secret".


Provider Chain
Available
Selected
default
ad-ldap


Logged in as GISUSER
Logout
The ancient geographes INC.
running version @project.version@. For more information
strator.
Service Capabilities
WCS
1.0.0
1.1.1
WFS
1.0.0
1.1.0
2.0.0
WMS
1.1.1
1.3.0
TMS
1.0.0
WMS-C
1.1.1
WMTS
1.0.0

Logging in as GISUSER doesn't yield any administrative functionality because the GISUSER account has not been mapped to the administrator role. In the next section GeoServer will be configured to map groups from the LDAP database to roles.

Now we will login with a user having administrative rights.

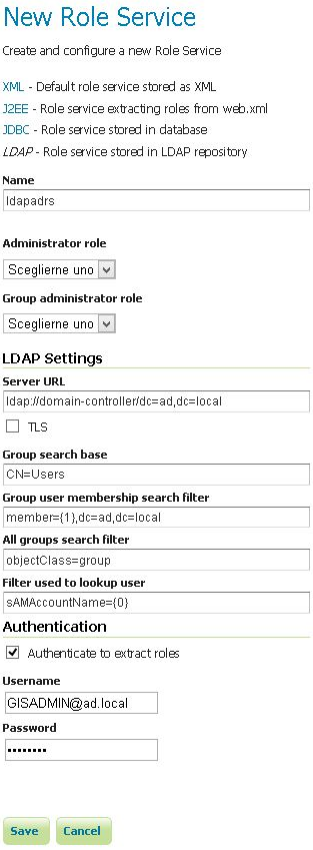
1. Navigate to the GeoServer home page and log out of the account.
2. Login as the user "GISADMIN" with the with the password "secret".

Once logged in full administrative functionality should be available.

Configure the LDAP role service

An additional step permits to configure a role service to get GeoServer roles from the LDAP repository and allow access rights to be assigned to those roles.

1. Click the `Users,Group,Roles` link located under the `Security` section of the navigation sidebar.
2. Click the `Add new link` under the `Role Services` section.
3. Click the `LDAP` option under the `New Role Service` section.



4. Enter `ldapadrs` in the `Name` text field.
5. Enter `ldap://domain-controller/dc=ad,dc=local` in the `Server URL` text field.
6. Enter `CN=Users` in the `Group search base` text field.

7. Enter `member={1},dc=ad,dc=local` in the `Group user membership search filter` text field.
8. Enter `objectClass=group` in the `All groups search filter` text field.
9. Enter `sAMAccountName={0}` in the `Filter used to lookup user` text field.

Then we need to a choose a user to authenticate on the server (many LDAP server don't allow anonymous data lookup).

1. Check the `Authenticate to extract roles` checkbox.
2. Enter `GISADMIN@ad.local` in the `Username` text field.
3. Enter `secret` in the `Password` text field.
4. Save.
5. Click the `ldapadrs` role service item under the `Role Services` section.
6. Select `ROLE_DOMAIN ADMINS` from the `Administrator role` combobox.
7. Select `ROLE_DOMAIN ADMINS` from the `Group administrator role` combobox.
8. Save again.

You should now be able to see and assign the new ActiveDirectory roles wherever an `Available Roles` list is shown (for example in the `Data` and `Services` rules sections.

16.9.3 Configuring Digest Authentication

Introduction

Out of the box GeoServer REST and OGC services support authentication via HTTP Basic authentication. One of the major downsides of basic auth is that it sends user passwords in plain text. HTTP Digest authentication offers a more secure alternative that applies a cryptographic hash function to passwords before sending them over the network.

This tutorial walks through the process of setting up digest authentication.

Prerequisites

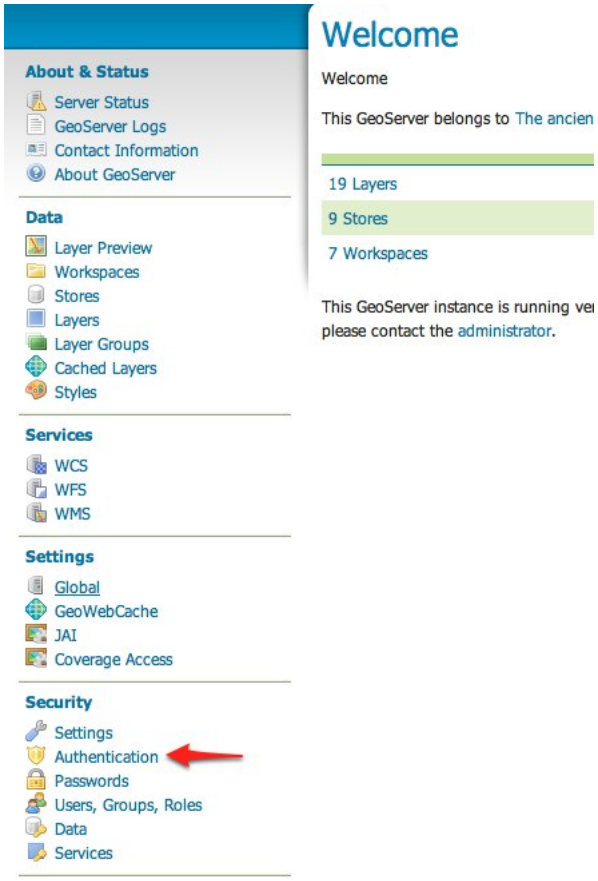
This tutorial uses the curl utility to issue HTTP request that test authentication. Install curl before proceeding.

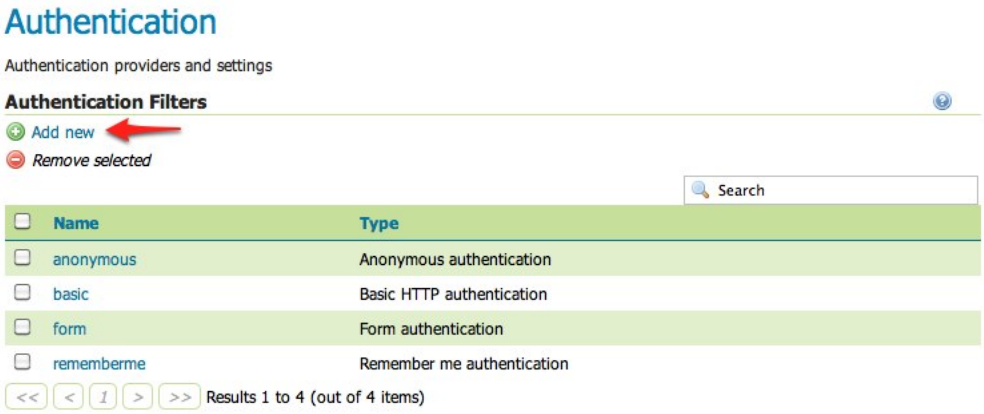
Note: Any utility that supports both basic and digest authentication can be used in place of curl. Most modern web browsers support both types of authentication.

Configure the Digest authentication filter

1. Start GeoServer and login to the web admin interface as the `admin` user.
2. Click the `Authentication` link located under the `Security` section of the navigation sidebar.

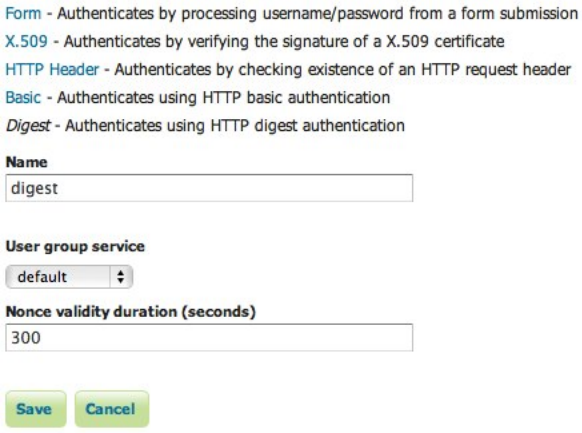
3. Scroll down to the `Authentication Filters` panel and click the `Add new` link.


About & Status
Server Status
GeoServer Logs
Contact Information
About GeoServer
Data
Layer Preview
Workspaces
Stores
Layers
Layer Groups
Cached Layers
Styles
Services
WCS
WFS
WMS
Settings
Global
GeoWebCache
JAI
Coverage Access
Security
Settings
Authentication
Passwords
Users, Groups, Roles
Data
Services
Welcome
Welcome
This GeoServer belongs to The ancien
19 Layers
9 Stores
7 Workspaces
This GeoServer instance is running ver
please contact the administrator.

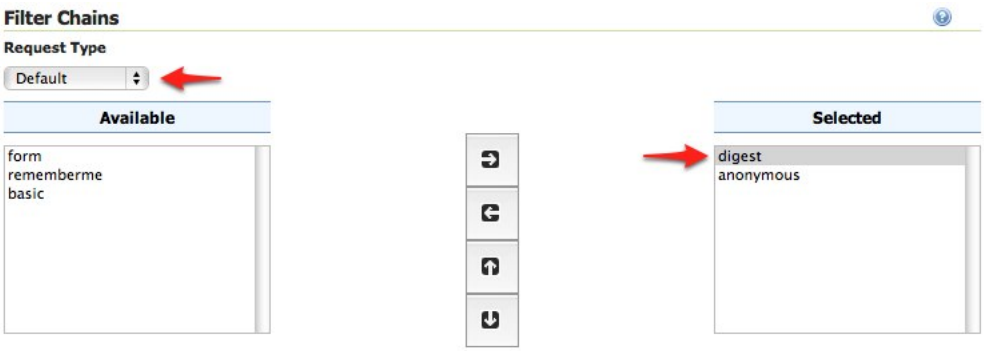

Authentication
Authentication providers and settings
Authentication Filters
Add new
Remove selected
Search
Name
Type
anonymous
Anonymous authentication
basic
Basic HTTP authentication
form
Form authentication
rememberme
Remember me authentication
<< < 1 > >> Results 1 to 4 (out of 4 items)

4. Click the `Digest` link.

5. Fill in the fields of the settings form as follows:
   - Set `Name` to "digest"
   - Set `User group service` to "default"



6. Save.
7. Back on the authentication page scroll down to the `Filter Chains` panel.
8. Select "Default" from the `Request type` drop down.
9. Unselect the `basic` filter and select the `digest` filter. Position the the `digest` filter before the `anonymous` filter.



10. Save.

Secure OGC service requests

In order to test the authentication settings configured in the previous section a service or resource must be first secured. The `Default` filter chain is the chain applied to all OGC service requests so a service security rule must be configured.

1. From the GeoServer home page and click the `Services` link located under the `Security` section of the navigation sidebar.

2. On the Service security page click the `Add new rule` link and add a catch all rule that secures all OGC service requests requiring the `ROLE_ADMINISTRATOR` role.

3. Save.

Test a digest authentication login

1. Ensure that basic authentication is disabled execute the following curl command:

```
curl -v -u admin:geoserver -G "http://localhost:8080/geoserve/wfs?request=getcapabilities"
```

The result should be a 401 response signaling that authentication is required. The output should look something like the following:

```
* About to connect() to localhost port 8080 (#0)
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 8080 (#0)
* Server auth using Basic with user 'admin'
> GET /geoserver/wfs?request=getcapabilities HTTP/1.1
> Authorization: Basic YWRtaW46Z2Vvc2VydmVy
> User-Agent: curl/7.19.7 (universal-apple-darwin10.0) libcurl/7.19.7 OpenSSL/0.9.8r zlib/1.2.3
> Host: localhost:8080
> Accept: */*
>
< HTTP/1.1 401 Full authentication is required to access this resource
< Set-Cookie: JSESSIONID=1dn2bi8qqu5qc;Path=/geoserver
< WWW-Authenticate: Digest realm="GeoServer Realm", qop="auth", nonce="MTMzMzQzMDkxMTU3MjphZGIwM
< Content-Type: text/html; charset=iso-8859-1
< Content-Length: 1491
< Server: Jetty(6.1.8)
<
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
<title>Error 401 Full authentication is required to access this resource</title>
</head>
...
```

2. Execute the same command but specify the `--digest` option to tell curl to use digest authentication rather than basic authentication:

   ```
   curl --digest -v -u admin:geoserver -G "http://localhost:8080/geoserve/wfs?request=getcapabiliti
   ```

   The result should be a successful authentication and contain the normal WFS capabilities response.

16.9.4 Configuring X.509 Certificate Authentication

Certificates authentication provides a much more secure alternative basic username password schemes that involves the usage of public/private keys to identify ones self. X.509 is a well defined standard for the format of public key certificates.

This tutorial walks through the process of setting up X.509 certificate authentication.

Prerequisites

This tutorial requires the following:

- A web browser that supports the usage of client certificates for authentication. This is also referred to as "two way SSL". This tutorial uses Firefox.
- An SSL capable servlet container. This tutorial uses Tomcat.

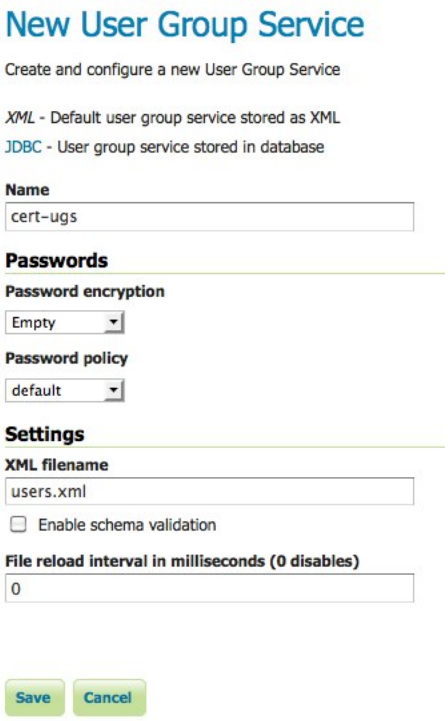
Deploy GeoServer in tomcat before proceeding.

Configure the user group service

Users authenticated via X.509 certificate must be configured in GeoServer. For this a new user group service will be added.

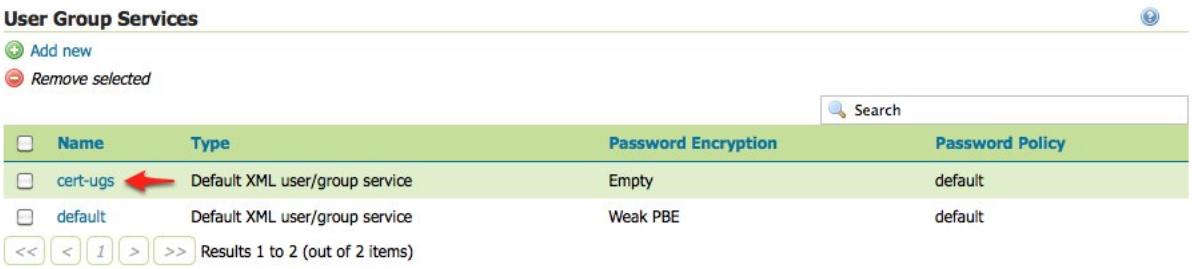
1. Login to the web admin interface as the `admin` user.
2. Click the `Users, Groups, and Roles` link located under the `Security` section of the navigation sidebar.

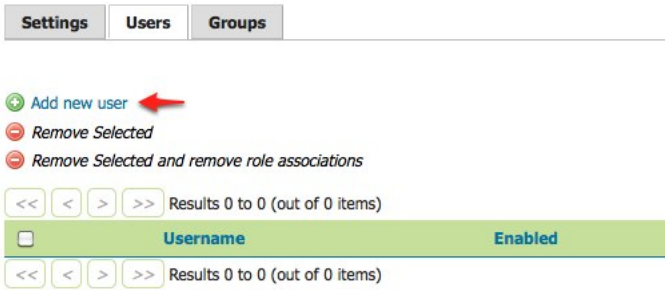
3. Scroll down to the `User Group Services` panel and click the `Add new` link.
4. Create a new user group service named "cert-ugs" and fill out the settings form as follows:
   - Set `Password encryption` to "Empty" since users will not authenticate via password.
   - Set `Password policy` to "default".

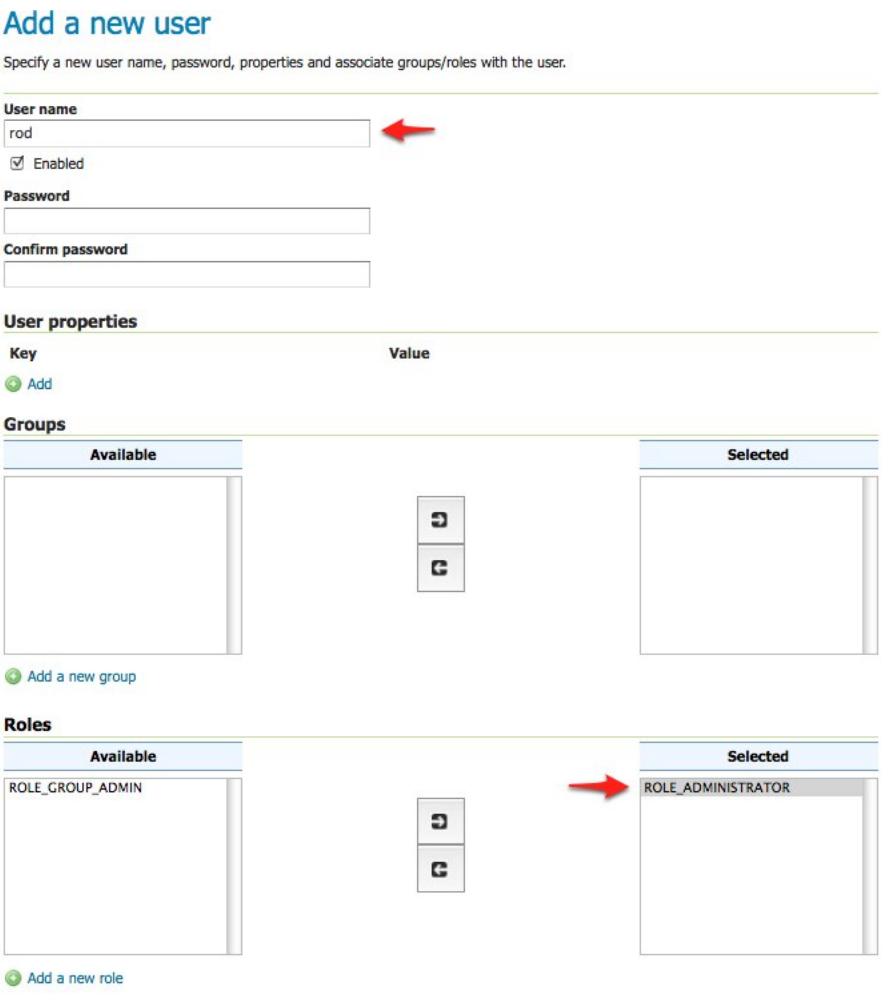


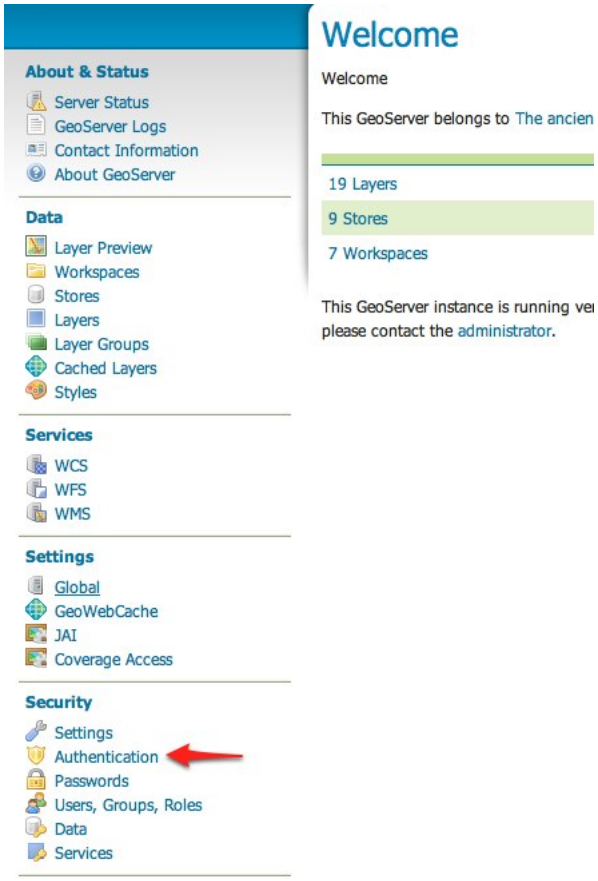
5. Save
6. Back on the `Users, Groups, and Roles` page click the `cert-ugs` link.
7. Select the `Users` tab and click the `Add new user` link.
8. Add a new user named "rod" the and assign the `ROLE_ADMINISTRATOR` role.
9. Save.
10. Click the `Authentication` link located under the `Security` section of the navigation sidebar.

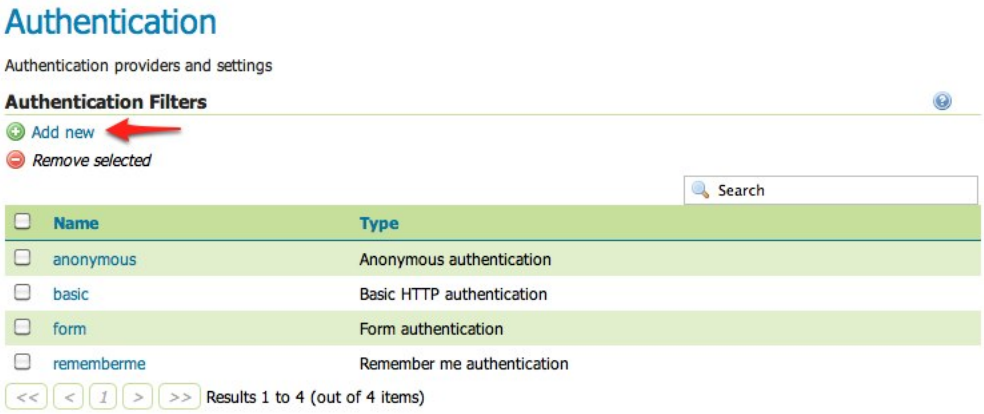
11. Scroll down to the `Authentication Filters` panel and click the `Add new` link.


User Group Services
Add new
Remove selected
Search
Name
Type
Password Encryption
Password Policy
cert-ugs
Default XML user/group service
Empty
default
default
Default XML user/group service
Weak PBE
default
Results 1 to 2 (out of 2 items)

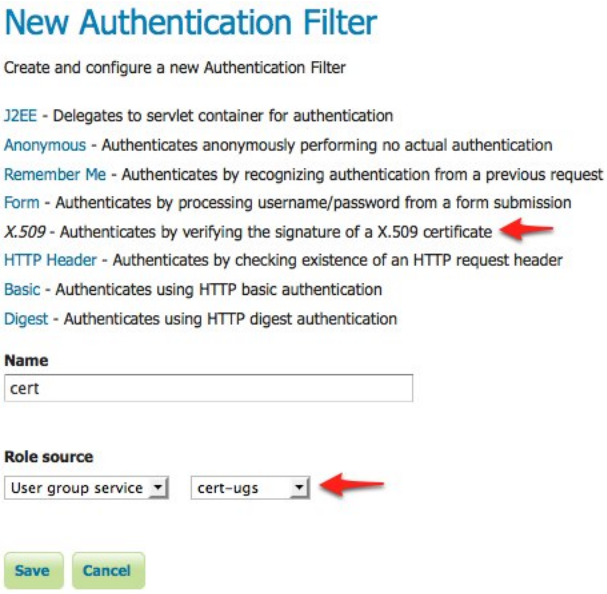

Settings
Users
Groups
Add new user
Remove Selected
Remove Selected and remove role associations
Results 0 to 0 (out of 0 items)
Username
Enabled
Results 0 to 0 (out of 0 items)


Add a new user
Specify a new user name, password, properties and associate groups/roles with the user.
User name
rod
Enabled
Password
Confirm password
User properties
Key
Value
Add
Groups
Available
Selected
Add a new group
Roles
Available
Selected
ROLE_GROUP_ADMIN
ROLE_ADMINISTRATOR
Add a new role

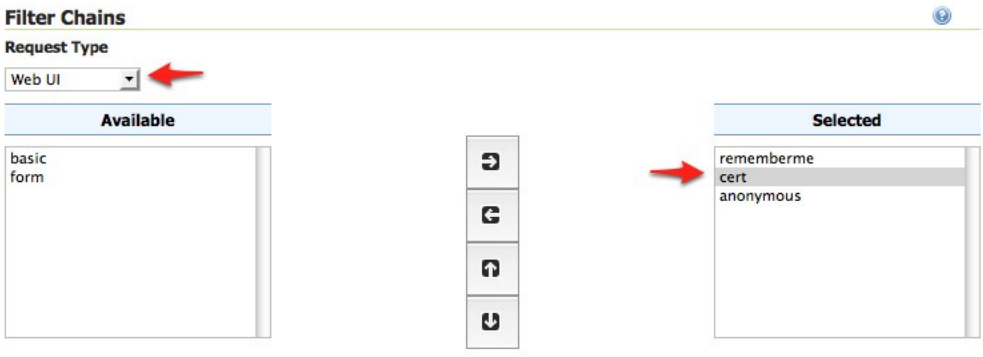

About & Status
Server Status
GeoServer Logs
Contact Information
About GeoServer
Data
Layer Preview
Workspaces
Stores
Layers
Layer Groups
Cached Layers
Styles
Services
WCS
WFS
WMS
Settings
Global
GeoWebCache
JAI
Coverage Access
Security
Settings
Authentication
Passwords
Users, Groups, Roles
Data
Services
Welcome
Welcome
This GeoServer belongs to The ancien
19 Layers
9 Stores
7 Workspaces
This GeoServer instance is running ver
please contact the administrator.


Authentication
Authentication providers and settings
Authentication Filters
Add new
Remove selected
Search
Name
Type
anonymous
Anonymous authentication
basic
Basic HTTP authentication
form
Form authentication
rememberme
Remember me authentication
<< < 1 > >> Results 1 to 4 (out of 4 items)

12. Click the `X.509` link and fill out settings form as follows:
    - Set `Name` to "cert"
    - Set `Role source` to "User group service" and set the associated drop down to "cert-ugs"



13. Save.
14. Back on the authentication page scroll down to the `Filter Chains` panel.
15. Select "Web UI" from the `Request type` drop down.
16. Select the `digest` filter and position it after the `anonymous` filter.



17. Save.

Download sample certificate files

Rather than demonstrate how to create or obtain valid certificates, which is beyond the scope of this tutorial, sample files available as part of the spring security sample applications will be used.

Download and unpack the `sample certificate files`. The zip archive contains the following files:

- `ca.pem` is the certificate authority (CA) certificate issued by the "Spring Security Test CA" certificate authority. This file is used to sign the server and client certificates.
- `server.jks` is the Java keystore containing the server certificate and private key used by Tomcat and presented to the user during the setup of the SSL connection.
- `rod.p12` contains the client certificate / key combination used to perform client authentication via the web browser.

Configure Tomcat for SSL

1. Copy the `server.jks` file into the `conf` directory under the root of the Tomcat installation.
2. Edit the Tomcat `conf/server.xml` and add an SSL connector:

   ```
   <Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true" scheme="https" secure="true"
        clientAuth="true" sslProtocol="TLS"
        keystoreFile="${catalina.home}/conf/server.jks"
        keystoreType="JKS" keystorePass="password"
        truststoreFile="${catalina.home}/conf/server.jks"
        truststoreType="JKS" truststorePass="password" />
   ```

   This enables SSL on port 8443.
3. Restart Tomcat.

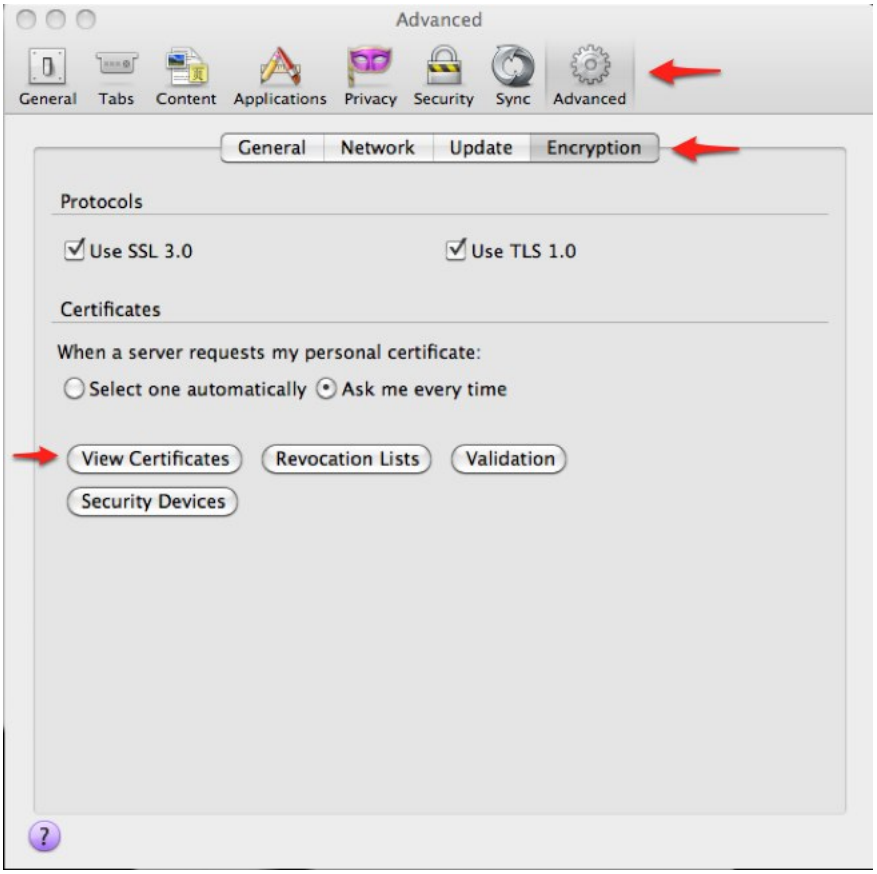
Install the client certificate

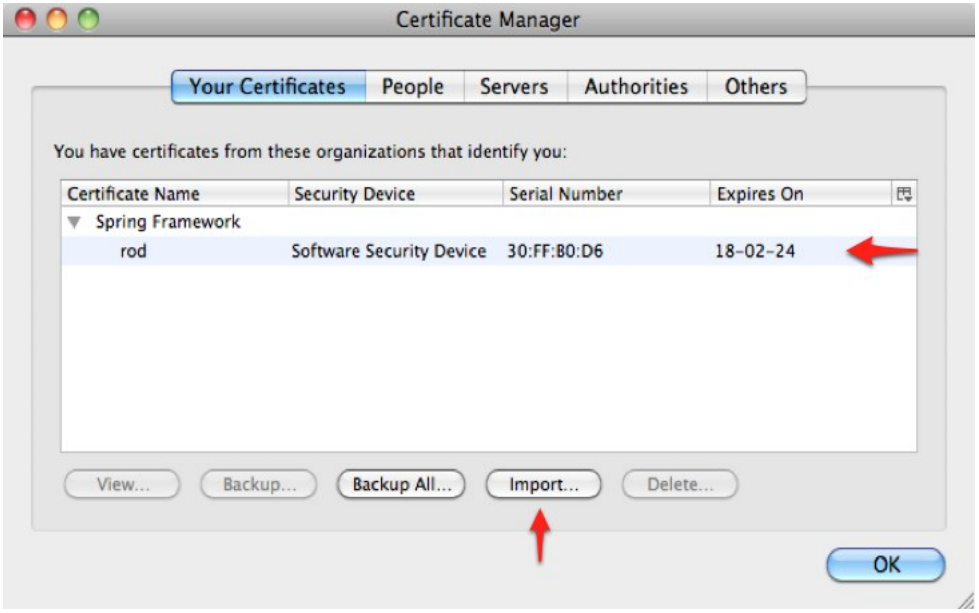
1. In Firefox select `Preferences` and navigate to the `Advanced` panel.
2. Select the `Encryption` tab and click the `View Certificates` button.
3. On the `Your Certificates` panel click the `Import...` button and in the file browser select the `rod.p12` file.
4. When prompted enter in the password "password".
5. Click `Ok` and close the Firefox preferences.

Test certificate login

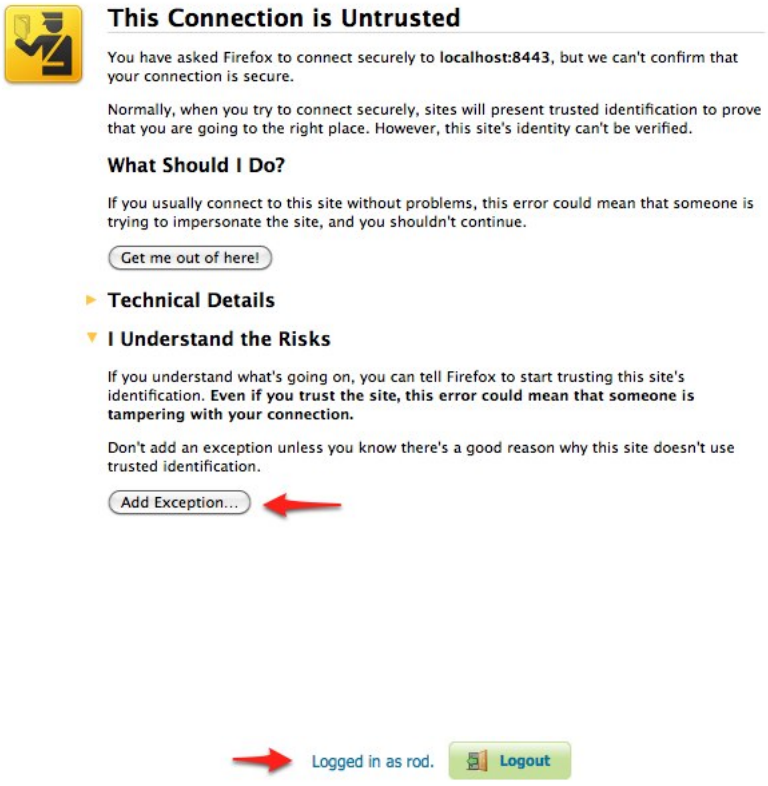
1. In Firefox navigate to the GeoServer admin on port "8443" using "https". https://localhost:8443/geoserver/web
2. When prompted select the "rod" certificate for identification.
3. When warned about the self signed server certificate add a security exception to proceed.

The result is the `rod` user logged into the GeoServer admin interface.


Advanced
General
Tabs
Content
Applications
Privacy
Security
Sync
Advanced
General
Network
Update
Encryption
Protocols
Use SSL 3.0
Use TLS 1.0
Certificates
When a server requests my personal certificate:
Select one automatically
Ask me every time
View Certificates
Revocation Lists
Validation
Security Devices


Certificate Manager
Your Certificates
People
Servers
Authorities
Others
You have certificates from these organizations that identify you:
Certificate Name
Security Device
Serial Number
Expires On
Spring Framework
rod
Software Security Device
30:FF:B0:D6
18-02-24
View...
Backup...
Backup All...
Import...
Delete...
OK


This site has requested that you identify yourself with a certificate:
localhost:8443
Organization: "Spring Security"
Issued Under: "Spring Framework"
Choose a certificate to present as identification:
rod [30:FF:B0:D6]
Details of selected certificate:
Issued to: CN=rod,OU=Spring Security,O=Spring Framework
Serial Number: 30:FF:B0:D6
Valid from 08-01-25 5:50:22 to 18-02-24 17:00:00
Certificate Key Usage: Signing
Issued by: CN=Spring Security Test CA,OU=Spring Security,O=Spring Framework,L=Glasgow,ST=Scotland,C=GB
Stored in: Software Security Device
Remember this decision
Cancel
OK


This Connection is Untrusted
You have asked Firefox to connect securely to localhost:8443, but we can't confirm that your connection is secure.
Normally, when you try to connect securely, sites will present trusted identification to prove that you are going to the right place. However, this site's identity can't be verified.
What Should I Do?
If you usually connect to this site without problems, this error could mean that someone is trying to impersonate the site, and you shouldn't continue.
Get me out of here!
Technical Details
I Understand the Risks
If you understand what's going on, you can tell Firefox to start trusting this site's identification. Even if you trust the site, this error could mean that someone is tampering with your connection.
Don't add an exception unless you know there's a good reason why this site doesn't use trusted identification.
Add Exception...

Logged in as rod.
Logout

16.9.5 Configuring J2EE Authentication

Servlet containers such as Tomcat and Jetty offer their own options for authentication. Often it is desirable for an application such as GeoServer to use that existing authentication mechanisms rather than require its own authentication configuration.

J2EE authentication allows GeoServer to delegate to the servlet container for authentication. This tutorial walks through the process of setting up J2EE authentication.

Prerequisites

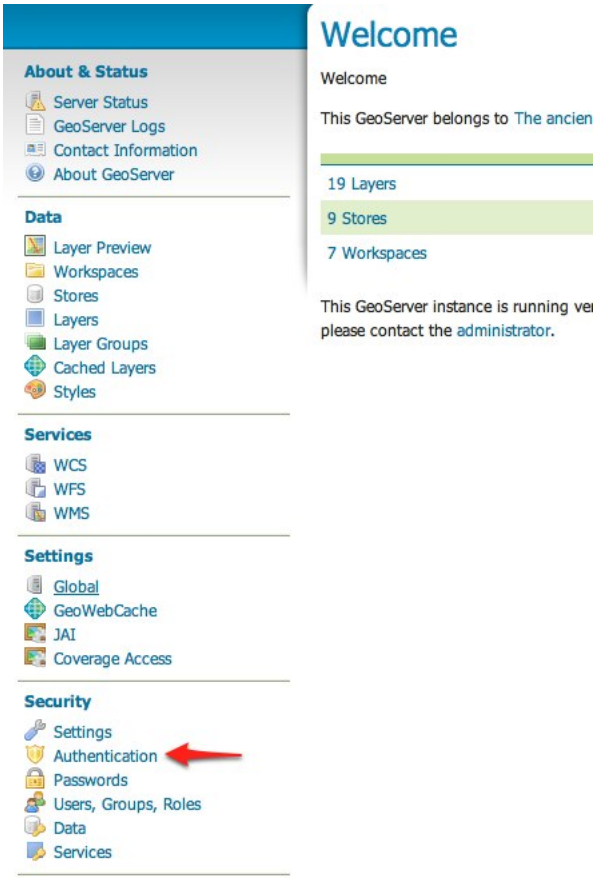
This tutorial requires a servlet container capable of doing its own authentication. This tutorial uses Tomcat.

Deploy GeoServer in tomcat before proceeding.

Configure the J2EE authentication filter

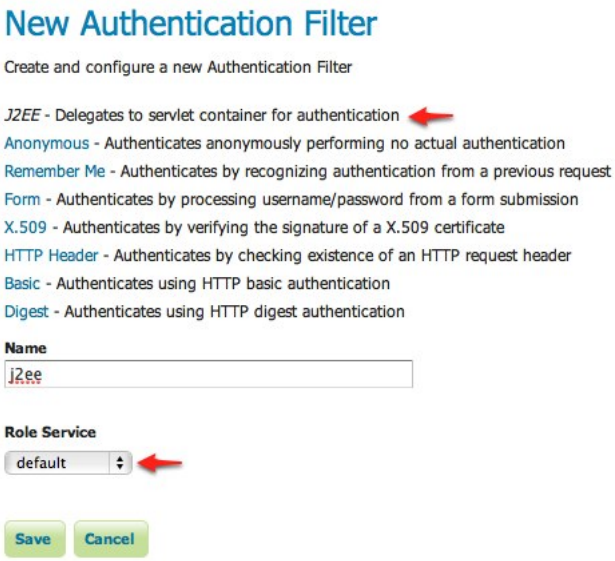
In order to delegate to the container for authentication a filter must first be configured to recognize the container authentication.

1. Login to the GeoServer web admin interface as the `admin` user.
2. Click the `Authentication` link located under the `Security` section of the navigation sidebar.

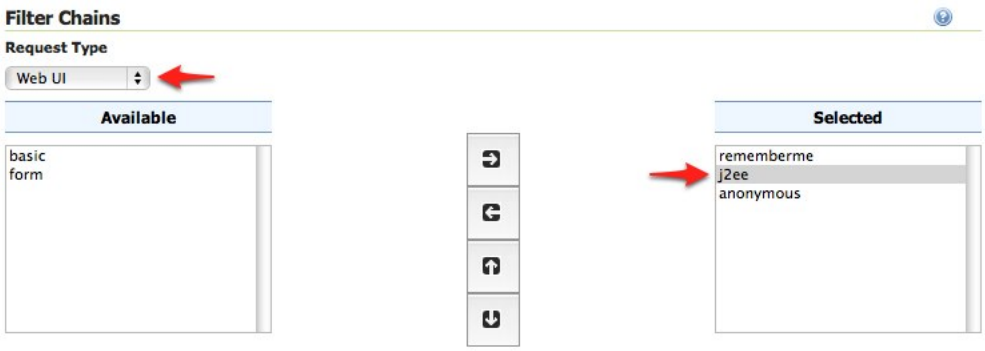


3. Scroll down to the `Authentication Filter` panel and click the `Add new` link.

4. Create a new filter named "j2ee" and fill out the settings form as follows:
   - Set the `Role service` to "default"



5. Save
6. Back on the authentication page scroll down to the `Filter Chains` panel.
7. Select "Web UI" from the `Request type` drop down.
8. Select the `j2ee` filter and position it after the `anonymous` filter.



9. Save.

Configure the role service

Since it is not possible to ask a J2EE container for the roles of a principal it is necessary to have all J2EE roles enlisted in a role service. The only J2EE API GeoServer can use is:

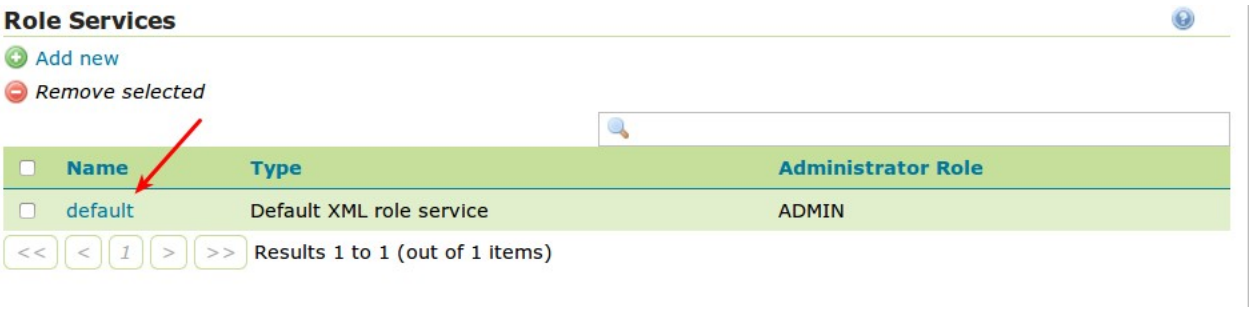
```
class: javax.servlet.http.HttpServletRequest
method: boolean isUserInRole(String role)
```

The idea is to query all roles from the role service and test each role with the "isUserInRole" method.

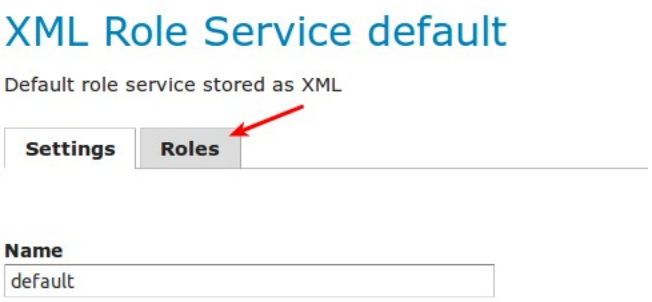
This tutorial assumes a user named "admin" with password "password" and a J2EE role named "tomcat".

1. Click the `Users, Groups, and Roles` link located under the `Security` section of the navigation sidebar.

2. Click on `default` to work with the role service named "default".



3. Click on the `Roles` tab.



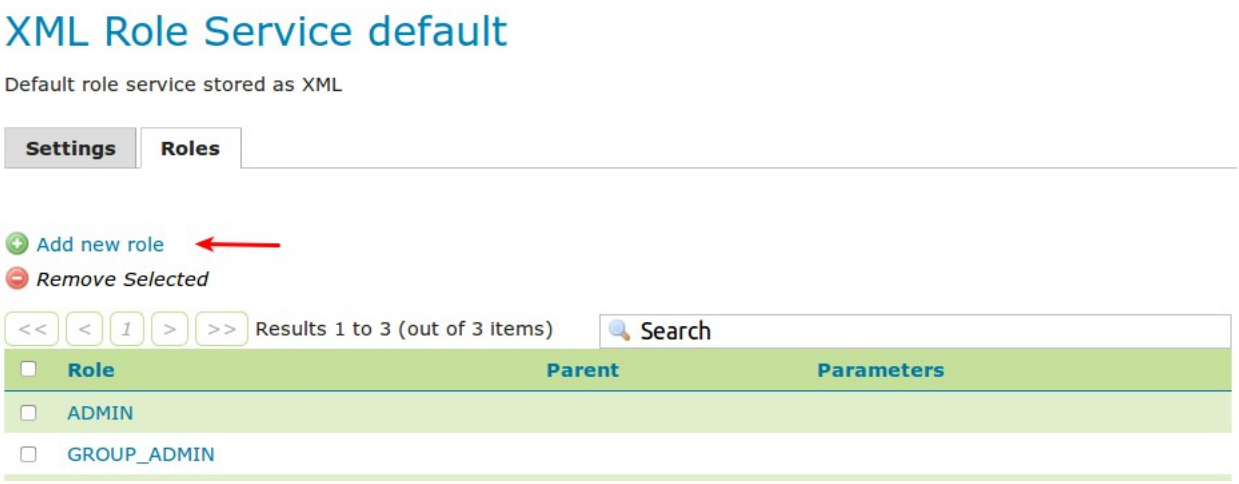
4. Click on the `Add new role` link.
   - Set the `Name` to "tomcat"
5. Save

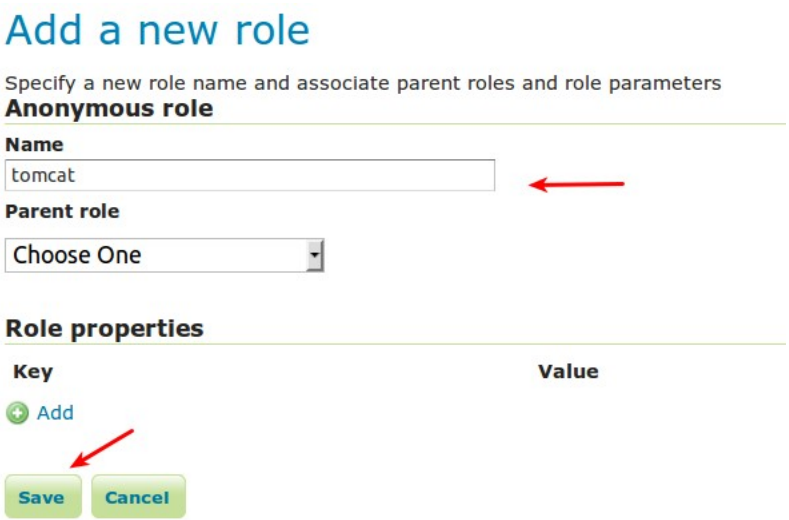
Configure Tomcat for authentication

By default Tomcat does not require authentication for web applications. In this section Tomcat will be configured to secure GeoServer requiring a basic authentication login.

1. Shut down Tomcat.
2. Edit the `conf/tomcat-users.xml` under the Tomcat root directory and add a user named "admin":

```
<user username="admin" password="password" roles="tomcat"/>
```


XML Role Service default
Default role service stored as XML
Settings
Roles
Add new role
Remove Selected
<< < 1 > >> Results 1 to 3 (out of 3 items)
Search
Role
Parent
Parameters
ADMIN
GROUP_ADMIN


Add a new role
Specify a new role name and associate parent roles and role parameters
Anonymous role
Name
tomcat
Parent role
Choose One
Role properties
Key
Value
Add
Save
Cancel

3. Edit the GeoServer `web.xml` file located at `webapps/geoserver/WEB-INF/web.xml` under the Tomcat root directory and add the following at the end of the file directly before the closing `</web-app>` element:

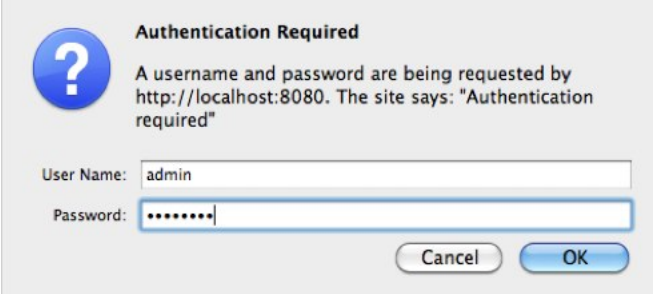
```
<security-constraint>
  <web-resource-collection>
    <url-pattern>/*</url-pattern>
     <http-method>GET</http-method>
     <http-method>POST</http-method>
  </web-resource-collection>
  <auth-constraint>
    <role-name>tomcat</role-name>
  </auth-constraint>
</security-constraint>

<login-config>
  <auth-method>BASIC</auth-method>
</login-config>
```

4. Save `web.xml` and restart Tomcat.

Note: It is necessary to add all the role names specified in the `web.xml` to the configured role service. This is duplicate work but there is currently no other solution.

Test J2EE login

1. Navigate to the GeoServer web admin interface. The result should be a prompt to authenticate.
2. Enter in the username "admin" and password "password"



The result should be the admin user logged into the GeoServer web admin.

16.9.6 Configuring HTTP Header Proxy Authentication

Introduction

Proxy authentication is used in multi-tier system. The user/principal authenticates at the proxy and the proxy provides the authentication information to other services.

This tutorial shows how to configure GeoServer to accept authentication information passed by HTTP header attribute(s). In this scenario GeoServer will do no actual authentication itself.

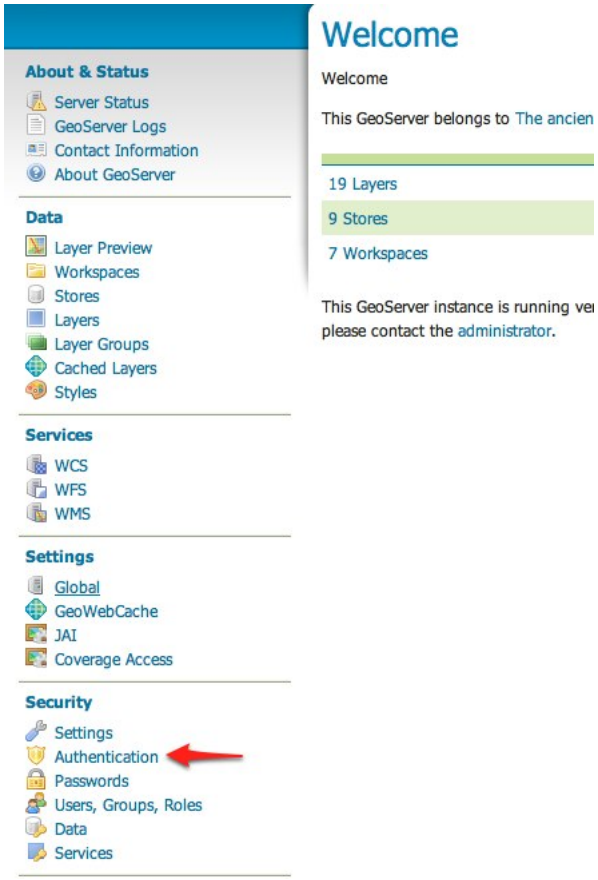
Prerequisites

This tutorial uses the curl utility to issue HTTP request that test authentication. Install curl before proceeding.

Note: Any utility that supports setting HTTP header attributes can be used in place of curl.

Configure the HTTP header filter

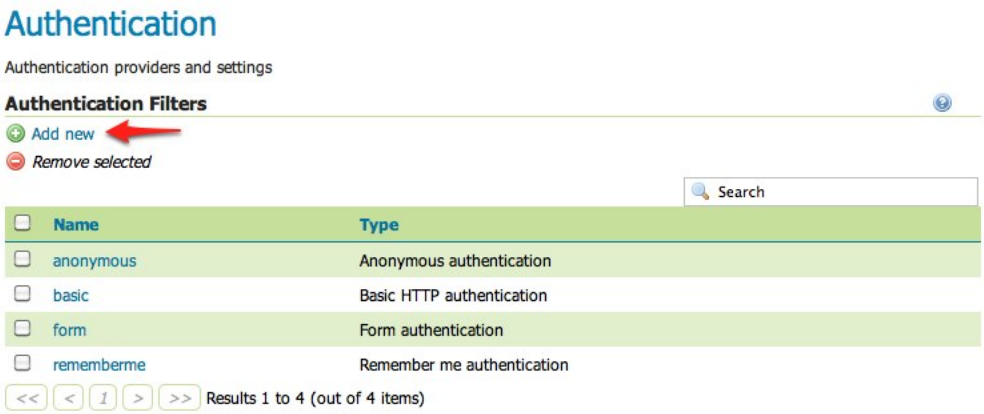
1. Start GeoServer and login to the web admin interface as the `admin` user.
2. Click the `Authentication` link located under the `Security` section of the navigation sidebar.

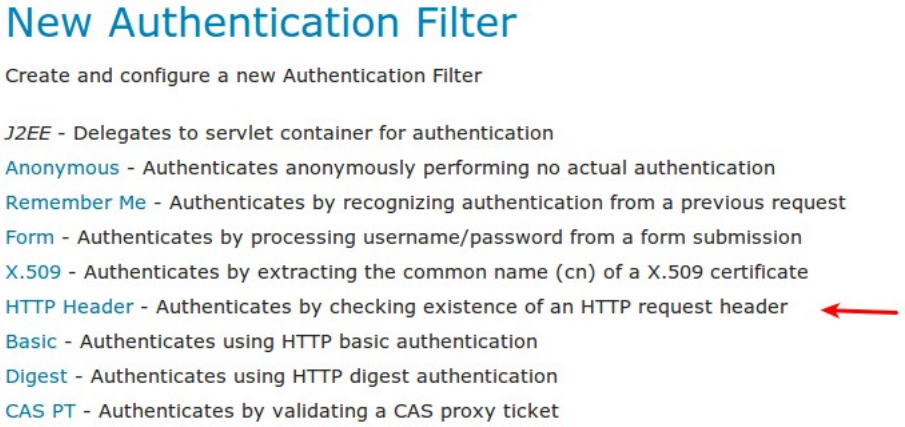


3. Scroll down to the `Authentication Filters` panel and click the `Add new` link.

4. Click the `HTTP Header` link.

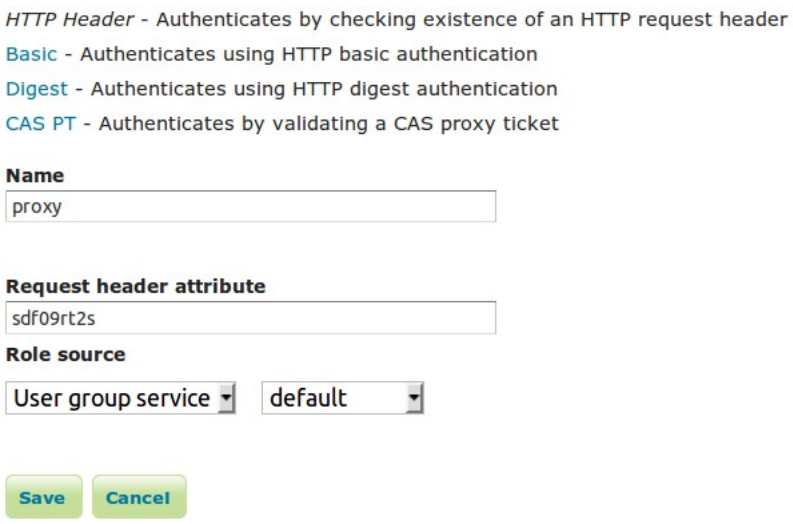
5. Fill in the fields of the settings form as follows:
   - Set `Name` to "proxy"
   - Set `Request header attribute to` to "sdf09rt2s"


Authentication
Authentication providers and settings
Authentication Filters
Add new
Remove selected
Search
Name
Type
anonymous
Anonymous authentication
basic
Basic HTTP authentication
form
Form authentication
rememberme
Remember me authentication
<< < 1 > >> Results 1 to 4 (out of 4 items)


New Authentication Filter
Create and configure a new Authentication Filter
J2EE - Delegates to servlet container for authentication
Anonymous - Authenticates anonymously performing no actual authentication
Remember Me - Authenticates by recognizing authentication from a previous request
Form - Authenticates by processing username/password from a form submission
X.509 - Authenticates by extracting the common name (cn) of a X.509 certificate
HTTP Header - Authenticates by checking existence of an HTTP request header
Basic - Authenticates using HTTP basic authentication
Digest - Authenticates using HTTP digest authentication
CAS PT - Authenticates by validating a CAS proxy ticket

- Set `Role source` to "User group service"
- Set the name of the user group service to "default"

Additional information about role services is here *Role source and role calculation*



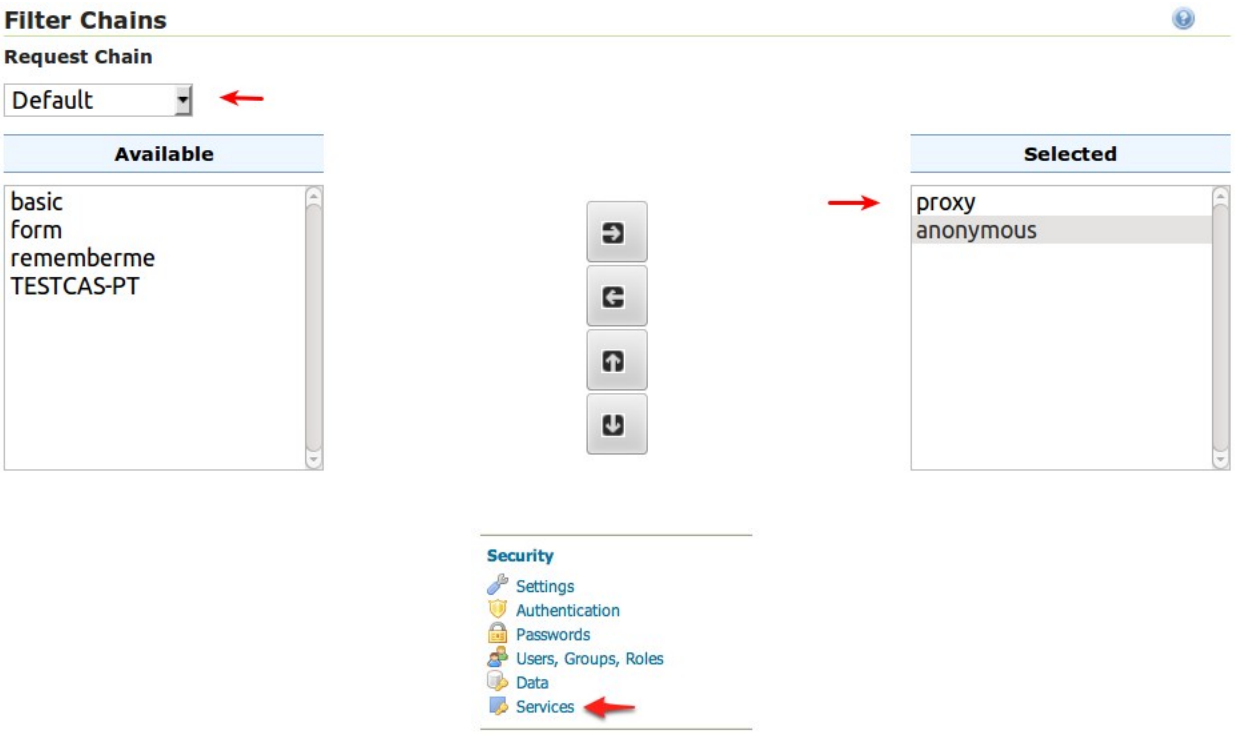
> **Warning:** The tutorial uses the obscure "sdf09rt2s" name for the header attribute. Why not use "user" or "username" ?. In a proxy scenario a relationship of trust is needed between the proxy and GeoServer. An attacker could easily send an HTTP request with an HTTP header attribute "user" and value "admin" and operate as an administrator.
> One possibility is to configure the network infrastructure preventing such requests from all IP addresses except the IP of the proxy.
> This tutorial uses a obscure header attribute name which should be a shared secret between the proxy and GeoServer. Additionally, the use of SSL is recommended, otherwise the shared secret is transported in plain text.

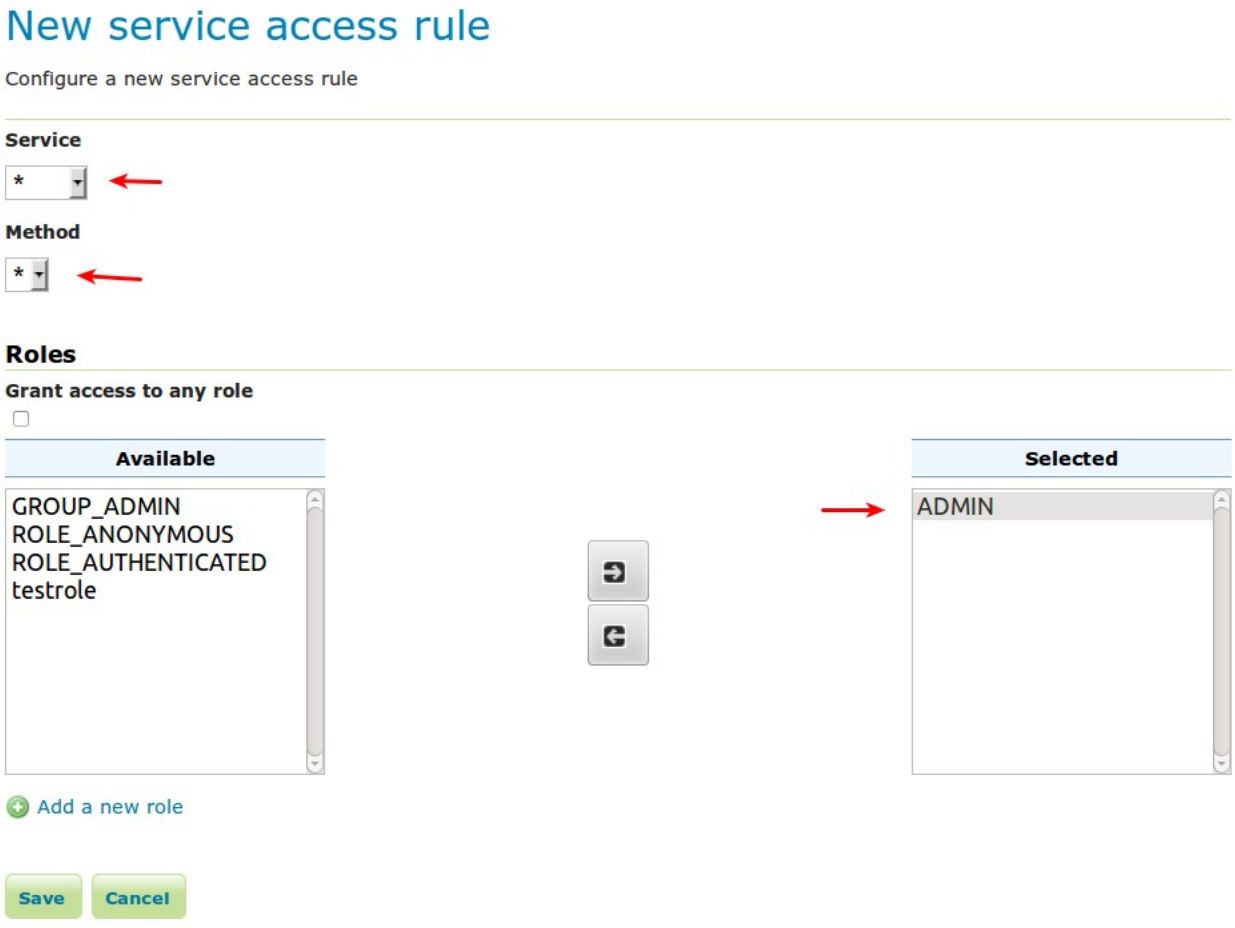
1. Save.
2. Back on the authentication page scroll down to the `Filter Chains` panel.
3. Select "Default" from the `Request type` drop down.
4. Unselect the `basic` filter and select the `proxy` filter. Position the the `proxy` filter before the `anonymous` filter.
5. Save.

Secure OGC service requests

In order to test the authentication settings configured in the previous section a service or resource must be first secured. The `Default` filter chain is the chain applied to all OGC service requests so a service security rule must be configured.

1. From the GeoServer home page and click the `Services` link located under the `Security` section of the navigation sidebar.
2. On the Service security page click the `Add new rule` link and add a catch all rule that secures all OGC service requests requiring the `ADMIN` role.
3. Save.


Filter Chains
Request Chain
Default
Available
basic
form
rememberme
TESTCAS-PT
Selected
proxy
anonymous
Security
Settings
Authentication
Passwords
Users, Groups, Roles
Data
Services


New service access rule
Configure a new service access rule
Service
*
Method
*
Roles
Grant access to any role
Available
GROUP_ADMIN
ROLE_ANONYMOUS
ROLE_AUTHENTICATED
testrole
Selected
ADMIN
Add a new role
Save
Cancel

Test a proxy login

1. Execute the following curl command:

   ```
   curl -v  -G "http://localhost:8080/geoserver/wfs?request=getcapabilities"
   ```

   The result should be a 403 response signaling that access is denied. The output should look something like the following:

   ```
   * About to connect() to localhost port 8080 (#0)
   *   Trying ::1... connected
   > GET /geoserver/wfs?request=getcapabilities HTTP/1.1
   > User-Agent: curl/7.22.0 (x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4 libidn
   > Host: localhost:8080
   > Accept: */*
   >
   < HTTP/1.1 403 Access Denied
   < Content-Type: text/html; charset=iso-8859-1
   < Content-Length: 1407
   < Server: Jetty(6.1.8)
   <
   <html>
   <head>
   <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
   <title>Error 403 Access Denied</title>
   </head>
   ...
   ```

2. Execute the same command but specify the `--header` option.:

   ```
   curl -v --header "sdf09rt2s: admin" -G "http://localhost:8080/geoserver/wfs?request=getcapabilit
   ```

   The result should be a successful authentication and contain the normal WFS capabilities response.

CHAPTER 17

Running in a Production Environment

GeoServer is geared towards many different uses, from a simple test server to the enterprise-level data server. While many optimizations for GeoServer are set by default, here are some extra considerations to keep in mind when running GeoServer in a production environment.

17.1 Java Considerations

17.1.1 Use Oracle JRE

Note: As of version 2.0, a Java Runtime Environment (JRE) is sufficient to run GeoServer. GeoServer no longer requires a Java Development Kit (JDK).

GeoServer's speed depends a lot on the chosen Java Runtime Environment (JRE). For best performance, use Oracle JRE 6 (also known as JRE 1.6) or newer. (As of GeoServer 2.2.x, Oracle JRE 5 is no longer supported.) JREs other than those released by Oracle may work correctly, but are generally not tested or supported. Users report GeoServer to be working with OpenJDK, but expect reductions in 2D rendering performance.

17.1.2 Install native JAI and JAI Image I/O extensions

The Java Advanced Imaging API (JAI) is an advanced image manipulation library built by Oracle. GeoServer requires JAI to work with coverages and leverages it for WMS output generation. By default, GeoServer ships with the pure Java version of JAI, but **for best performance, install the native JAI version in your JDK/JRE**.

In particular, installing the native JAI is important for all raster processing, which is used heavily in both WMS and WCS to rescale, cut and reproject rasters. Installing the native JAI is also important for all raster reading and writing, which affects both WMS and WCS. Finally, native JAI is very useful even if there is no raster data involved, as WMS output encoding requires writing PNG/GIF/JPEG images, which are themselves rasters.

Native extensions are available for Windows, Linux and Solaris (32 and 64 bit systems). They are, however, not available for OS X.

Note: These installers are limited to allow adding native extensions to just one version of the JDK/JRE on your system. If native extensions are needed on multiple versions, manually unpacking the extensions will be necessary. See the section on *Installing native JAI manually*.

Note: These installers are also only able to apply the extensions to the currently used JDK/JRE. If native extensions are needed on a different JDK/JRE than that which is currently used, it will be necessary to uninstall the current one first, then run the setup program against the remaining JDK/JRE.

Installing native JAI on Windows

1. Go to the JAI download page and download the Windows installer for version 1.1.3. At the time of writing only the 32 bit version of the installer is available, so if you are using a JDK, you will want to download jai-1_1_3-lib-windows-i586-jdk.exe, and if you are using a JRE, you will want to download jai-1_1_3-lib-windows-i586-jre.exe.
2. Run the installer and point it to the JDK/JRE install that GeoServer will use to run.
3. Go to the JAI Image I/O download page and download the Windows installer for version 1.1. At the time of writing only the 32 bit version of the installer is available, so if you are using a JDK, you will want to download jai_imageio-1_1-lib-windows-i586-jdk.exe, and if you are using a JRE, you will want to download jai_imageio-1_1-lib-windows-i586-jre.exe
4. Run the installer and point it to the JDK/JRE install that GeoServer will use to run.

Installing native JAI on Linux

1. Go to the JAI download page and download the Linux installer for version 1.1.3, choosing the appropriate architecture:
   - *i586* for the 32 bit systems
   - *amd64* for the 64 bit ones (even if using Intel processors)
2. Copy the file into the directory containing the JDK/JRE and then run it. For example, on an Ubuntu 32 bit system:

```
$ sudo cp jai-1_1_3-lib-linux-i586-jdk.bin /usr/lib/jvm/java-6-sun
$ cd /usr/lib/jvm/java-6-sun
$ sudo sh jai-1_1_3-lib-linux-i586-jdk.bin
# accept license
$ sudo rm jai-1_1_3-lib-linux-i586-jdk.bin
```

3. Go to the JAI Image I/O download page and download the Linux installer for version 1.1, choosing the appropriate architecture:
   - *i586* for the 32 bit systems
   - *amd64* for the 64 bit ones (even if using Intel processors)
4. Copy the file into the directory containing the JDK/JRE and then run it. If you encounter difficulties, you may need to export the environment variable `_POSIX2_VERSION=199209`. For example, on a Ubuntu 32 bit Linux system:

```
$ sudo cp jai_imageio-1_1-lib-linux-i586-jdk.bin /usr/lib/jvm/java-6-sun
$ cd /usr/lib/jvm/java-6-sun
$ sudo su
$ export _POSIX2_VERSION=199209
$ sh jai_imageio-1_1-lib-linux-i586-jdk.bin
# accept license
$ rm ./jai_imageio-1_1-lib-linux-i586-jdk.bin
$ exit
```

Installing native JAI manually

You can install the native JAI manually if you encounter problems using the above installers, or if you wish to install the native JAI for more than one JDK/JRE.

Please refer to the GeoTools page on JAI installation for details.

GeoServer cleanup

Once the installation is complete, you may optionally remove the original JAI files from the GeoServer instance:

```
jai_core-x.y.z.jar
jai_imageio-x.y.jar
jai_codec-x.y.z.jar
```

where `x`, `y`, and `z` refer to specific version numbers.

17.2 Container Considerations

Java web containers such as Tomcat or Jetty ship with configurations that allow for fast startup, but don't always deliver the best performance.

17.2.1 Optimize your JVM

Set the following performance settings in the Java virtual machine (JVM) for your container. These settings are not specific to any container.

Option	Description
`-server`	Enables the server Java Virtual Machine (JVM), which compiles bytecode much earlier and with stronger optimizations. Startup and initial calls will be slower due to "just-in-time" (JIT) compilation taking longer, but subsequent calls will be faster.
`-Xmx256M -Xms48m`	Allocates extra memory to your server. By default, JVM will use only 64MB of heap. If you're serving just vector data, you'll be streaming, so having more memory won't increase performance. If you're serving coverages, however, JAI will use a disk cache. `-Xmx256M` allocates 256MB of memory to GeoServer (use more if you have excess memory). It is also a good idea to configure the JAI tile cache size (see the Server Config page in the *Web Administration Interface* section) so that it uses 75% of the heap (0.75). `-Xms48m` will tell the virtual machine to grab a 48MB heap on startup, which will make heap management more stable during heavy load serving.
`-XX:SoftRefLRUPolicyMSPerMB=36000`	Increases the lifetime of "soft references" in GeoServer. GeoServer uses soft references to cache datastore references and other similar requests. Making them live longer will increase the effectiveness of the cache.
`-XX:MaxPermSize=128m`	Increases the maximum size of permanent generation (or "permgen") allocated to GeoServer to 128MB. Permgen is the heap portion where the class bytecode is stored. GeoServer uses lots of classes, and it may exhaust that space quickly, leading to out of memory errors. This is especially important if you're deploying GeoServer along with other applications in the same container, or if you need to deploy multiple GeoServer instances inside the same container.
`-XX:+UseParallelGC`	Enables the throughput garbage collector.

For more information about JVM configuration, see the article Performance tuning garbage collection in Java.

17.3 Configuration Considerations

17.3.1 Use production logging

Logging may visibly affect the performance of your server. High logging levels are often necessary to track down issues, but by default you should run with low levels. (You can switch the logging levels while GeoServer is running.)

You can change the logging level in the *Web Administration Interface*. You'll want to choose the *PRODUCTION* logging configuration.

17.3.2 Set a service strategy

A service strategy is the method in which output is served to the client. This is a balance between proper form (being absolutely sure of reporting errors with the proper OGC codes, etc) and speed (serving output as quickly as possible). This is a decision to be made based on the function that GeoServer is providing. You can configure the service strategy by modifying the `web.xml` file of your GeoServer instance.

The possible strategies are:

Strategy	Description
`SPEED`	Serves output right away. This is the fastest strategy, but proper OGC errors are usually omitted.
`BUFFER`	Stores the whole result in memory, and then serves it after the output is complete. This ensures proper OGC error reporting, but delays the response quite a bit and can exhaust memory if the response is large.
`FILE`	Similar to `BUFFER`, but stores the whole result in a file instead of in memory. Slower than `BUFFER`, but ensures there won't be memory issues.
`PARTIAL-BUFFER`	A balance between `BUFFER` and `SPEED`, this strategy tries to buffer in memory a few KB of response, then serves the full output.

17.3.3 Personalize your server

This is isn't a performance consideration, but is just as important. In order to make GeoServer as useful as possible, you should customize the server's metadata to your organization. It may be tempting to skip some of the configuration steps, and leave in the same keywords and abstract as the sample, but this will only confuse potential users.

Suggestions:

- Fill out the WFS, WMS, and WCS Contents sections (this info will be broadcast as part of the capabilities documents)
- Serve your data with your own namespace (and provide a correct URI)
- Remove default layers (such as `topp:states`)

17.3.4 Configure service limits

Make sure clients cannot request an inordinate amount of resources from your server.

In particular:

- Set the maximum amount of features returned by each WFS GetFeature request (this can also be set on a per featuretype basis by modifying the `info.xml` files directly)
- Set the WMS `request limits` so that no request will consume too much memory or too much time

17.3.5 Set security

GeoServer includes support for WFS-T (transactions) by default, which lets users modify your data. If you don't want your database modified, you can turn off transactions in the the *Web Administration Interface*. Set the *Service Level* to `Basic`.

If you'd like some users to be able to modify some but not all of your data, you will have to set up an external security service. An easy way to accomplish this is to run two GeoServer instances and configure them differently, and use authentication to only allow certain users to have access.

For extra security, make sure that the connection to the datastore that is open to all is through a user who has read-only permissions. This will eliminate the possibility of a SQL injection (though GeoServer is generally not vulnerable to that sort of attack).

17.3.6 Cache your data

Server-side caching of WMS tiles is the best way to increase performance. In caching, pre-rendered tiles will be saved, eliminating the need for redundant WMS calls. There are several ways to set up WMS caching for GeoServer. GeoWebCache is the simplest method, as it comes bundled with GeoServer. (See the section on *Caching with GeoWebCache* for more details.) Another option is TileCache. You can also use a more generic caching system, such as OSCache (an embedded cache service) or Squid (a web cache proxy).

17.3.7 Disable the GeoServer web administration interface

In some circumstances, you might want to completely disable the web administration interface. There are two ways of doing this:

- Set the Java system property GEOSERVER_CONSOLE_DISABLED to true by adding -DGEOSERVER_CONSOLE_DISABLED=true to your container's JVM options
- Remove all of the web*-.jar files from WEB-INF/lib

17.4 Data Considerations

17.4.1 Use an external data directory

GeoServer comes with a built-in data directory. However, it is a good idea to separate the data from the application. Using an external data directory allows for much easier upgrades, since there is no risk of configuration information being overwritten. An external data directory also makes it easy to transfer your configuration elsewhere if desired. To point to an external data directory, you only need to edit the `web.xml` file. If you are new to GeoServer, you can copy (or just move) the data directory that comes with GeoServer to another location.

17.4.2 Use a spatial database

Shapefiles are a very common format for geospatial data. But if you are running GeoServer in a production environment, it is better to use a spatial database such as PostGIS. This is essential if doing transactions (WFS-T). Most spatial databases provide shapefile conversion tools. Although there are many options for spatial databases (see the section on *Working with Databases*), PostGIS is recommended. Oracle, DB2, and ArcSDE are also supported.

17.4.3 Pick the best performing coverage formats

There are very significant differences between performance of the various coverage formats.

Serving big coverage data sets with good performance requires some knowledge and tuning, since usually data is set up for distribution and archival. The following tips try to provide you with a base knowledge of how data restructuring affects performance, and how to use the available tools to get optimal data serving performance.

Choose the right format

The first key element is choosing the right format. Some formats are designed for data exchange, others for data rendering and serving. A good data serving format is binary, allows for multi-resolution extraction, and provides support for quick subset extraction at native resolutions.

Examples of such formats are GeoTiff, ECW, JPEG 2000 and MrSid. ArcGrid on the other hand is an example of format that's particularly ill-suited for large dataset serving (it's text based, no multi-resolution, and we have to read it fully even to extract a data subset in the general case).

GeoServer supports MrSID, ECW and JPEG 2000 through the GDAL Image Format plugin. MrSID is the easiest to work with, as their reader is now available under a GeoServer compatible open source format. If you have ECW files you have several non-ideal options. If you are only using GeoServer for educational or non-profit purposes you can use the plugin for free. If not you need to buy a license, since it's server software. You could also use GDAL to convert it to MrSID or tiled GeoTiffs. If your files are JPEG 2000 you can use the utilities of ECW and MrSID software. But the fastest is Kakadu, which requires a license.

Setup Geotiff data for fast rendering

As soon as your Geotiffs gets beyond some tens of megabytes you'll want to add the following capabilities:

- inner tiling
- overviews

Inner tiling sets up the image layout so that it's organized in tiles instead of simple stripes (rows). This allows much quicker access to a certain area of the geotiff, and the Geoserver readers will leverage this by accessing only the tiles needed to render the current display area. The following sample command instructs gdal_translate to create a tiled geotiff.

```
gdal_translate -of GTiff -projwin -180 90 -50 -10 -co "TILED=YES" bigDataSet.ecw myTiff.tiff
```

Overviews are downsampled version of the same image, that is, a zoomed out version, which is usually much smaller. When Geoserver needs to render the Geotiff, it'll look for the most appropriate overview as a starting point, thus reading and converting way less data. Overviews can be added using gdaladdo, or the the OverviewsEmbedded command included in Geotools. Here is a sample of using gdaladdo to add overviews that are downsampled 2, 4, 8 and 16 times compared to the original:

```
gdaladdo -r average mytiff.tif 2 4 8 16
```

For more hands on information on how to use GDAL utilites along with Geoserver, have a look at the BlueMarble data loading tutorial.

As a final note, Geotiff supports various kinds of compression, but we do suggest to not use it. Whilst it allows for much smaller files, the decompression process is expensive and will be performed on each data access, significantly slowing down rendering. In our experience, the decompression time is higher than the pure disk data reading.

Handling huge data sets

If you have really huge data sets (several gigabytes), odds are that simply adding overviews and tiles does not cut it, making intermediate resolution serving slow. This is because tiling occurs only on the native resolution levels, and intermediate overviews are too big for quick extraction.

So, what you need is a way to have tiling on intermediate levels as well. This is supported by the ImagePyramid plugin.

This plugin assumes you have create various seamless image mosaics, each for a different resolution level of the original image. In the mosaic, tiles are actual files (for more info about mosaics, see the *Using the ImageMosaic plugin*). The whole pyramid structures looks like the following:

```
rootDirectory
    +- pyramid.properties
    +- 0
       +- mosaic metadata files
       +- mosaic_file_0.tiff
       +- ...
       +- mosiac_file_n.tiff
    +- ...
    +- 32
       +- mosaic metadata files
       +- mosaic_file_0.tiff
       +- ...
       +- mosiac_file_n.tiff
```

Creating a pyramid by hand can theoretically be done with gdal, but in practice it's a daunting task that would require some scripting, since gdal provides no "tiler" command to extract regular tiles out of an image, nor one to create a downsampled set of tiles. As an alternative, you can use the geotools PyramidBuilder tool (documentation on how to use this is pending, contact the developers if you need to use it).

17.5 Linux init scripts

You will have to adjust the scripts to your environment. Download a script, rename it to `geoserver` and move it to `/etc/init.d`. Use **chmod** to make the script executable and test with **/etc/init.d/geoserver**.

To set different values for environment variables, create a file `/etc/default/geoserver` and specify your environment.

Example settings in `/etc/default/geoserver` for your environment:

```
USER=geoserver
GEOSERVER_DATA_DIR=/home/$USER/data_dir
GEOSERVER_HOME=/home/$USER/geoserver
```

```
JAVA_HOME=/usr/lib/jvm/java-6-sun
JAVA_OPTS="-Xms128m -Xmx512m"
```

17.5.1 Debian/Ubuntu

`Download the init script`

17.5.2 Suse

`Download the init script`

17.5.3 Starting GeoServer in Tomcat

`Download the init script`

17.6 Other Considerations

17.6.1 Host your application separately

GeoServer includes a few sample applications in the demo section of the *Web Administration Interface*. For production instances, we recommend against this bundling of your application. To make upgrades and troubleshooting easier, please use a separate container for your application. It is perfectly fine, though, to use one container manager (such as Tomcat or Jetty) to host both GeoServer and your application.

17.6.2 Proxy your server

GeoServer can have the capabilities documents properly report a proxy. You can configure this in the Server configuration section of the *Web Administration Interface* and entering the URL of the external proxy in the field labeled `Proxy base URL`.

17.6.3 Publish your server's capabilities documents

In order to make it easier to find your data, put a link to your capabilities document somewhere on the web. This will ensure that a search engine will crawl and index it.

17.6.4 Set up clustering

Setting up a Cluster is one of the best ways to improve the reliability and performance of your GeoServer installation. All the most stable and high performance GeoServer instances are configured in some sort of cluster. There are a huge variety of techniques to configure a cluster, including at the container level, the virtual machine level, and the physical server level.

Andrea Aime is currently working on an overview of what some of the biggest GeoServer users have done, for his 'GeoServer in Production' talk at FOSS4G 2009. In time that information will be migrated to tutorials and white papers.

17.7 Troubleshooting

17.7.1 Checking WFS requests

It often happens that users report issues with hand made WFS requests not working as expected. In the majority of the cases the request is malformed, but GeoServer does not complain and just ignores the malformed part (this behaviour is the default to make older WFS clients work fine with GeoServer).

If you want GeoServer to validate most WFS XML request you can post it to the following URL:

```
http://host:port/geoserver/ows?strict=true
```

Any deviation from the required structure will be noted in an error message. The only request type that is not validated in any case is the INSERT one (this is a GeoServer own limitation).

17.7.2 Leveraging GeoServer own log

GeoServer can generate a quite extensive log of its operations in the `$GEOSERVER_DATA_DIR/logs/geoserver.log` file. Looking into such file is one of the first things to do when troubleshooting a problem, in particular it's interesting to see the log contents in correspondence of a misbehaving request. The amount of information logged can vary based on the logging profile chosen in the *Server Settings* configuration page.

17.7.3 Logging service requests

GeoServer provides a request logging filter that is normally inactive. The filter can log both the requested URL and POST requests contents. Normally it is disabled due to its overhead. If you need to have an history of the incoming requests you can enable it by changing the `geoserver/WEB-INF/web.xml` contents to look like:

```
<filter>
  <filter-name>Request Logging Filter</filter-name>
  <filter-class>org.geoserver.filters.LoggingFilter</filter-class>
  <init-param>
      <param-name>enabled</param-name>
      <param-value>true</param-value>
  </init-param>
  <init-param>
      <param-name>log-request-bodies</param-name>
      <param-value>true</param-value>
  </init-param>
</filter>
```

This will log both the requests and the bodies, resulting in something like the following:

```
08 gen 11:30:13 INFO [geoserver.filters] - 127.0.0.1 "GET /geoserver/wms?HEIGHT=330&WIDTH=660&LAYERS=
08 gen 11:30:13 INFO [geoserver.filters] - 127.0.0.1 "GET /geoserver/wms?HEIGHT=330&WIDTH=660&LAYERS=
08 gen 11:30:14 INFO [geoserver.filters] - 127.0.0.1 "GET /geoserver/wms?REQUEST=GetFeatureInfo&EXCEP
08 gen 11:30:14 INFO [geoserver.filters] - 127.0.0.1 "GET /geoserver/wms?REQUEST=GetFeatureInfo&EXCEP
```

17.7.4 Using JDK tools to get stack and memory dumps

The JDK contains three useful command line tools that can be used to gather information about GeoServer instances that are leaking memory or not performing as requested: `jps`, `jstack` and `jmap`.

All tools work against a live Java Virtual Machine, the one running GeoServer in particular. In other for them to work properly you'll have to run them with a user that has enough privileges to connect to the JVM process, in particular super user or the same user that's running the JVM usually have the required right.

jps

jps is a tool listing all the Java processing running. It can be used to retried the `pid` (process id) of the virtual machine that is running GeoServer. For example:

```
> jps -mlv

16235 org.apache.catalina.startup.Bootstrap start -Djava.util.logging.manager=org.apache.juli.ClassLo
11521  -XX:MinHeapFreeRatio=10 -XX:MaxHeapFreeRatio=20 -Djava.library.path=/usr/lib/jni -Dosgi.requi
16287 sun.tools.jps.Jps -mlv -Dapplication.home=/usr/lib/jvm/java-6-sun-1.6.0.16 -Xms8m
```

The output shows the `pid`, the main class name if available, and the parameters passed to the JVM at startup. In this example `16235` is Tomcat hosting GeoServer, `11521` is an Eclipse instance, and `16287` is `jps` itself. In the common case you'll have only few JVM and the one running GeoServer can be identified by the parameters passed to it.

jstack

jstack is a tool extracting a the current stack trace for each thread running in the virtual machine. It can be used to identify scalability issues and to gather what the program is actually doing.

It usually takes people knowing about the inner workings of GeoServer can properly interpret the jstack output.

An example of usage:

```
> jstack -F -l 16235 > /tmp/tomcat-stack.txt
Attaching to process ID 16235, please wait...
Debugger attached successfully.
Server compiler detected.
JVM version is 14.2-b01
```

And the file contents might look like:

```
Deadlock Detection:

No deadlocks found.

Thread 16269: (state = BLOCKED)
 - java.lang.Object.wait(long) @bci=0 (Interpreted frame)
 - org.apache.tomcat.util.threads.ThreadPool$MonitorRunnable.run() @bci=10, line=565 (Interpreted fra
 - java.lang.Thread.run() @bci=11, line=619 (Interpreted frame)

Locked ownable synchronizers:
    - None

Thread 16268: (state = IN_NATIVE)
 - java.net.PlainSocketImpl.socketAccept(java.net.SocketImpl) @bci=0 (Interpreted frame)
 - java.net.PlainSocketImpl.accept(java.net.SocketImpl) @bci=7, line=390 (Interpreted frame)
 - java.net.ServerSocket.implAccept(java.net.Socket) @bci=60, line=453 (Interpreted frame)
 - java.net.ServerSocket.accept() @bci=48, line=421 (Interpreted frame)
 - org.apache.jk.common.ChannelSocket.accept(org.apache.jk.core.MsgContext) @bci=46, line=306 (Interp
 - org.apache.jk.common.ChannelSocket.acceptConnections() @bci=72, line=660 (Interpreted frame)
```

```
 - org.apache.jk.common.ChannelSocket$SocketAcceptor.runIt(java.lang.Object[]) @bci=4, line=870 (Inte
 - org.apache.tomcat.util.threads.ThreadPool$ControlRunnable.run() @bci=167, line=690 (Interpreted fr
 - java.lang.Thread.run() @bci=11, line=619 (Interpreted frame)

Locked ownable synchronizers:
    - None

Thread 16267: (state = BLOCKED)
 - java.lang.Object.wait(long) @bci=0 (Interpreted frame)
 - java.lang.Object.wait() @bci=2, line=485 (Interpreted frame)
 - org.apache.tomcat.util.threads.ThreadPool$ControlRunnable.run() @bci=26, line=662 (Interpreted fra
 - java.lang.Thread.run() @bci=11, line=619 (Interpreted frame)

Locked ownable synchronizers:
    - None

...
```

jmap

jmap is a tool to gather information about the a Java virtual machine. It can be used in a few interesting ways.

By running it without arguments (past the pid of the JVM) it will print out a **dump of the native libraries used by the JVM**. This can come in handy when one wants to double check GeoServer is actually using a certain version of a native library (e.g., GDAL):

```
> jmap 17251

Attaching to process ID 17251, please wait...
Debugger attached successfully.
Server compiler detected.
JVM version is 14.2-b01
0x08048000      46K     /usr/lib/jvm/java-6-sun-1.6.0.16/jre/bin/java
0x7f87f000      6406K   /usr/lib/jvm/java-6-sun-1.6.0.16/jre/lib/i386/libNCSEcw.so.0
0x7f9b2000      928K    /usr/lib/libstdc++.so.6.0.10
0x7faa1000      7275K   /usr/lib/jvm/java-6-sun-1.6.0.16/jre/lib/i386/libgdal.so.1
0x800e9000      1208K   /usr/lib/jvm/java-6-sun-1.6.0.16/jre/lib/i386/libclib_jiio.so
0x80320000      712K    /usr/lib/jvm/java-6-sun-1.6.0.16/jre/lib/i386/libNCSUtil.so.0
0x80343000      500K    /usr/lib/jvm/java-6-sun-1.6.0.16/jre/lib/i386/libNCSCnet.so.0
0x8035a000      53K     /lib/libgcc_s.so.1
0x8036c000      36K     /usr/lib/jvm/java-6-sun-1.6.0.16/jre/lib/i386/libnio.so
0x803e2000      608K    /usr/lib/jvm/java-6-sun-1.6.0.16/jre/lib/i386/libawt.so
0x80801000      101K    /usr/lib/jvm/java-6-sun-1.6.0.16/jre/lib/i386/libgdaljni.so
0x80830000      26K     /usr/lib/jvm/java-6-sun-1.6.0.16/jre/lib/i386/headless/libmawt.so
0x81229000      93K     /usr/lib/jvm/java-6-sun-1.6.0.16/jre/lib/i386/libnet.so
0xb7179000      74K     /usr/lib/jvm/java-6-sun-1.6.0.16/jre/lib/i386/libzip.so
0xb718a000      41K     /lib/tls/i686/cmov/libnss_files-2.9.so
0xb7196000      37K     /lib/tls/i686/cmov/libnss_nis-2.9.so
0xb71b3000      85K     /lib/tls/i686/cmov/libnsl-2.9.so
0xb71ce000      29K     /lib/tls/i686/cmov/libnss_compat-2.9.so
0xb71d7000      37K     /usr/lib/jvm/java-6-sun-1.6.0.16/jre/lib/i386/native_threads/libhpi.so
0xb71de000      184K    /usr/lib/jvm/java-6-sun-1.6.0.16/jre/lib/i386/libjava.so
0xb7203000      29K     /lib/tls/i686/cmov/librt-2.9.so
0xb725d000      145K    /lib/tls/i686/cmov/libm-2.9.so
0xb7283000      8965K   /usr/lib/jvm/java-6-sun-1.6.0.16/jre/lib/i386/server/libjvm.so
0xb7dc1000      1408K   /lib/tls/i686/cmov/libc-2.9.so
```

```
0xb7f24000      9K      /lib/tls/i686/cmov/libdl-2.9.so
0xb7f28000      37K     /usr/lib/jvm/java-6-sun-1.6.0.16/jre/lib/i386/jli/libjli.so
0xb7f32000      113K    /lib/tls/i686/cmov/libpthread-2.9.so
0xb7f51000      55K     /usr/lib/jvm/java-6-sun-1.6.0.16/jre/lib/i386/libverify.so
0xb7f60000      114K    /lib/ld-2.9.so
```

It's also possible to get a **quick summary of the JVM heap status**:

```
> jmap -heap 17251

Attaching to process ID 17251, please wait...
Debugger attached successfully.
Server compiler detected.
JVM version is 14.2-b01

using thread-local object allocation.
Parallel GC with 2 thread(s)

Heap Configuration:
   MinHeapFreeRatio = 40
   MaxHeapFreeRatio = 70
   MaxHeapSize      = 778043392 (742.0MB)
   NewSize          = 1048576 (1.0MB)
   MaxNewSize       = 4294901760 (4095.9375MB)
   OldSize          = 4194304 (4.0MB)
   NewRatio         = 8
   SurvivorRatio    = 8
   PermSize         = 16777216 (16.0MB)
   MaxPermSize      = 67108864 (64.0MB)

Heap Usage:
PS Young Generation
Eden Space:
   capacity = 42401792 (40.4375MB)
   used     = 14401328 (13.734176635742188MB)
   free     = 28000464 (26.703323364257812MB)
   33.96396076845054% used
From Space:
   capacity = 4718592 (4.5MB)
   used     = 2340640 (2.232208251953125MB)
   free     = 2377952 (2.267791748046875MB)
   49.60462782118056% used
To Space:
   capacity = 4587520 (4.375MB)
   used     = 0 (0.0MB)
   free     = 4587520 (4.375MB)
   0.0% used
PS Old Generation
   capacity = 43188224 (41.1875MB)
   used     = 27294848 (26.0303955078125MB)
   free     = 15893376 (15.1571044921875MB)
   63.19974630121396% used
PS Perm Generation
   capacity = 38404096 (36.625MB)
   used     = 38378640 (36.60072326660156MB)
   free     = 25456 (0.0242767333984375MB)
   99.93371540369027% used
```

In the result it can be seen that the JVM is allowed to use up to 742MB of memory, and that at the moment

the JVM is using 130MB (rough sum of the capacities of each heap section). In case of a persistent memory leak the JVM will end up using whatever is allowed to and each section of the heap will be almost 100% used.

To see **how the memory is actually being used in a succinct way** the following command can be used (on Windows, replace `head -25` with `more`):

```
> jmap -histo:live 17251 | head -25

 num     #instances         #bytes  class name
----------------------------------------------
   1:         81668       10083280  <constMethodKlass>
   2:         81668        6539632  <methodKlass>
   3:         79795        5904728  [C
   4:        123511        5272448  <symbolKlass>
   5:          7974        4538688  <constantPoolKlass>
   6:         98726        3949040  org.hsqldb.DiskNode
   7:          7974        3612808  <instanceKlassKlass>
   8:          9676        2517160  [B
   9:          6235        2465488  <constantPoolCacheKlass>
  10:         10054        2303368  [I
  11:         83121        1994904  java.lang.String
  12:         27794        1754360  [Ljava.lang.Object;
  13:          9227         868000  [Ljava.util.HashMap$Entry;
  14:          8492         815232  java.lang.Class
  15:         10645         710208  [S
  16:         14420         576800  org.hsqldb.CachedRow
  17:          1927         574480  <methodDataKlass>
  18:          8937         571968  org.apache.xerces.dom.ElementNSImpl
  19:         12898         561776  [[I
  20:         23122         554928  java.util.HashMap$Entry
  21:         16910         541120  org.apache.xerces.dom.TextImpl
  22:          9898         395920  org.apache.xerces.dom.AttrNSImpl
```

By the dump we can see most of the memory is used by the GeoServer code itself (first 5 items) followed by the HSQL cache holding a few rows of the EPSG database. In case of a memory leak a few object types will hold the vast majority of the live heap. Mind, to look for a leak the dump should be gathered with the server almost idle. If, for example, the server is under a load of GetMap requests the main memory usage will be the byte[] holding the images while they are rendered, but that is not a leak, it's legitimate and temporary usage.

In case of memory leaks a developer will probably ask for a **full heap dump** to analyze with a high end profiling tool. Such dump can be generated with the following command:

```
> jmap -dump:live,file=/tmp/dump.hprof 17251
Dumping heap to /tmp/dump.hprof ...
Heap dump file created
```

The dump files are generally as big as the memory used so it's advisable to compress the resulting file before sending it to a developer.

CHAPTER 18

Caching with GeoWebCache

GeoWebCache is a tiling server. It runs as a proxy between a map client and map server, caching (storing) tiles as they are requested, eliminating redundant request processing and thus saving large amounts of processing time. GeoWebCache is integrated with GeoServer, though it is also available as a standalone product for use with other map servers.

This section will discuss the version of GeoWebCache integrated with GeoServer. For information about the standalone product, please see the GeoWebCache homepage.

18.1 Using GeoWebCache

Note: For an more in-depth discussion of using GeoWebCache, please see the GeoWebCache documentation.

18.1.1 Direct integration with GeoServer WMS

GeoWebCache can be transparently integrated with the GeoServer WMS, and so requires no special endpoint or custom URL. In this way one can have the simplicity of a standard WMS endpoint with the performance of a tiled client.

Although this direct integration is disabled by default, it can be enabled by going to the *Caching defaults* page in the *Web Administration Interface*.

When this feature is enabled, GeoServer WMS will cache and retrieve tiles from GeoWebCache (via a GetMap request) only if **all of the following criteria are followed**:

- WMS Direct integration is enabled (you can set this on the *Caching defaults* page)
- `tiled=true` is included in the request
- The request only references a single layer
- Caching is enabled for that layer
- The image requested is of the same height and width as the size saved in the layer configuration
- The requested CRS matches one of the available tile layer gridsets

- The image requested lines up with the existing grid bounds
- A parameter is included for which there is a corresponding Parameter Filter

In addition, when direct integration is enabled, the WMS capabilities document (via a GetCapabilities request) will only return the WMS-C vendor-specific capabilities elements (such as a `<TileSet>` element for each cached layer/CRS/format combination) if `tiled=true` is appended to the GetCapabilities request.

Note: For more information on WMS-C, please see the WMS Tiling Client Recommendation from OSGeo.

Note: GeoWebCache integration is not compatible with the OpenLayers-based *Layer Preview*, as the preview does not usually align with the GeoWebCache layer gridset. This is because the OpenLayers application calculates the `tileorigin` based on the layer's bounding box, which is different from the gridset. It is, possible to create an OpenLayers application that caches tiles; just make sure that the `tileorigin` aligns with the gridset.

Virtual services

When direct WMS integration is enabled, GeoWebCache will properly handle requests to *Virtual OWS Services* (`/geoserver/<workspace>/wms?tiled=true&...`).

Virtual services capabilities documents will contain `<TileSet>` entries only for the layers that belong to that workspace (and global layer groups), and will be referenced by unqualified layer names (no namespace). For example, the layer `topp:states` will be referred to as `<Layers>states</Layers>` instead of `<Layers>topp:states</Layers>`, and GetMap requests to the virtual services endpoint using `LAYERS=states` will properly be handled.

Supported parameter filters

With direct WMS integration, the following parameter filters are supported for GetMap requests:

- `ANGLE`
- `BGCOLOR`
- `BUFFER`
- `CQL_FILTER`
- `ELEVATION`
- `ENV`
- `FEATUREID`
- `FEATUREVERSION`
- `FILTER`
- `FORMAT_OPTIONS`
- `MAXFEATURES`
- `PALETTE`
- `STARTINDEX`
- `TIME`
- `VIEWPARAMS`

If a request is made using any of the above parameters, the request will be passed to GeoServer, unless a parameter filter has been set up, in which case GeoWebCache will process the request.

18.1.2 GeoWebCache endpoint URL

When not using direct integration, you can point your client directly to GeoWebCache.

> **Warning:** GeoWebCache is not a true WMS, and so the following is an oversimplification. If you encounter errors, see the *Troubleshooting* page for help.

To direct your client to GeoWebCache (and thus receive cached tiles) you need to change the WMS URL.

If your application requests WMS tiles from GeoServer at this URL:

```
http://example.com/geoserver/wms
```

You can invoke the GeoWebCache WMS instead at this URL:

```
http://example.com/geoserver/gwc/service/wms
```

In other words, add `/gwc/service/wms` in between the path to your GeoServer instance and the WMS call.

As soon as tiles are requested through GeoWebCache, GeoWebCache automatically starts saving them. This means that initial requests for tiles will not be accelerated since GeoServer will still need to generate the tiles. To automate this process of requesting tiles, you can **seed** the cache. See the section on *Seeding and refreshing* for more details.

18.1.3 Disk quota

GeoWebCache has a built-in disk quota feature to prevent disk space from growing unbounded. You can set the maximum size of the cache directory, poll interval, and what policy of tile removal to use when the quota is exceeded. Tiles can be removed based on usage ("Least Frequently Used" or LFU) or timestamp ("Least Recently Used" or LRU).

Disk quotas are turned off by default, but can be configured on the *Disk Quotas* page in the *Web Administration Interface*.

18.1.4 Integration with external mapping sites

The documentation on the GeoWebCache homepage contains examples for creating applications that integrate with Google Maps, Google Earth, Bing Maps, and more.

18.1.5 Support for custom projections

The version of GeoWebCache that comes embedded in GeoServer automatically configures every layer served in GeoServer with the two most common projections:

- **EPSG:4326** (latitude/longitude)
- **EPSG:900913** (Spherical Mercator, the projection used in Google Maps)

You can also set a custom CRS from any that GeoServer recognizes. See the *Gridsets* page for details.

18.2 Configuration

GeoWebCache is automatically configured for use with GeoServer using the most common options, with no setup required. All communication between GeoServer and GeoWebCache happens by passing messages inside the JVM.

By default, all layers served by GeoServer will be known to GeoWebCache. See the *Tile Layers* page to test the configuration.

Note: Version 2.2.0 of GeoServer introduced changes to the configuration of the integrated GeoWebCache.

18.2.1 Integrated user interface

GeoWebCache has a full integrated web-based configuration. See the *Tile Caching* section in the *Web Administration Interface*.

18.2.2 Determining tiled layers

In versions of GeoServer prior to 2.2.0, the GeoWebCache integration was done in a such way that every GeoServer layer and layer group was forced to have an associated GeoWebCache tile layer. In addition, every such tile layer was forcedly published in the EPSG:900913 and EPSG:4326 gridsets with PNG and JPEG output formats.

It is possible to selectively turn caching on or off for any layer served through GeoServer. This setting can be configured in the *Tile Layers* section of the *Web Administration Interface*.

18.2.3 Configuration files

It is possible to configure most aspects of cached layers through the *Tile Caching* section in the *Web Administration Interface* or the *GeoWebCache REST API*.

GeoWebCache keeps the configuration for each GeoServer tiled layer separately, inside the `<data_dir>/gwc-layers/` directory. There is one XML file for each tile layer. These files contain a different syntax from the `<wmsLayer>` syntax in the standalone version and are *not* meant to be edited by hand. Instead you can configure tile layers on the *Tile Layers* page or through the *GeoWebCache REST API*.

Configuration for the defined gridsets is saved in `<data_dir>/gwc/geowebcache.xml` so that the integrated GeoWebCache can continue to serve externally-defined tile layers from WMS services outside GeoServer.

If upgrading from a version prior to 2.2.0, a migration process is run which creates a tile layer configuration for all the available layers and layer groups in GeoServer with the old defaults. From that point on, you should configure the tile layers on the *Tile Layers* page.

18.2.4 Changing the cache directory

GeoWebCache will automatically store cached tiles in a `gwc` directory inside your GeoServer data directory. To set a different directory, stop GeoServer (if it is running) and add the following code to your GeoServer `web.xml` file (located in the `WEB-INF` directory):

```
<context-param>
   <param-name>GEOWEBCACHE_CACHE_DIR</param-name>
   <param-value>C:\temp</param-value>
</context-param>
```

Change the path inside `<param-value>` to the desired cache path (such as `C:\temp` or `/tmp`). Restart GeoServer when done.

Note: Make sure GeoServer has write access in this directory.

18.2.5 GeoWebCache with multiple GeoServer instances

For stability reasons, it is not recommended to use the embedded GeoWebCache with multiple GeoServer instances. If you want to configure GeoWebCache as a front-end for multiple instances of GeoServer, we recommend using the standalone GeoWebCache.

18.2.6 Geoserver Data Security

GWC Data Security is an option that can be turned on and turned off through the *Caching defaults* page. By default it is turned off.

When turned on, the embedded GWC will do a data security check before calling GeoWebCache, i.e. verify whether the user actually has access to the layer, and reject the request if this is not the case. In the case of WMS-C requests, there is also limited support for data access limit filters, only with respect to geographic boundaries (all other types of data access limits will be ignored). The embedded GWC will reject requests for which the requested bounding box is (partly) inaccessible. It is only possible to request a tile within a bounding box that is fully accessible. This behaviour is different from the regular WMS, which will filter the data before serving it. However, if the integrated WMS/WMS-C is used, the request will be forwarded back to WMS and give the desired result.

When using the default GeoServer security system, rules cannot combine data security with service security. However, when using a security subsystem it may be possible to make such particular combinations. In this case the WMS-C service inherits all security rules from the regular WMS service; while all other GWC services will get their security from rules associated with the 'GWC' service itself.

18.3 Seeding and refreshing

The primary benefit to GeoWebCache is that it allows for the acceleration of normal WMS tile request processing by eliminating the need for the tiles to be regenerated for every request. This page discusses tile generation.

You can configure seeding processes via the *Web Administration Interface*. See the *Tile Layers* page for more information.

18.3.1 Generating tiles

There are two ways for tiles to be generated by GeoWebCache. The first way for tiles to be generated is during **normal map viewing**. In this case, tiles are cached only when they are requested from a client, either through map browsing (such as in OpenLayers) or through manual WMS tile requests. The first time a map view is requested it will be roughly at the same speed as a standard GeoServer WMS request. The second and subsequent map viewings will be greatly accelerated as those tiles will have already been

generated. The main advantage to this method is that it requires no preprocessing, and that only the data that has been requested will be cached, thus potentially saving disk space as well. The disadvantage to this method is that map viewing will be only intermittently accelerated, reducing the quality of user experience.

The other way for tiles to be generated is by **seeding**. Seeding is the process where map tiles are generated and cached internally from GeoWebCache. When processed in advance, the user experience is greatly enhanced, as the user never has to wait for tiles to be generated. The disadvantage to this process is that seeding can be a very time- and disk-consuming process.

In practice, a combination of both methods are usually used, with certain zoom levels (or certain areas of zoom levels) seeded, and the less-likely-viewed tiles are left uncached.

18.4 HTTP Response Headers

The GeoWebCache integrated with GeoServer employs special information stored in the header of responses. These headers are available either with direct calls to the *GeoWebCache endpoint* or with *direct WMS integration*.

18.4.1 Custom response headers

GeoWebCache returns both standard and custom HTTP response headers when serving a tile request. This aids in the debugging process, as well as adhering to an HTTP 1.1 transfer control mechanism.

The response headers can be determined via a utility such as cURL.

Example

Note: For all cURL commands below, make sure to replace `>/dev/null` with `>nul` if you are running on Windows.

This is a sample request and response using cURL:

```
curl -v "http://localhost:8080/geoserver/gwc/service/wms?LAYERS=sde%3Abmworld&FORMAT=image%2Fpng&SERV

< HTTP/1.1 200 OK
< geowebcache-tile-index: [0, 1, 2]
< geowebcache-cache-result: HIT
< geowebcache-tile-index: [0, 1, 2]
< geowebcache-tile-bounds: -180.0,-38.0,-52.0,90.0
< geowebcache-gridset: GlobalCRS84Pixel
< geowebcache-crs: EPSG:4326
< Content-Type: image/png
< Content-Length: 102860
< Server: Jetty(6.1.8)
```

From this, one can learn that the tile was found in the cache (`HIT`), the requested tile was from the gridset called `GlobalCRS84Pixel` and had a CRS of `EPSG:4326`.

List of custom response headers

The following is the full list of custom response headers. Whenever GeoWebCache serves a tile request, it will write some or all of the following custom headers on the HTTP response.

Response Header	Description
geowebcache-cache-result	Shows whether the GeoWebCache WMS was used. Options are: • HIT: Tile requested was found on the cache • MISS: Tile was not found on the cache but was acquired from the layer's data source • WMS: Request was proxied directly to the origin WMS (for example, for GetFeatureInfo requests) • OTHER: Response was the default white/transparent tile or an error occurred
geowebcache-tile-index	Contains the three-dimensional tile index in x,y,z order of the returned tile image in the corresponding grid space (e.g. [1, 0, 0])
geowebcache-tile-bounds	Bounds of the returned tile in the corresponding coordinate reference system (e.g. -180,-90,0,90)
geowebcache-gridset	Name of the gridset the tile belongs to (see *Gridsets* for more information)
geowebcache-crs	Coordinate reference system code of the matching gridset (e.g. EPSG:900913, EPSG:4326, etc).

18.4.2 Last-Modified and If-Modified-Since

Well behaved HTTP 1.1 clients and server applications can make use of `Last-Modified` and `If-Modified-Since` HTTP control mechanisms to know when locally cached content is up to date, eliminating the need to download the same content again. This can result in considerable bandwidth savings. (See HTTP 1.1 RFC 2616, sections 14.29 and 14.25, for more information on these mechanisms.)

GeoWebCache will write a `Last-Modified` HTTP response header when serving a tile image. The date is written as an RFC-1123 `HTTP-Date`:

```
Last-Modified: Wed, 15 Nov 1995 04:58:08 GMT
```

Clients connecting to GeoWebCache can create a "conditional GET" request with the `If-Modified-Since` request header. If the tile wasn't modified after the date specified in the `Last-Modified` response header, GeoWebCache will return a `304` status code indicating that the resource was available and not modified.

Example

A query for a specific tile returns the `Last-Modified` response header:

```
curl -v "http://localhost:8080/geoserver/gwc/service/wms?LAYERS=img%20states&FORMAT=image%2Fpng&SERV

> Host: localhost:8080
> Accept: */*
>
< HTTP/1.1 200 OK
...
< Last-Modified: Wed, 25 Jul 2012 00:42:00 GMT
< Content-Type: image/png
< Content-Length: 31192
```

This request has the `If-Modified-Since` header set to one second after what was returned by `Last-Modified`:

```
curl --header "If-Modified-Since: Wed, 25 Jul 2012 00:42:01 GMT" -v "http://localhost:8080/geoserver/

> Host: localhost:8080
> Accept: */*
> If-Modified-Since: Wed, 25 Jul 2012 00:42:01 GMT
>
< HTTP/1.1 304 Not Modified
< Last-Modified: Wed, 25 Jul 2012 00:42:00 GMT
< Content-Type: image/png
< Content-Length: 31192
```

The response code is `304`. As the file hasn't been modified since the time specified in the request, no content is actually transferred. The client is informed that its copy of the tile is up to date.

However, if you were to set the `If-Modified-Since` header to *before* the time stored in `Last-Modified`, you will instead receive a `200` status code and the tile will be downloaded.

This example sets the `If-Modified-Since` header to one second before what was returned by `Last-Modified`:

```
curl --header "If-Modified-Since: Wed, 25 Jul 2012 00:41:59 GMT" -v "http://localhost:8080/geoserver/

> Host: localhost:8080
> Accept: */*
> If-Modified-Since: Wed, 25 Jul 2012 00:41:59 GMT
>
< HTTP/1.1 200 OK
...
< Last-Modified: Wed, 25 Jul 2012 00:42:00 GMT
< Content-Type: image/png
< Content-Length: 31192
```

18.5 GeoWebCache REST API

This section discusses the GeoWebCache REST API, an interface for working programmatically with the integrated GeoWebCache without the need for a GUI.

The GeoWebCache REST endpoint when integrated with GeoServer is available at:

```
<GEOSERVER_HOME>/gwc/rest/
```

For example:

```
http://example.com:8080/geoserver/gwc/rest/
```

18.5.1 Managing Layers

The GeoWebCache REST API provides a RESTful interface through which users can add, modify, or remove cached layers.

Note: JSON is not recommended for managing layers as the JSON library has a number of issues with multi-valued properties such as "parameterFilters".

Layer list

URL: `/gwc/rest/seed/layers.xml`

Method	Action	Return Code	Formats
GET	Return the list of available layers	200	XML
POST		400	
PUT		400	
DELETE		400	

The following example will request a full list of layers:

```
curl -u admin:geoserver "http://localhost:8080/geoserver/gwc/rest/layers"
```

```
<layers>
  <layer>
    <name>img states</name>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" href="http://localhost:8080/g
  </layer>
  <layer>
    <name>raster test layer</name>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" href="http://localhost:8080/g
  </layer>
  <layer>
    <name>topp:states</name>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" href="http://localhost:8080/g
  </layer>
 </layers>
```

Layer Operations

URL: `/gwc/rest/seed/layers/<layer>.xml`

Note: JSON is not recommended for managing layers as the JSON library has a number of issues with multi-valued properties such as "parameterFilters".

Method	Action	Return Code	Formats
GET	Return the XML representation of the layer	200	XML
POST	Modify the definition/configuration of the layer	200	XML
PUT	Add a new layer	200	XML
DELETE	Delete the layer	200	

Note: There are two different representations for cached layers, depending on whether the tile layer is created from the GeoServer WMS layer or layer group (`GeoServerLayer`), or is configured in `geowebcache.xml` as a regular GWC layer (`wmsLayer`). A GeoServer layer is referred to as a `GeoServerLayer` and contains no image data source information such as origin WMS URL.

Representations:

- GeoWebCache (`wmsLayer`) `XML minimal`
- GeoWebCache (`wmsLayer`) `XML`
- GeoServer (`GeoServerLayer`) `XML minimal`
- GeoServer (`GeoServerLayer`) `XML`

The examples below use the cURL tool, though the examples apply to any HTTP-capable tool or library.

Adding a GeoWebCache layer

The following example will add a new layer to GeoWebCache:

```
curl -v -u admin:geoserver -XPUT -H "Content-type: text/xml" -d @layer.xml  "http://localhost:8080/ge
```

The `layer.xml` file is defined as the following:

```
<wmsLayer>
  <name>newlayer</name>
  <mimeFormats>
    <string>image/png</string>
  </mimeFormats>
  <gridSubsets>
    <gridSubset>
      <gridSetName>EPSG:900913</gridSetName>
    </gridSubset>
  </gridSubsets>
  <wmsUrl>
    <string>http://localhost:8080/geoserver/wms</string>
  </wmsUrl>
  <wmsLayers>topp:states</wmsLayers>
</wmsLayer>
```

Note: The addressed resource (`newlayer` in this example) must match the name of the layer in the XML representation.

Adding a GeoServer layer

The following example will add a new layer to both GeoServer and GeoWebCache:

```
curl -v -u admin:geoserver -XPUT -H "Content-type: text/xml" -d @poi.xml  "http://localhost:8080/geos
```

The `poi.xml` file is defined as the following:

```
<GeoServerLayer>
  <id>LayerInfoImpl--570ae188:124761b8d78:-7fd0</id>
  <enabled>true</enabled>
  <name>tiger:poi</name>
  <mimeFormats>
    <string>image/png8</string>
  </mimeFormats>
  <gridSubsets>
    <gridSubset>
      <gridSetName>GoogleCRS84Quad</gridSetName>
      <zoomStart>0</zoomStart>
      <zoomStop>14</zoomStop>
      <minCachedLevel>1</minCachedLevel>
      <maxCachedLevel>9</maxCachedLevel>
    </gridSubset>
  </gridSubsets>
  <metaWidthHeight>
    <int>4</int>
    <int>4</int>
  </metaWidthHeight>
  <gutter>50</gutter>
```

```
  <autoCacheStyles>true</autoCacheStyles>
</GeoServerLayer>
```

Note: The addressed resource (`tiger:poi` in this example) must match the name of the layer in the XML representation, as well as the name of an *existing* GeoServer layer or layer group.

Modifying a layer

This example modifies the layer definition via the `layer.xml` file. The request adds a parameter filter and a grid subset to the existing `tiger:poi` tile layer:

```
<GeoServerLayer>
 <enabled>true</enabled>
 <name>tiger:poi</name>
 <mimeFormats>
   <string>image/png8</string>
 </mimeFormats>
 <gridSubsets>
   <gridSubset>
     <gridSetName>GoogleCRS84Quad</gridSetName>
     <zoomStart>0</zoomStart>
     <zoomStop>14</zoomStop>
     <minCachedLevel>1</minCachedLevel>
     <maxCachedLevel>9</maxCachedLevel>
   </gridSubset>
   <gridSubset>
     <gridSetName>EPSG:900913</gridSetName>
     <extent>
       <coords>
         <double>-8238959.403861314</double>
         <double>4969300.121476209</double>
         <double>-8237812.689219721</double>
         <double>4971112.167757057</double>
       </coords>
     </extent>
   </gridSubset>
 </gridSubsets>
 <metaWidthHeight>
   <int>4</int>
   <int>4</int>
 </metaWidthHeight>
 <parameterFilters>
   <floatParameterFilter>
     <key>ELEVATION</key>
     <defaultValue>0.0</defaultValue>
     <values>
       <float>0.0</float>
       <float>1.0</float>
       <float>2.0</float>
       <float>3.0</float>
       <float>4.0</float>
     </values>
     <threshold>1.0E-3</threshold>
   </floatParameterFilter>
 </parameterFilters>
 <gutter>50</gutter>
```

```
 <autoCacheStyles>true</autoCacheStyles>
</GeoServerLayer>
```

Instead of PUT, use the HTTP POST method instead:

```
curl -v -u admin:geoserver -XPOST -H "Content-type: text/xml" -d @poi.xml  "http://localhost:8080/geo
```

Deleting a layer

Deleting a GeoWebCache tile layer deletes the layer configuration *as well as the layer's disk cache*. No tile images will remain in the cache directory after deleting a tile layer.

To delete a layer, use the HTTP DELETE method against the layer resource:

```
curl -v -u admin:geoserver -XDELETE "http://localhost:8080/geoserver/gwc/rest/layers/newlayer.xml"
```

Note: If trying to delete a tile layer that is an integrated `GeoServerLayer`, only the GeoWebCache layer definition will be deleted; the GeoServer definition is left untouched. To delete a layer in GeoServer, use the GeoServer *REST configuration* to manipulate GeoServer resources.

On the other hand, deleting a GeoServer layer via the GeoServer REST API *will* automatically delete the associated tile layer.

18.5.2 Seeding and Truncating

The GeoWebCache REST API provides a RESTful interface through which users can add or remove tiles from the cache on a per-layer basis.

Operations

URL: `/gwc/rest/seed/<layer>.<format>`

Method	Action	Return Code	Formats
GET	Return the status of the seeding threads	200	JSON
POST	Issue a seed or truncate task request	200	XML, JSON
PUT		405	
DELETE		405	

Representations:

- `XML`
- `JSON`

The examples below use the cURL tool, though the examples apply to any HTTP-capable tool or library.

Seeding

The following XML request initiates a seeding task:

```
curl -v -u admin:geoserver -XPOST -H "Content-type: text/xml" -d '<seedRequest><name>nurc:Arc_Sample<
```

```
* About to connect() to localhost port 8080 (#0)
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 8080 (#0)
* Server auth using Basic with user 'admin'
> POST /geoserver/gwc/rest/seed/nurc:Arc_Sample.xml HTTP/1.1
> Authorization: Basic YWRtaW46Z2Vvc2VydmVy
> User-Agent: curl/7.21.3 (x86_64-pc-linux-gnu) libcurl/7.21.3 OpenSSL/0.9.8o zlib/1.2.3.4 libidn/1.1
> Host: localhost:8080
> Accept: */*
> Content-type: text/xml
> Content-Length: 209
>
< HTTP/1.1 200 OK
```

The following is a more complete XML fragment for a seed request, including parameter filters:

```
<?xml version="1.0" encoding="UTF-8"?>
<seedRequest>
  <name>topp:states</name>
  <bounds>
    <coords>
      <double>-2495667.977678598</double>
      <double>-2223677.196231552</double>
      <double>3291070.6104286816</double>
      <double>959189.3312465074</double>
    </coords>
  </bounds>

  <!-- These are listed on http://localhost:8080/geoserver/gwc/demo -->
  <gridSetId>EPSG:2163</gridSetId>
  <zoomStart>0</zoomStart>
  <zoomStop>2</zoomStop>
  <format>image/png</format>

  <!-- type can be seed, reseed, or truncate -->
  <type>truncate</type>

  <!-- Number of seeding threads to run in parallel.
       If type == truncate only one thread will be used
       regardless of this parameter -->
  <threadCount>1</threadCount>
  <!-- Parameter filters -->
  <parameters>
    <entry>
      <string>STYLES</string>
      <string>pophatch</string>
    </entry>
    <entry>
      <string>CQL_FILTER</string>
      <string>TOTPOP > 10000</string>
    </entry>
  </parameters>
</seedRequest>
```

Truncating

The following XML request initiates a seeding task:

```
curl -v -u admin:geoserver -XPOST -H "Content-type: application/json" -d "{'seedRequest':{'name':'top

* About to connect() to localhost port 8080 (#0)
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 8080 (#0)
* Server auth using Basic with user 'admin'
> POST /geoserver/gwc/rest/seed/nurc:Arc_Sample.json HTTP/1.1
> Authorization: Basic YWRtaW46Z2Vvc2VydmVy
> User-Agent: curl/7.21.3 (x86_64-pc-linux-gnu) libcurl/7.21.3 OpenSSL/0.9.8o zlib/1.2.3.4 libidn/1.1
> Host: localhost:8080
> Accept: */*
> Content-type: application/json
> Content-Length: 205
>
< HTTP/1.1 200 OK
< Date: Fri, 14 Oct 2011 22:09:21 GMT
< Server: Noelios-Restlet-Engine/1.0..8
< Transfer-Encoding: chunked
<
* Connection #0 to host localhost left intact
* Closing connection #0
```

Querying running tasks

URL: `/gwc/rest/seed[/<layer>].json`

Method	Action	Return Code	Formats
GET	Get the global or per layer state of running and pending tasks	200	JSON
POST		405	
PUT		405	
DELETE		405	

Getting current state of the seeding threads

Sending a GET request to the `/gwc/rest/seed.json` resource returns a list of pending (scheduled) and running tasks for all the layers.

Sending a GET request to the `/gwc/rest/seed/<layer name>.json` resource returns a list of pending (scheduled) and running tasks for that specific layer.

The returned content is a JSON array of the form:

```
{"long-array-array":[[<long>,<long>,<long>,<long>,<long>],...]}
```

If there are no pending or running tasks, the returned array is empty:

```
{"long-array-array":[]}
```

The returned array of arrays contains one array per seeding/truncating task. The meaning of each long value in each thread array is:

```
[tiles processed, total # of tiles to process, # of remaining tiles, Task ID, Task status]
```

The returned `Task Status` value will be one of the following:

```
-1 = ABORTED
 0 = PENDING
 1 = RUNNING
 2 = DONE
```

The example below returns the current state of tasks for the `topp:states` layer:

```
curl -u <user>:<password> -v -XGET http://localhost:8080/geoserver/gwc/rest/seed/topp:states.json
```

```
{"long-array-array":[[17888,44739250,18319,1,1],[17744,44739250,18468,2,1],[16608,44739250,19733,3,0
```

In the above response, tasks `1` and `2` for the `topp:states` layer are running, and tasks `3` and `4` are in a pending state waiting for an available thread.

The example below returns a list of tasks for all the layers.

```
curl -u <user>:<password> -XGET http://localhost:8080/geoserver/gwc/rest/seed.json
```

```
{"long-array-array":[[2240,327426,1564,2,1],[2368,327426,1477,3,1],[2272,327426,1541,4,1],[2176,32742
```

Terminating running tasks

URL: `/gwc/rest/seed[/<layer>]`

Method	Action	Return Code	Formats
GET	Issue a kill running and/or pending tasks request	405	
POST		200	
PUT		405	
DELETE		405	

A POST request to the `/gwc/rest/seed` resource terminates pending and/or running tasks for **all layers**. A POST request to the `/gwc/rest/seed/<layername>` resource terminates pending and/or running tasks for a specific layer.

It is possible to terminate individual or all pending and/or running tasks. Use the parameter `kill_all` with one of the following values: `running`, `pending`, or `all`.

Note: For backward compatibility, the `kill_all` parameter value `1` is also accepted and is equivalent to `running`.

The following request terminates all running seed and truncate tasks.

```
curl -v -u admin:geoserver -d "kill_all=all"  "http://localhost:8080/geoserver/gwc/rest/seed"
```

```
* About to connect() to localhost port 8080 (#0)
*   Trying 127.0.0.1... connected
< HTTP/1.1 200 OK
< Date: Fri, 14 Oct 2011 22:23:04 GMT
< Server: Noelios-Restlet-Engine/1.0..8
< Content-Type: text/html; charset=ISO-8859-1
< Content-Length: 426
<
<html>
...
* Connection #0 to host localhost left intact
* Closing connection #0
```

18.5.3 Disk Quota

The GeoWebCache REST API provides a RESTful interface through which users can configure the disk usage limits and expiration policies for a GeoWebCache instance.

Operations

URL: `/gwc/rest/diskquota.<format>`

Method	Action	Return Code	Formats
GET	Return the global disk quota configuration	200	XML, JSON
POST		405	
PUT	Modify the global disk quota configuration	200	XML, JSON
DELETE		405	

Representations:

- `XML`
- `JSON`

The examples below use the cURL tool, though the examples apply to any HTTP-capable tool or library.

Retrieving the current configuration

The following returns the current disk quota configuration in **XML** format:

```
curl -u admin:geoserver -v -XGET http://localhost:8080/geoserver/gwc/rest/diskquota.xml

< HTTP/1.1 200 OK
< Date: Mon, 21 Mar 2011 13:50:49 GMT
< Server: Noelios-Restlet-Engine/1.0..8
< Content-Type: text/xml; charset=ISO-8859-1
< Content-Length: 422
<
<gwcQuotaConfiguration>
  <enabled>true</enabled>
  <diskBlockSize>2048</diskBlockSize>
  <cacheCleanUpFrequency>5</cacheCleanUpFrequency>
  <cacheCleanUpUnits>SECONDS</cacheCleanUpUnits>
  <maxConcurrentCleanUps>5</maxConcurrentCleanUps>
  <globalExpirationPolicyName>LRU</globalExpirationPolicyName>
  <globalQuota>
    <value>100</value>
    <units>MiB</units>
  </globalQuota>
  <layerQuotas/>
</gwcQuotaConfiguration>
```

The following returns the current disk quota configuration in **JSON** format:

```
curl -u admin:geoserver -v -XGET http://localhost:8080/geoserver/gwc/rest/diskquota.json

< HTTP/1.1 200 OK
< Date: Mon, 21 Mar 2011 13:53:42 GMT
< Server: Noelios-Restlet-Engine/1.0..8
< Content-Type: application/json; charset=ISO-8859-1
< Content-Length: 241
```

```
<
* Connection #0 to host localhost left intact
* Closing connection #0
{"gwcQuotaConfiguration":{"diskBlockSize":2048,"enabled":true,"maxConcurrentCleanUps":5,"cacheCleanUp
```

Changing configuration

Note: The request body for PUT should contain only the desired properties to be modified. For example, the following will only change the maxConcurrentCleanups property in XML format:

```
<gwcQuotaConfiguration><maxConcurrentCleanUps>2</maxConcurrentCleanUps></gwcQuotaConfiguration>
```

The following will only change the diskBlockSize, enabled, and globalQuota properties in JSON format:

```
{"gwcQuotaConfiguration":{"diskBlockSize":2048,"enabled":true,"globalQuota":{"value":"100","units":"N
```

The following XML example successfully enables the quota and sets the globalQuota size:

```
curl -v -u admin:geoserver "http://localhost:8090/geoserver/gwc/rest/diskquota.xml" -X PUT -d "<gwcQu

< HTTP/1.1 200 OK
< Date: Fri, 18 Mar 2011 20:59:31 GMT
< Server: Noelios-Restlet-Engine/1.0..8
< Content-Type: text/xml; charset=ISO-8859-1
< Content-Length: 422
<
<gwcQuotaConfiguration>
  <enabled>true</enabled>
  <diskBlockSize>2048</diskBlockSize>
  <cacheCleanUpFrequency>5</cacheCleanUpFrequency>
  <cacheCleanUpUnits>SECONDS</cacheCleanUpUnits>
  <maxConcurrentCleanUps>5</maxConcurrentCleanUps>
  <globalExpirationPolicyName>LFU</globalExpirationPolicyName>
  <globalQuota>
    <value>100</value>
    <units>GiB</units>
  </globalQuota>
  <layerQuotas/>
</gwcQuotaConfiguration>
```

The following JSON example changes the globalQuote and expirationPolicyName parameters:

```
curl -v -u admin:geoserver "http://localhost:8090/geoserver/gwc/rest/diskquota.json" -X PUT -d "{"gwc

< HTTP/1.1 200 OK
< Date: Fri, 18 Mar 2011 21:02:20 GMT
< Server: Noelios-Restlet-Engine/1.0..8
< Content-Type: application/json; charset=ISO-8859-1
< Content-Length: 241
<
* Connection #0 to host localhost left intact
* Closing connection #0
{"gwcQuotaConfiguration":{"diskBlockSize":2048,"enabled":true,"maxConcurrentCleanUps":5,"cacheCleanUp
```

The following *invalid* XML example has an invalid parameter (maxConcurrentCleanUps must be > 0). It returns a 400 response code and contains an error message as plain text:

```
curl -v -u admin:geoserver "http://localhost:8090/geoserver/gwc/rest/diskquota.xml" -X PUT -d "<gwcQu

< HTTP/1.1 400 Bad Request
< Date: Fri, 18 Mar 2011 20:53:26 GMT
< Server: Noelios-Restlet-Engine/1.0..8
< Content-Type: text/plain; charset=ISO-8859-1
< Content-Length: 53
<
* Connection #0 to host localhost left intact
* Closing connection #0
maxConcurrentCleanUps shall be a positive integer: -1
```

The following *invalid* JSON example uses an unknown unit of measure (ZZiB). It returns a 400 response code and contains an error message as plain text:

```
curl -v -u admin:geoserver "http://localhost:8090/geoserver/gwc/rest/diskquota.json" -X PUT -d "{"gwc

< HTTP/1.1 400 Bad Request
< Date: Fri, 18 Mar 2011 20:56:23 GMT
< Server: Noelios-Restlet-Engine/1.0..8
< Content-Type: text/plain; charset=ISO-8859-1
< Content-Length: 601
<
No enum const class org.geowebcache.diskquota.storage.StorageUnit.ZZiB : No enum const class org.geow
---- Debugging information ----
message             : No enum const class org.geowebcache.diskquota.storage.StorageUnit.ZZiB
cause-exception     : java.lang.IllegalArgumentException
cause-message       : No enum const class org.geowebcache.diskquota.storage.StorageUnit.ZZiB
class               : org.geowebcache.diskquota.DiskQuotaConfig
required-type       : org.geowebcache.diskquota.storage.Quota
line number         : -1
* Connection #0 to host localhost left intact
* Closing connection #0
```

18.6 Troubleshooting

This section will discuss some common issues with the integrated GeoWebCache and their solutions.

18.6.1 Grid misalignment

Sometimes errors will occur when requesting data from GeoWebCache endpoints. The error displayed might say that the "resolution is not supported" or the "bounds do not align." This is due to the client making WMS requests that do not align with the grid of tiles that GeoWebCache has created, such as differing map bounds or layer bounds, or an unsupported resolution. If you are using OpenLayers as a client, looking at the source code of the included demos may provide more clues to matching up the grid.

An alternative workaround is to enable direct WMS integration with the GeoServer WMS. You can set this on the *Caching defaults* page.

18.6.2 Direct WMS integration

Direct integration allows WMS requests served through GeoServer to be cached as if they were received and processed by GeoWebCache. With Direct WMS Integration, a request may either be handled by the

GeoServer WMS or GeoWebCache WMS.

Sometimes requests that should go to GeoWebCache will instead be passed through to GeoServer, resulting in no tiles saved. That said, it is possible to determine why a request was not handled by GeoWebCache when intended. This is done by using the command-line utility cURL and inspecting the response headers.

First, obtain a sample request. This can easily be done by going to the Layer Preview for a given layer, setting the *Tiled* parameter to *Tiled*, then right-clicking on an area of the map and copy the full path to the image location. If done correctly, the result will be a GET request that looks something like this:

```
http://localhost:8090/geoserver/nurc/wms?LAYERS=nurc%3AArc_Sample&STYLES=&FORMAT=image%2Fjpeg&TILED=t
```

You can then paste this URL into a curl request:

```
curl -v "URL"
```

For example:

```
curl -v "http://localhost:8090/geoserver/nurc/wms?LAYERS=nurc%3AArc_Sample&STYLES=&FORMAT=image%2Fjpe
```

Note: To omit the raw image output to the terminal, pipe the output to your system's null. On Linux / OS X, append `> /dev/null` to these requests, and on Windows, append `> nul`.

If the request doesn't go through GeoWebCache's WMS, a reason will be given in a custom response header. Look for the following response headers:

- `geowebcache-cache-result`: Will say `HIT` if the GeoWebCache WMS processed the request, and `MISS` otherwise.
- `geowebcache-miss-reason`: If the above shows as `MISS`, this will generated a short description of why the request wasn't handled by the GeoWebCache WMS.

The following are some example requests made along with the responses. These responses have been truncated to show only the information relevant for troubleshooting.

Successful request

This request was successfully handled by the GeoWebCache WMS.

Request:

```
curl -v "http://localhost:8080/geoserver/topp/wms?TILED=true&LAYERS=states&FORMAT=image/png&REQUEST=G
```

Response:

```
< HTTP/1.1 200 OK
< Content-Type: image/png
< geowebcache-crs: EPSG:4326
...
< geowebcache-layer: topp:states
< geowebcache-gridset: EPSG:4326
< geowebcache-tile-index: [2, 6, 3]
...
< geowebcache-cache-result: HIT
< geowebcache-tile-bounds: -135.0,45.0,-112.5,67.5
...
```

Wrong height parameter

The following request is not handled by the GeoWebCache WMS because the image requested (256x257) does not conform to the expected 256x256 tile size.

Request:

```
curl -v "http://localhost:8080/geoserver/topp/wms?TILED=true&LAYERS=states&FORMAT=image/png&REQUEST=G
```

Response:

```
< HTTP/1.1 200 OK
< Content-Type: image/png
< geowebcache-miss-reason: request does not align to grid(s) 'EPSG:4326'
...
```

No tile layer associated

The following request is not handled by the GeoWebCache WMS because the layer requested has no tile layer configured.

Request:

```
curl -v "http://localhost:8080/geoserver/topp/wms?TILED=true&LAYERS=tasmania_roads&FORMAT=image/png&R
```

Response:

```
< HTTP/1.1 200 OK
< Content-Type: image/png
< geowebcache-miss-reason: not a tile layer
...
```

Missing parameter filter

The following request is not handled by the GeoWebCache WMS because the request contains a parameter filter (BGCOLOR) that is not configured for this layer.

Request:

```
curl -v "http://localhost:8080/geoserver/topp/wms?BGCOLOR=0xAAAAAA&TILED=true&LAYERS=states&FORMAT=im
```

Response:

```
< HTTP/1.1 200 OK
< Content-Type: image/png
< geowebcache-miss-reason: no parameter filter exists for BGCOLOR
...
```

CRS not defined

The following request is not handled by the GeoWebCache WMS because the request references a CRS (EPSG:26986) that does not match any of the tile layer gridsets:

Request:

```
curl -v "http://localhost:8080/geoserver/topp/wms?TILED=true&LAYERS=states&FORMAT=image/png&REQUEST=G
```

Response:

```
< HTTP/1.1 200 OK
< Content-Type: image/png
< geowebcache-miss-reason: no cache exists for requested CRS
...
```

CHAPTER 19

Google Earth

This section contains information on Google Earth support in GeoServer.

Google Earth is a 3-D virtual globe program. A free download from Google, it allows the user to virtually view, pan, and fly around Earth imagery. The imagery on Google Earth is obtained from a variety of sources, mainly from commercial satellite and aerial photography providers.

Google Earth recognizes a markup language called KML (Keyhole Markup Language) for data exchange. GeoServer integrates with Google Earth by supporting KML as a native output format. Any data configured to be served by GeoServer is thus able to take advantage of the full visualization capabilities of Google Earth.

19.1 Overview

19.1.1 Why use GeoServer with Google Earth?

GeoServer is useful when one wants to put a lot of data on to Google Earth. GeoServer automatically generates KML that can be easily and quickly served and visualized in Google Earth. GeoServer operates entirely through a Network Link, which allows it to selectively return information for the area being viewed. With GeoServer as a robust and powerful server and Google Earth providing rich visualizations, they are a perfect match for sharing your data.

19.1.2 Standards-based implementation

GeoServer supports Google Earth by providing KML as a Web Map Service (WMS) output format. This means that adding data published by GeoServer is as simple as constructing a standard WMS request and specifying "**application/vnd.google-earth.kml+xml**" as the `outputFormat`. Since generating KML is just a WMS request, it fully supports *Styling* via SLD.

See the next section (*Quickstart*) to view GeoServer and Google Earth in action.

19.2 Quickstart

Note: If you are using GeoServer locally, the GEOSERVER_URL is usually `http://localhost:8080/geoserver`

19.2.1 Viewing a layer

Once GeoServer is installed and running, open up a web browser and go to the web admin console (*Interface basics*). Navigate to the *Layer Preview* by clicking on the Layer Preview link at the bottom of the left sidebar. You will be presented with a list of the currently configured layers in your GeoServer instance. Find the row that says `topp:states`. To the right of the layer click on the link that says **KML**.

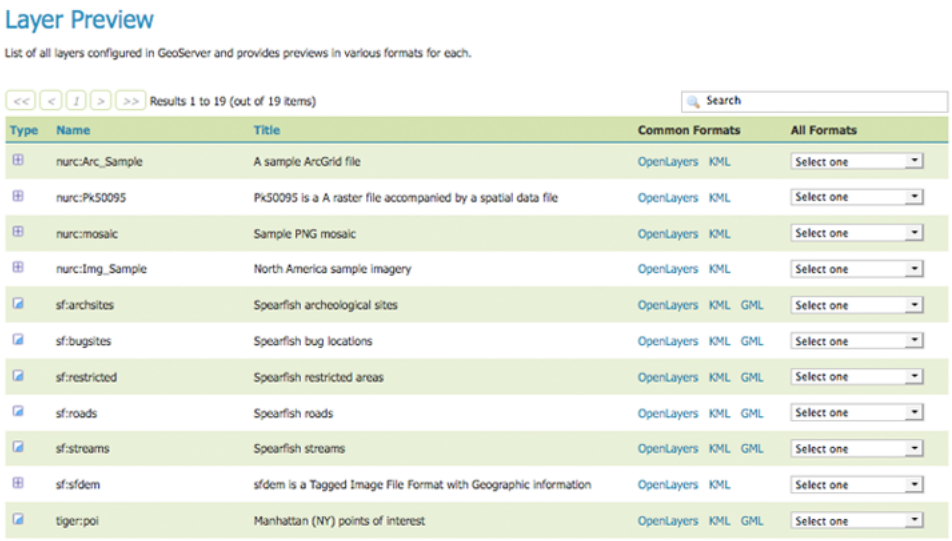


Figure 19.1: *The Map Preview page*

If Google Earth is correctly installed on your computer, you will see a dialog asking how to open the file. Select **Open with Google Earth**.

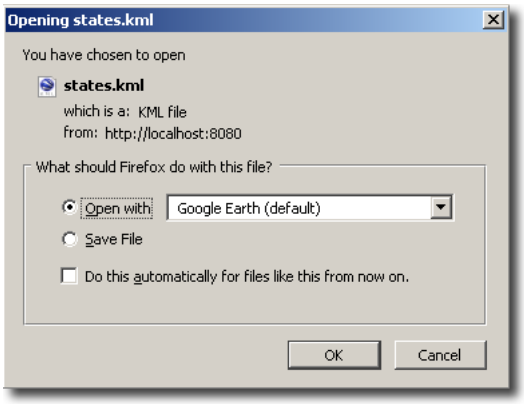


Figure 19.2: *Open with Google Earth*

When Google Earth is finished loading the result will be similar to below.

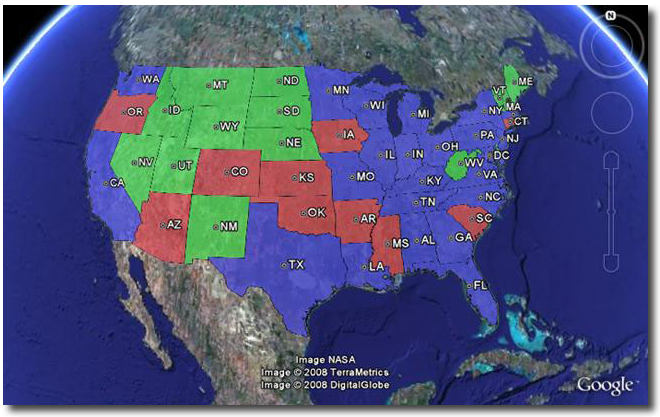


Figure 19.3: *The topp:states layer rendered in Google Earth*

19.2.2 Direct access to KML

All of the configured FeatureTypes are available to be output as KML (and thus loaded into Google Earth). The URL structure for KMLs is:

```
http://GEOSERVER_URL/wms/kml?layers=<layername>
```

For example, the topp:states layer URL is:

```
http://GEOSERVER_URL/wms/kml?layers=topp:states
```

19.2.3 Adding a Network Link

An alternative to serving KML directly into Google Earth is to use a Network Link. A Network Link allows for better integration into Google Earth. For example, using a Network Link enables the user to refresh the data within Google Earth, without having to retype a URL, or click on links in the GeoServer Map Preview again.

To add a Network Link, pull down the **Add** menu, and go to **Network Link**. The **New Network Link** dialog box will appear. Name your layer in the **Name** field. (This will show up in **My Places** on the main Google Earth screen.) Set **Link** to:

```
http://GEOSERVER_URL/wms/kml?layers=topp:states
```

(Don't forget to replace the GEOSERVER_URL.) Click **OK**. You can now save this layer in your **My Places**.

Check out the sections on *Tutorials* and the *KML Styling* for more information.

19.3 KML Styling

19.3.1 Introduction

Keyhole Markup Langauge (KML), when created and output by GeoServer, is styled using Styled Layer Descriptors (SLD). This is the same approach used to style standard WMS output formats, but is a bit different from how Google Earth is normally styled, behaving more like Cascading Style Sheets (CSS).

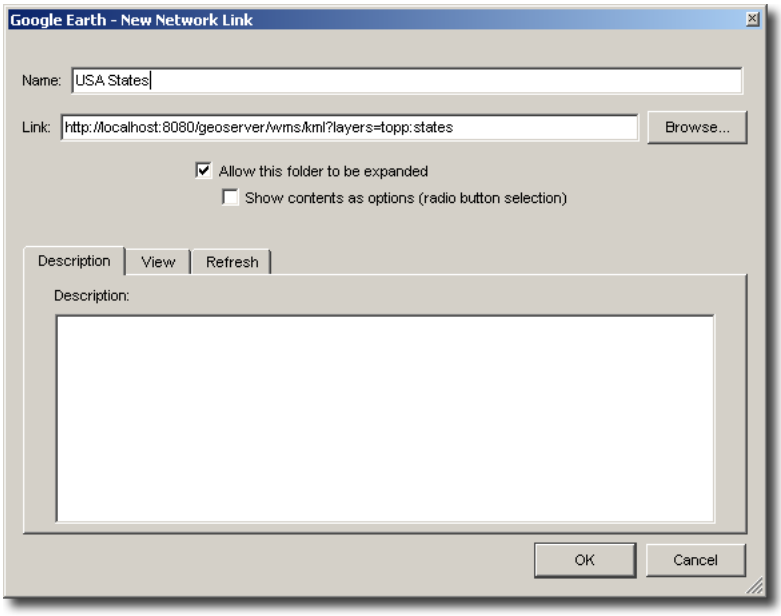


Figure 19.4: *Adding a network link*

The style of the map is specified in the SLD file as a series of rules, and then the data matching those rules is styled appropriately in the KML output. For those unfamiliar with SLD, a good place to start is the *Introduction to SLD*. The remainder of this guide contains information about how to construct SLD documents in order to impact the look of KML produced by GeoServer.

Contents

19.3.2 Basic SLD Creation Wizard

Basic SLD styling can be accomplished with the coming GeoExt Styler. It provides a GUI to create new styles. These styles will work seamlessly with KML output from GeoServer.

19.3.3 Creating SLD by hand

One can edit the SLD files directly instead of using the Styler GUI. For the most complete exploration of editing SLDs see the *Styling* section. The examples below show how some of the basic styles show up in

Google Earth.

19.3.4 SLD Structure

The following is a skeleton of a SLD document. It can be used as a base on which to expand upon to create more interesting and complicated styles.

```
<StyledLayerDescriptor version="1.0.0"
   xsi:schemaLocation="http://www.opengis.net/sld StyledLayerDescriptor.xsd"
   xmlns="http://www.opengis.net/sld"
   xmlns:ogc="http://www.opengis.net/ogc"
   xmlns:xlink="http://www.w3.org/1999/xlink"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <NamedLayer>
      <Name>Default Line</Name>
      <UserStyle>
         <Title>My Style</Title>
         <Abstract>A style</Abstract>
         <FeatureTypeStyle>
            <Rule>

                  <!-- symbolizers go here -->

            </Rule>
         </FeatureTypeStyle>
      </UserStyle>
   </NamedLayer>
</StyledLayerDescriptor>
```

Figure 3: Basic SLD structure

In order to test the code snippets in this document, create an SLD with the content as shown in Figure 3, and then add the specific code you wish to test in the space that says `<!-- symbolizers go here -->`. To view, edit, or add SLD files to GeoServer, navigate to **Config** -> **Data** -> **Styles**.

19.3.5 Points

In SLD, styles for points are specified via a PointSymbolizer. An empty PointSymbolizer element will result in a default KML style:

```
<PointSymbolizer>
</PointSymbolizer>
```

Figure 19.5: *Figure 4: Default point*

Three aspects of points that can be specified are *color*, *opacity*, and the *icon*.

Point Color

The color of a point is specified with a `CssParameter` element and a `fill` attribute. The color is specified as a six digit hexadecimal code.

```
<PointSymbolizer>
   <Graphic>
      <Mark>
         <Fill>
            <CssParameter name="fill">#ff0000</CssParameter>
         </Fill>
      </Mark>
   </Graphic>
</PointSymbolizer>
```

Figure 19.6: *Figure 5: Setting the point color (#ff0000 = 100% red)*

Point Opacity

The opacity of a point is specified with a CssParameter element and a `fill-opacity` attribute. The opacity is specified as a floating point number between **0** and **1**, with 0 being completely transparent, and 1 being completely opaque.

```
<PointSymbolizer>
   <Graphic>
      <Mark>
         <Fill>
            <CssParameter name="fill-opacity">0.5</CssParameter>
         </Fill>
      </Mark>
   </Graphic>
</PointSymbolizer>
```

Figure 19.7: *Figure 6: Setting the point opacity (0.5 = 50% opaque)*

Point Icon

An icon different from the default can be specified with the `ExternalGraphic` element:

```
<PointSymbolizer>
   <Graphic>
      <ExternalGraphic>
         <OnlineResource xlink:type="simple"
            xlink:href="http://maps.google.com/mapfiles/kml/pal3/icon55.png"/>
         <Format>image/png</Format>
      </ExternalGraphic>
   </Graphic>
</PointSymbolizer>
```

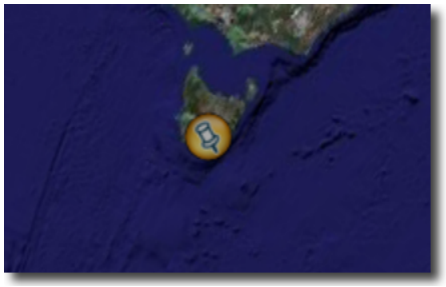


Figure 19.8: *Figure 7: A custom icon for points*

In Figure 7, the custom icon is specified as a remote URL. It is also possible to place the graphic in the GeoServer `styles` directory, and then specify the filename only:

```
<PointSymbolizer>
   <Graphic>
      <ExternalGraphic>
         <OnlineResource xlink:type="simple" xlink:href="icon55.png"/>
         <Format>image/png</Format>
      </ExternalGraphic>
   </Graphic>
</PointSymbolizer>
```

Figure 8: Specifying a local file for a graphic point

19.3.6 Lines

Styles for lines are specified via a `LineSymbolizer`. An empty `LineSymbolizer` element will result in a default KML style:

```
<LineSymbolizer>
</LineSymbolizer>
```

The aspects of the resulting line which can be specified via a `LineSymbolizer` are *color*, *width*, and *opacity*.

Line Color

The color of a line is specified with a `CssParameter` element and a `stroke` attribute. The color is specified as a six digit hexadecimal code.

Figure 19.9: *Figure 9: Default line*

```
<LineSymbolizer>
   <Stroke>
      <CssParameter name="stroke">#ff0000</CssParameter>
   </Stroke>
</LineSymbolizer>
```

Figure 19.10: *Figure 10: Line rendered with color #ff0000 (100% red)*

Line Width

The width of a line is specified with a `CssParameter` element and a `stroke-width` attribute. The width is specified as an integer (in pixels):

```
<LineSymbolizer>
   <Stroke>
      <CssParameter name="stroke-width">5</CssParameter>
```

```
   </Stroke>
</LineSymbolizer>
```

Figure 19.11: *Figure 11: Line rendered with a width of five (5) pixels*

Line Opacity

The opacity of a line is specified with a `CssParameter` element and a `fill-opacity` attribute. The opacity is specified as a floating point number between **0** and **1**, with 0 being completely transparent, and 1 being completely opaque.

```
<LineSymbolizer>
   <Stroke>
      <CssParameter name="stroke-opacity">0.5</CssParameter>
   </Stroke>
</LineSymbolizer>
```

Figure 19.12: *Figure 12: A line rendered with 50% opacity*

19.3.7 Polygons

Styles for polygons are specified via a `PolygonSymbolizer`. An empty `PolygonSymbolizer` element will result in a default KML style:

```
<PolygonSymbolizer>
</PolygonSymbolizer>
```

Polygons have more options for styling than points and lines, as polygons have both an inside ("fill") and an outline ("stroke"). The aspects of polygons that can be specified via a `PolygonSymbolizer` are *stroke color*, *stroke width*, *stroke opacity*, *fill color*, and *fill opacity*.

Polygon Stroke Color

The outline color of a polygon is specified with a `CssParameter` element and `stroke` attribute inside of a `Stroke` element. The color is specified as a 6 digit hexadecimal code:

```
<PolygonSymbolizer>
   <Stroke>
      <CssParameter name="stroke">#0000FF</CssParameter>
   </Stroke>
</PolygonSymbolizer>
```

Figure 19.13: *Figure 13: Outline of a polygon (#0000FF or 100% blue)*

Polygon Stroke Width

The outline width of a polygon is specified with a `CssParameter` element and `stroke-width` attribute inside of a `Stroke` element. The width is specified as an integer.

```
<PolygonSymbolizer>
   <Stroke>
      <CssParameter name="stroke-width">5</CssParameter>
   </Stroke>
</PolygonSymbolizer>
```

Figure 14: Polygon with stroke width of five (5) pixels

Polygon Stroke Opacity

The stroke opacity of a polygon is specified with a `CssParameter` element and `stroke` attribute inside of a `Stroke` element. The opacity is specified as a floating point number between **0** and **1**, with 0 being completely transparent, and 1 being completely opaque.

```
<PolygonSymbolizer>
   <Stroke>
      <CssParameter name="stroke-opacity">0.5</CssParameter>
   </Stroke>
</PolygonSymbolizer>
```

Figure 19.14: *Figure 15: Polygon stroke opacity of 0.5 (50% opaque)*

Polygon Fill Color

The fill color of a polygon is specified with a `CssParameter` element and `fill` attribute inside of a `Fill` element. The color is specified as a six digit hexadecimal code:

```
<PolygonSymbolizer>
   <Fill>
      <CssParameter name="fill">#0000FF</CssParameter>
   </Fill>
</PolygonSymbolizer>
```

Figure 19.15: *Figure 16: Polygon fill color of #0000FF (100% blue)*

Polygon Fill Opacity

The fill opacity of a polygon is specified with a `CssParameter` element and `fill-opacity` attribute inside of a `Fill` element. The opacity is specified as a floating point number between **0** and **1**, with 0 being completely transparent, and 1 being completely opaque.

```
<PolygonSymbolizer>
   <Fill>
      <CssParameter name="fill-opacity">0.5</CssParameter>
   </Fill>
</PolygonSymbolizer>
```

19.3.8 Text Labels

There are two ways to specify a label for a feature in Google Earth. The first is with Freemarker templates (LINK?), and the second is with a `TextSymbolizer`. Templates take precedence over symbolizers.

Freemarker Templates

Specifying labels via a Freemarker template involves creating a special text file called `title.ftl` and placing it into the `workspaces/<ws name>/<datastore name>/<feature type name>` directory (inside the GeoServer data directory) for the dataset to be labeled. For example, to create a template to label the `states` dataset by state name one would create the file here: `<data_dir>/workspaces/topp/states_shapefile/states/title.ftl`. The content of the file would be:

Figure 19.16: *Figure 17: Polygon fill opacity of 0.5 (50% opaque)*

```
${STATE_NAME.value}
```

Figure 19.17: *Figure 18: Using a Freemarker template to display the value of STATE_NAME*

For more information on Placemark Templates, please see our full tutorial (LINK FORTHCOMING).

TextSymbolizer

In SLD labels are specified with the Label element of a `TextSymbolizer`. (Note the `ogc:` prefix on the `PropertyName` element.)

```
<TextSymbolizer>
   <Label>
      <ogc:PropertyName>STATE_NAME</ogc:PropertyName>
   </Label>
</TextSymbolizer>
```

The aspects of the resulting label which can be specified via a `TextSymbolizer` are *color* and *opacity*.

Figure 19.18: *Figure 19: Using a TextSymbolizer to display the value of STATE_NAME*

TextSymbolizer Color

The color of a label is specified with a `CssParameter` element and `fill` attribute inside of a `Fill` element. The color is specified as a six digit hexadecimal code:

```
<TextSymbolizer>
   <Label>
      <ogc:PropertyName>STATE_NAME</ogc:PropertyName>
   </Label>
   <Fill>
      <CssParameter name="fill">#000000</CssParameter>
   </Fill>
</TextSymbolizer>
```

Figure 19.19: *Figure 20: TextSymbolizer with black text color (#000000)*

TextSymbolizer Opacity

The opacity of a label is specified with a `CssParameter` element and `fill-opacity` attribute inside of a `Fill` element. The opacity is specified as a floating point number between **0** and **1**, with 0 being completely transparent, and 1 being completely opaque.

```
<TextSymbolizer>
   <Label>
      <ogc:PropertyName>STATE_NAME</ogc:PropertyName>
   </Label>
   <Fill>
      <CssParameter name="fill-opacity">0.5</CssParameter>
   </Fill>
</TextSymbolizer>
```

Figure 19.20: *Figure 21: TextSymbolizer with opacity 0.5 (50% opaque)*

19.3.9 Descriptions

When working with KML, each feature is linked to a description, accessible when the feature is clicked on. By default, GeoServer creates a list of all the attributes and values for the particular feature.

It is possible to modify this default behavior. Much like with featureType titles, which are edited by creating a `title.ftl` template, a custom description can be used by creating template called `description.ftl` and placing it into the feature type directory (inside the GeoServer data directory) for the dataset. For instance, to create a template to provide a description for the states dataset, one would create the file: `<data_dir>/workspaces/topp/states_shapefile/states/description.ftl`. As an example, if the content of the description template is:

```
This is the state of ${STATE_NAME.value}.
```

The resultant description will look like this:

It is also possible to create one description template for all featureTypes in a given namespace. To do this, create a `description.ftl` file as above, and save it as `<data_dir>/templates/<workspace>/description.ftl`. Please note that if a description template is created for a specific featureType that also has an associated namespace description template, the featureType template (i.e. the most specific template) will take priority.

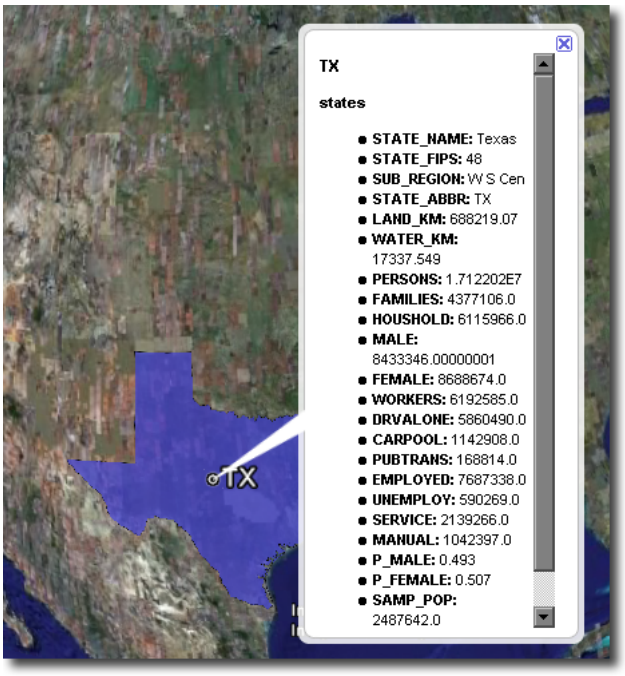


Figure 19.21: *Figure 22: Default description for a feature*

Figure 19.22: *Figure 23: A custom description*

One can also create more complex descriptions using a combination of HTML and the attributes of the data. A full tutorial on how to use templates to create descriptions is available in our page on KML Placemark Templates. (LINK?)

Basic SLD Creation Wizard SLD Structure Points Lines Polygons Text Labels Descriptions

19.4 Tutorials

19.4.1 KML Placemark Templates

Introduction

In KML a "Placemark" is used to mark a position on a map, often visualized with a yellow push pin. A placemark can have a "description" which allows one to attach information to it. Placemark descriptions are nothing more then an HTML snippet and can contain anything we want it to.

By default GeoServer produces placemark descriptions which are HTML tables describing all the attributes available for a particular feature in a dataset. In the following image we see the placemark description for the feature representing Idaho state:

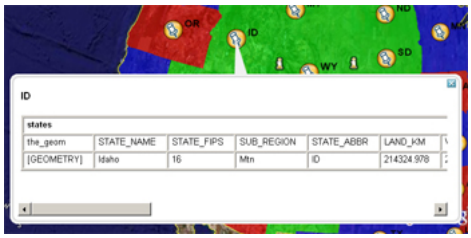


Figure 19.23: *The default placemark*

This is great, but what about if one wanted some other sort of information to be conveyed in the description. Or perhaps one does not want to show all the attributes of the dataset. The answer is Templates!!

A template is more or less a way to create some output.

Getting Started

First let us get set up. To complete the tutorial you will need the following:

- A GeoServer install
- A text editor

And thats it. For this tutorial we will assume that GeoServer is running the same configuration (data directory) that it does out of the box.

Hello World

Ok, time to get to creating our first template. We will start off an extremely simple template which, you guessed it, creates the placemark description "Hello World!". So lets go.

1. Using the text editor of your choice start a new file called `description.ftl`

2. Add the following content to the file:

   ```
   Hello World!
   ```

3. Save the file in the `workspaces/topp/states_shapefile/states` directory of your "data directory". The data directory is the location of all the GeoServer configuration files. It is normally pointed to by the environment variable `GEOSERVER_DATA_DIR`.

4. Start GeoServer is it is not already running.

And thats it. We can now test out our template by adding the following network link in google earth:

```
http://localhost:8080/geoserver/wms/kml?layers=states
```

And voila. Your first template

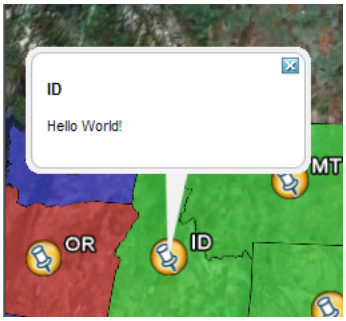


Figure 19.24: *Hello World template.*

Refreshing Templates: One nice aspect of templates is that they are read upon every request. So one can simply edit the template in place and have it picked up by Geoserver as soon as the file is saved. So when after editing and saving a template simply "Refresh" the network link in Google Earth to have the new content picked up.

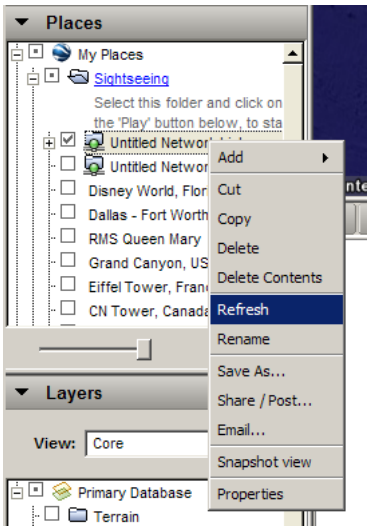


Figure 19.25: *Refresh Template*

As stated before template descriptions are nothing more than html. Play around with `description.ftl` and add some of your own html. Some examples you may want to try:

1. A simple link to the homepage of your organization:

```
Provided by the <a href="http://topp.openplans.org">The Open Planning Project</a>.
```

Homepage of Topp

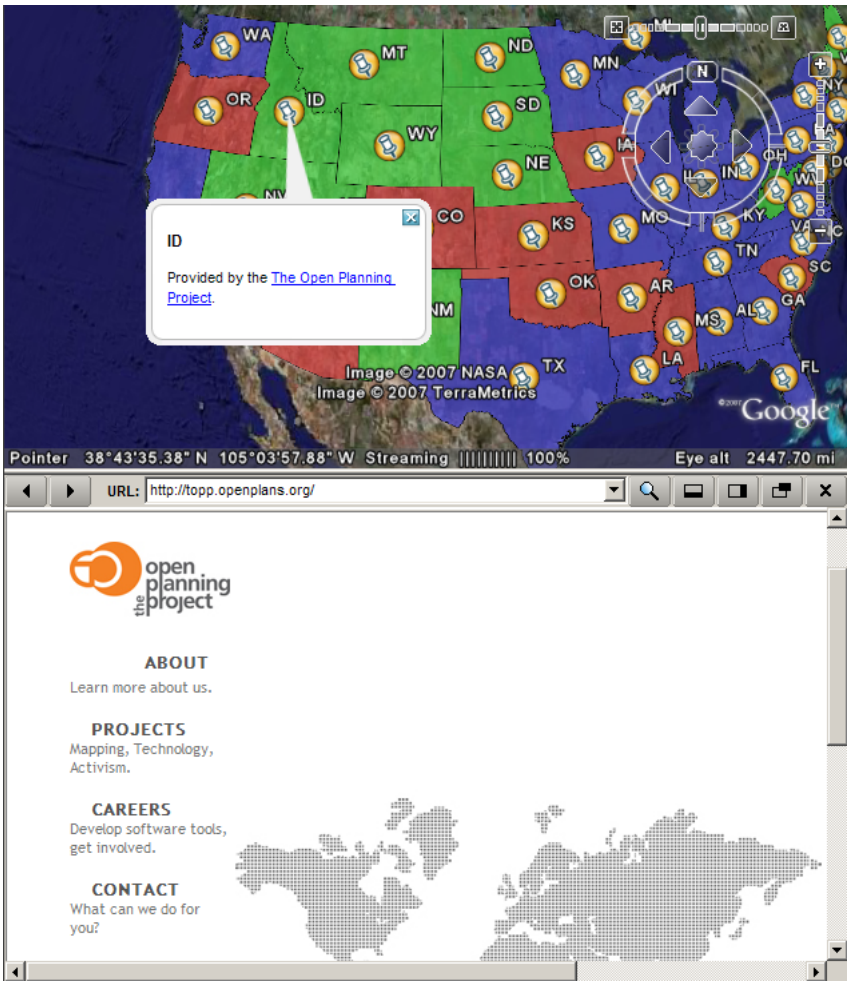


Figure 19.26: *Homepage of Topp*

2. The logo of your organization:

```
<img src="http://topp.openplans.org/images/logo.jpg"/>
```

Logo of Topp

The possibilities are endless. Now this is all great and everything but these examples are some what lacking in that the content is static. In the next section we will create more realistic template which actually access some the attributes of our data set.

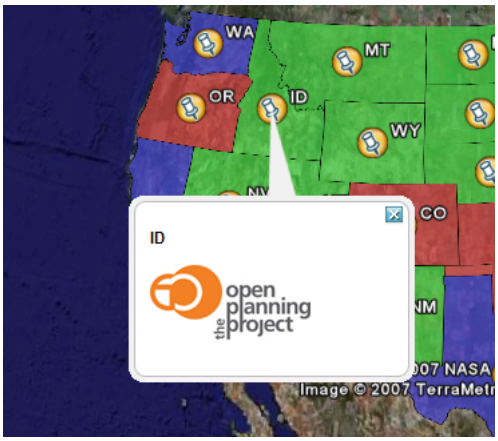


Figure 19.27: *Logo of Topp*

Data Content

The real power of templates is the ability to easily access content, in the case of features this content is the attributes of features.In a KML placemark description template, there are a number of "template variables" available.

- The variable "fid", which corresponds to the id of the feature
- The variable "typeName", which corresponds to the name of the type of the feature
- A sequence of variables corresponding to feature attributes, each named the same name as the attribute

So with this knowledge in hand let us come up with some more examples:

Simple fid/typename access:

```
This is feature ${fid} of type ${typeName}.
```

This is a feature of 3.1 of type states.

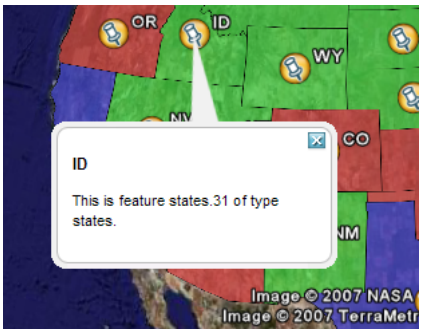


Figure 19.28: *FID*

Access to the values of two attributes named `STATE_NAME`, and `PERSONS`:

```
This is ${STATE_NAME.value} state which has a population of ${PERSONS.value}.
```

ID This is Idaho state which has a population of 1.006.749.

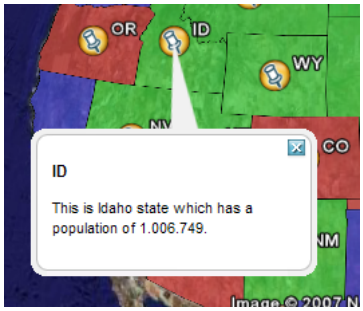


Figure 19.29: *Attributes*

Attribute Variables

A feature attribute a "complex object" which is made up of three parts:

1. **A value**, given as a default string representation of the actual attribute value feasible to be used directly
2. **A rawValue**, being the actual value of the attribute, to allow for more specialized customization (for example, `${attribute.value?string("Enabled", "Disabled")}` for custom representations of boolean attributes, etc).
3. **A type**, each of which is accessible via `${<attribute_name>.name}`, `${<attribute_name>.value}`, `${<attribute_name>.rawValue}`, `${<attribute_name>.type}` respectively. The other variables: fid, and typeName and are "simple objects" which are available directly.

WMS Demo Example

We will base our final example off the "WMS Example" demo which ships with GeoServer. To check out the demo visit http://localhost:8080/geoserver/popup_map/index.html in your web browser.

You will notice that hovering the mouse over one of the points on the map displays an image specific to that point. Let us replicate this with a KML placemark description.

1. In the `featureTypes/DS_poi_poi` directory of the geoserver data directory create the following template:

   ```
   <img src="http://localhost:8080/geoserver/popup_map/${THUMBNAIL.value}"/>
   ```

2. Add the following network link in Google Earth:

   ```
   http://localhost:8080/geoserver/wms/kml?layers=tiger:poi
   ```

Poi.4

19.4.2 Heights Templates

Introduction

Height Templates in KML allow you to use an attribute of your data as the 'height' of features in Google Earth.

Note: This tutorial assumes that GeoServer is running on http://localhost:8080.

Figure 19.30: *WMS Example*

Getting Started

For the purposes of this tutorial, you just need to have GeoServer with the release configuration, and Google Earth installed. Google Earth is available for free from <http://earth.google.com/ <http://earth.google.com/>'_.

Step One

By default GeoServer renders all features with 0 height, so they appear to lay flat on the world's surface in Google Earth.

To view the `topp:states` layer (packaged with all releases of GeoServer) in Google Earth, the easiest way is to use a network link. In Google Earth, under *Places*, right-click on *Temporary Places*, and go to *Add* → *Network Link*. In the dialog box, fill in `topp:states` as the *Name*, and the following URL as the *Link*:

```
http://localhost:8080/geoserver/wms/reflect?layers=topp:states&format=application/vnd.google-earth.kr
```

Step Two

An interesting value to use for the height would be the population of each state (so that more populated states appear taller on the map). We can do this by creating a file called `height.ftl` in the GeoServer data directory under `workspaces/topp/states_shapefile/states`. To set the population value, we enter the following text inside this new file:

```
${PERSONS.value}
```

This uses the value of the `PERSONS` attribute as the height for each feature. To admire our handiwork, we can refresh our view by right-clicking on our temporary place (called `topp:states`) and selecting *Refresh*:

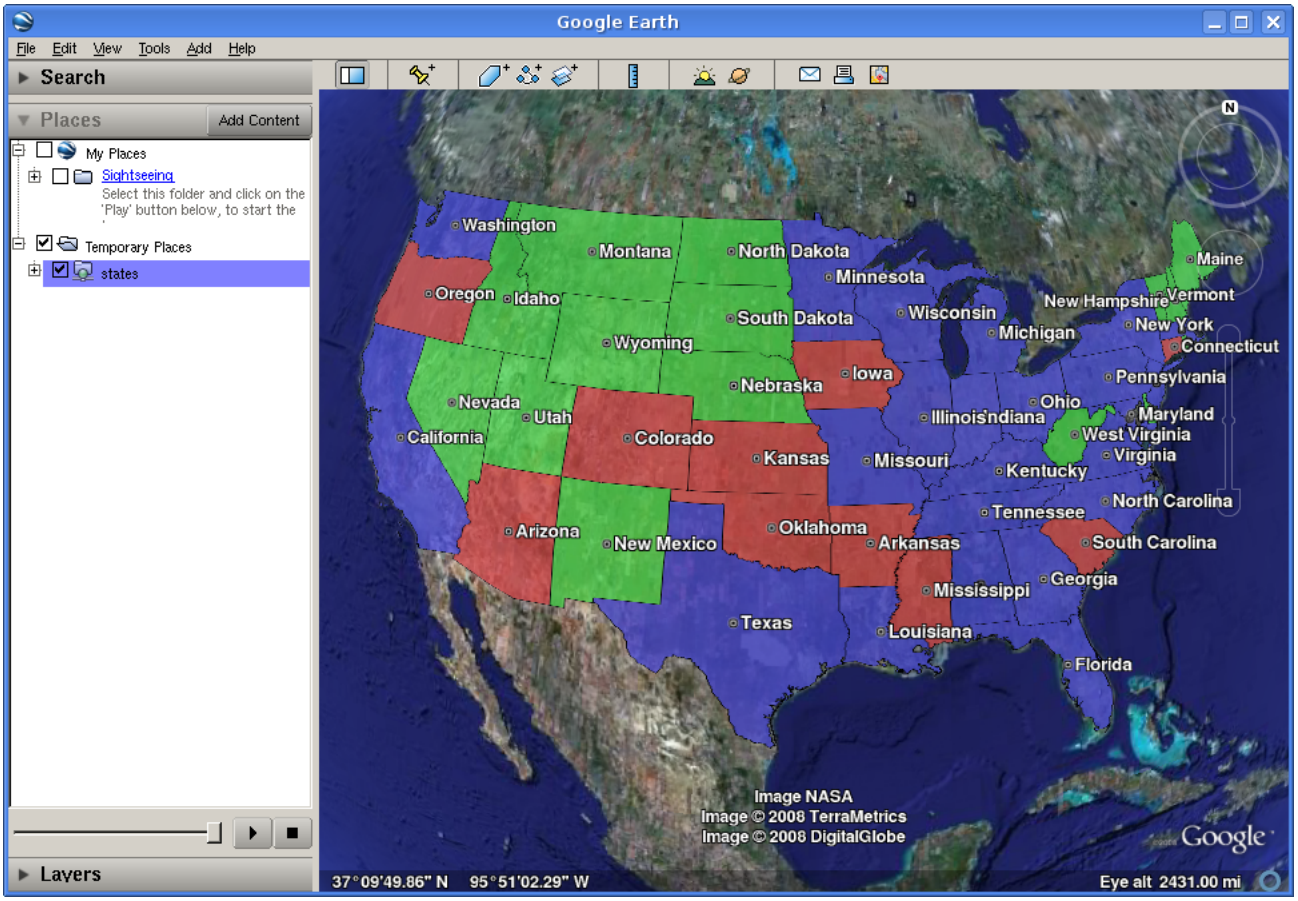


Figure 19.31: *topp:states in Google Earth*

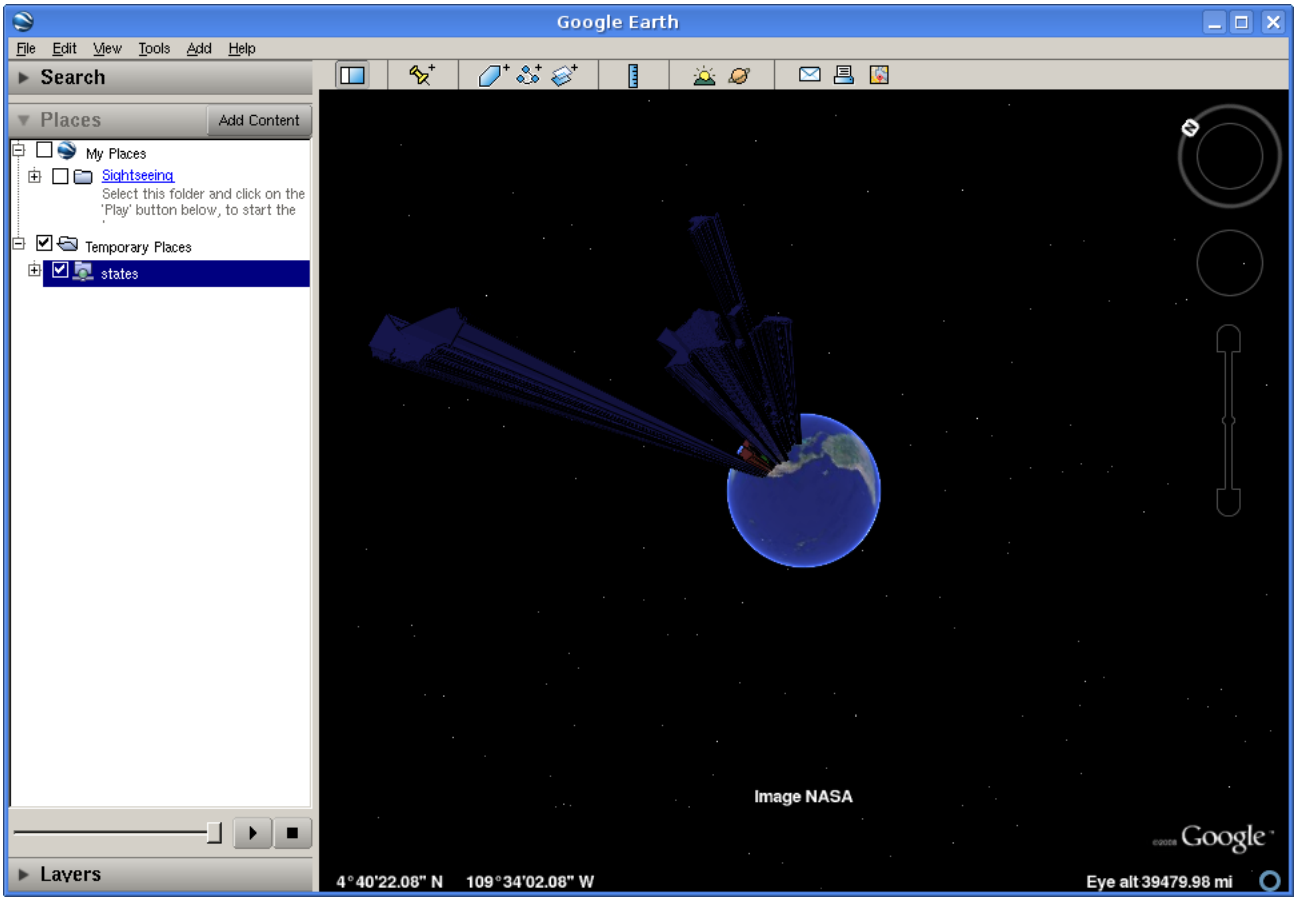


Figure 19.32: *Height by Population*

Step Three

Looking at our population map, we see that California dwarfs the rest of the nation, and in general all of the states are too tall for us to see the heights from a convenient angle. In order to scale things down to a more manageable size, we can divide all height values by 100. Just change the template we wrote earlier to read:

```
${PERSONS.value / 100}
```

Refreshing our view once again, we see that our height field has disappeared. Looking at the GeoServer log (in the data directory under logs/geoserver.log) we see something like:

```
Caused by: freemarker.core.NonNumericalException: Error on line 1, column 3 in height.ftl
Expression PERSONS.value is not numerical
```

However, we know that the `PERSONS` field is numeric, even if it is declared in the shapefile as a string value. To force a conversion, we can append `?number`, like so:

```
${PERSONS.value?number / 100}
```

One final *Refresh* brings us to a nicely sized map of the US:

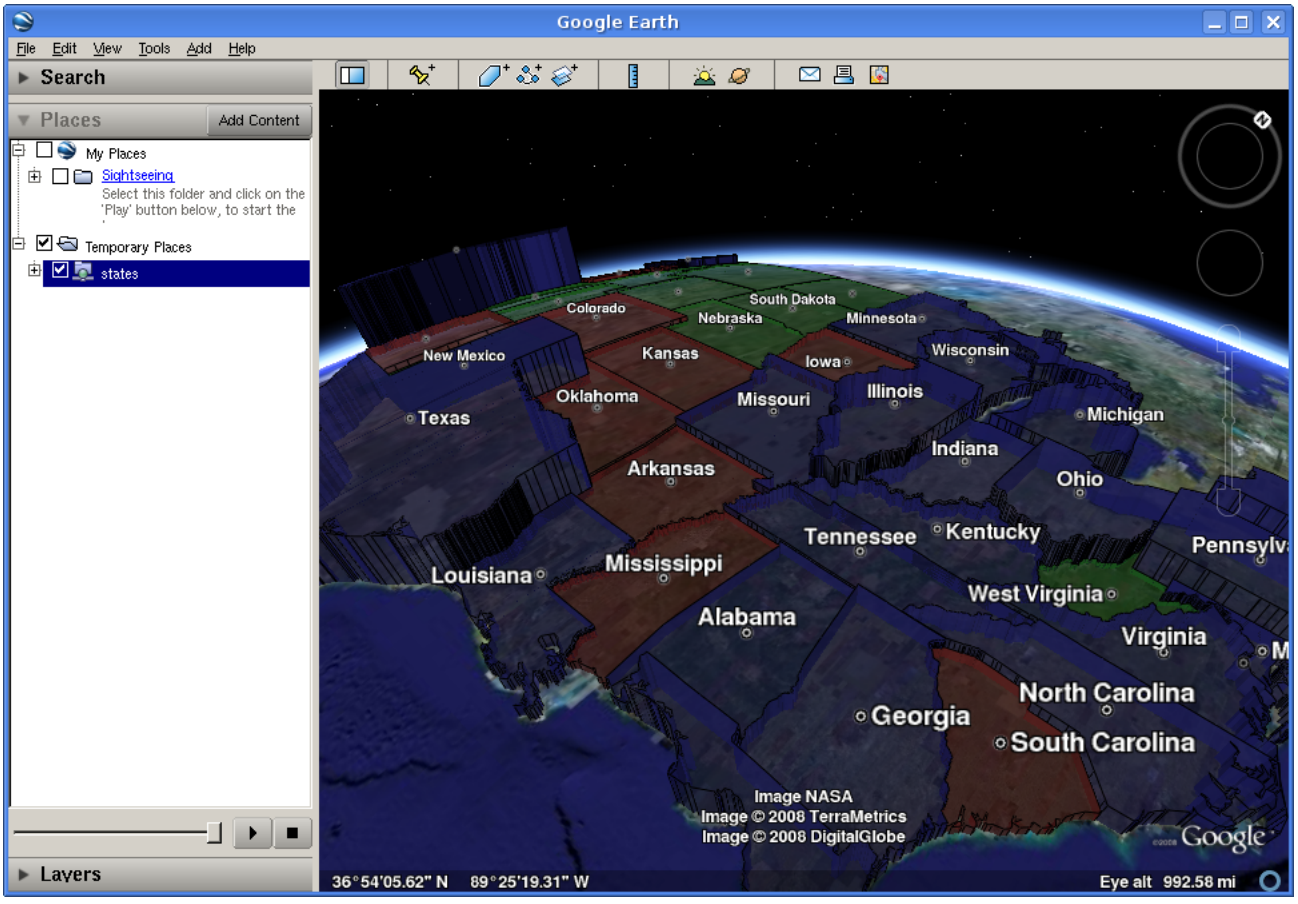


Figure 19.33: *Scaled Height*

Step Four

There are still a couple of tweaks we can make. The default is to create a 'solid' look for features with height, but Google Earth can also create floating polygons that are disconnected from the ground. To turn off the 'connect to ground' functionality, add a format option called 'extrude' whose value is 'false'. That is, change the *Link* in the Network Link to be:

```
http://localhost:8080/geoserver/wms/reflect?layers=topp:states&format=application/vnd.google-earth.kr
```

We also have a few options for how Google Earth interprets the height field. By default, the height is interpreted as relative to the ground, but we can also set the heights relative to sea level, or to be ignored (useful for reverting to the 'flat' look without erasing your template). This is controlled with a format option named altitudeMode, whose values are summarized below.

altitudeMode	**Purpose**
altitudeMode	Interpret height as relative to ground level
absolute	Interpret height as relative to sea level
clampToGround	Ignore height entirely

19.4.3 Time

Warning: The screenshots on this tutorial have not yet been updated for the 2.0.x user interface. But most all the rest of the information should be valid, and the user interface is roughly the same, but a bit more easy to use.

Getting Started

For this tutorial we will using a Shapefile which contains information about the number of Internet users in the countries of Western Europe for a rang of years.

1. Download and unzip `inet_weu.zip`
2. Configure GeoServer to serve the Shapefile `inet_weu.zip`. (A tutorial is available *Publishing a Shapefile*.)
3. Add the SLD "`inet_weu.sld` to GeoServer. (A tutorial is available for *Styling a Map*)
4. Set the style of the feature type added in step 2 to the style added in step 3

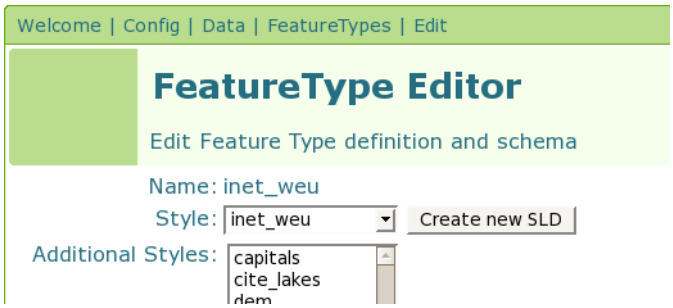


Figure 19.34: *Style*

Checking the Setup

If all is configured properly you should be able to navigate to http://localhost:8080/geoserver/wms/kml?layers=topp:inet_weu&format=openlayers&bbox=-33.780,26.266,21.005,56.427 and see the following map:

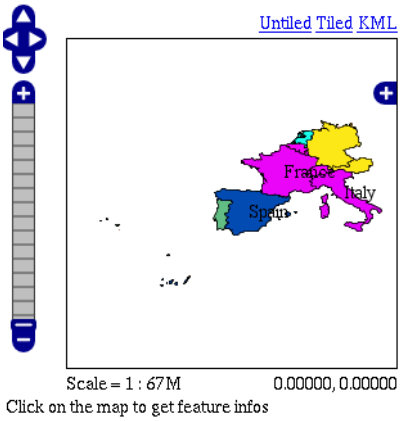


Figure 19.35: *Setup*

Creating the Template

Next we will create a template which allows us to specify the temporal aspects of the dataset. The schema of our dataset looks like:

INET_P100n	Number of internet users per 100 people
NAME	Name of country
RPT_YEAR	Year
Geometry	Polygon representing the country

The temporal attribute is `RPT_YEAR` and is the one that matters to us. Ok, time to create the template.

1. In your text editor of choice, create a new text file called `time.ftl`.
2. Add the following text:

```
${RPT_YEAR.value?date('yyyy')}
```

3. Save the file to the `<GEOSERVER_DATA_DIR>/workspaces/topp/inet_weu_shapefile/inet_weu` directory. Where `<GEOSERVER_DATA_DIR>` is the location of the "data directory" of your GeoServer installation. Usually pointed to via the `GEOSERVER_DATA_DIR` environment variable.

See the ref:*references* section for more information about specifying a date format.

Trying it Out

Ok time to try it out.

1. Navigate to http://localhost:8080/geoserver/wms/kml?layers=inet_weu&legend=true. This should cause Google Earth to open.
2. In Google Earth, adjust the time bar so that it captures a time interval that is approximately 1 year wide

Figure 19.36: *Google Earth*

Figure 19.37: *Google Earth Time Bar*

3. Slide the time bar forward in time and notice how the polygon colors change

References

Specifying a Date Format

When setting up a time template for your own dataset the most important issue is the format of your temporal data. It may or may not be in a format in which GeoServer can read directly. You can check if the date/time format can be used directly by GeoServer by using the following time template. This is an example time template file (time.ftl) file without explicit formatting.

```
${DATETIME_ATTRIBUTE_NAME.value}
```

While GeoServer will try its best to parse the data there are cases in which your data is in a format which it cannot parse. When this occurs it is necessary to explicitly specify the format. Luckily Freemarker provides us with functionality to do just this.

Consider the date time `12:30 on January 01, 2007` specified in the following format: `01?01%2007&12$30!00`. When creating the template we need to explicitly tell Freemarker the format the date time is in with the datetime function. This is an example time template file (time.ftl) file with explicit formatting:

```
${DATETIME_ATTRIBUTE_NAME.value?datetime("M?d%y&H:m:s")}
```

The process is similar for dates (no time). The date `01?01%2007` would be specified in a template with explicit formatting:

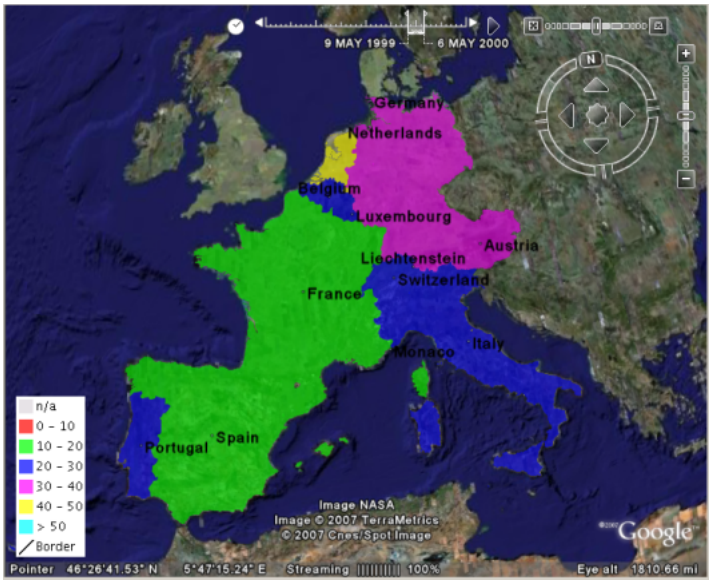


Figure 19.38: *Sliding the Time Bar*

```
${DATETIME_ATTRIBUTE_NAME.value?date("M?d%y")}
```

So when must you specify the date format in this manner? The following table illustrates the *date* formats that GeoServer can understand. Note that the '-' character can be one of any of the following characters: '/' (forward slash), ' ' (space), '.' (period), ',' (comma)

Date Format	Example
yyyy-MM-dd	2007-06-20
yyyy-MMM-dd	2007-Jun-20
MM-dd-yyyy	06-20-2007
MMM-dd-yyyy	Jun-20-2007
dd-MM-yyyy	20-06-2007
dd-MMM-yyyy	20-Jun-2007

The set of *date time* formats which GeoServer can be understand is formed by appending the timestamp formats `hh:mm` and `hh:mm:ss` to the entries in the above table:

DateTime Format	Example
yyyy-MM-dd hh:mm	2007-06-20 12:30
yyyy-MMM-dd hh:mm	2007-Jun-20 12:30
yyyy-MM-dd hh:mm:ss	2007-06-20 12:30:00
yyyy-MMM-dd hh:mm:ss	2007-Jun-20 12:30:00

Warning: Setting the Timezone
Be aware that the KML output for *date time* formats will reflect the timezone of the java virtual machine, which can be set using the user.timezone parameter in the startup script. For example, the following command starts GeoServer using the Coordinated Universal Time (UTC) timezone.

```
exec "$_RUNJAVA" -DGEOSERVER_DATA_DIR="$GEOSERVER_DATA_DIR"
    -Djava.awt.headless=true -DSTOP.PORT=8079    -Duser.timezone=UTC
    -DSTOP.KEY=geoserver -jar start.jar
```

If the timezone is not set, it will default to the timezone of the operating system.

Specifying a Date Range

In the above example a single time stamp is output for the dataset. GeoServer also supports specifying date ranges via a template. The syntax for ranges is:

Where begin is the first date in the range, end is the last date in the range, and `||` is the delimiter between the two. As an example:

Would the date range starting at `January 1, 2007` and ending `June 1, 2007`. Date ranges can also be open ended:

The first date specifies a date range where the beginning is open-ended. The second specifies a date range where the end is open-ended.

19.4.4 Super-Overlays and GeoWebCache

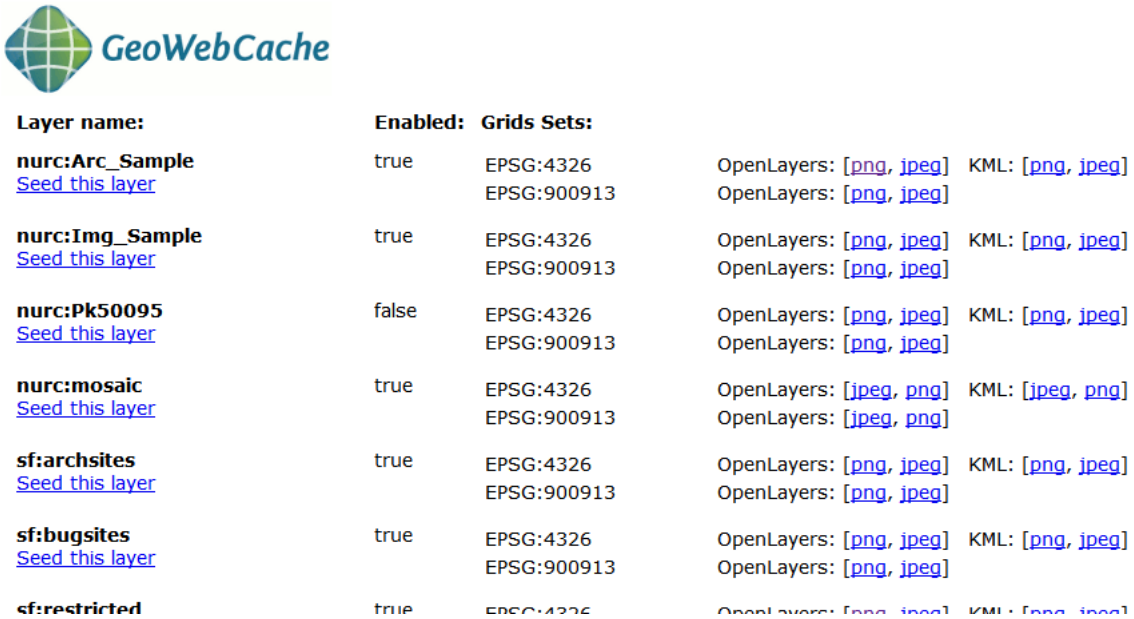
Overview

This tutorial explains how to use GeoWebCache (GWC) to enhance the performance of super-overlays in Google Earth. For more information please see the page on *KML Super-Overlays*

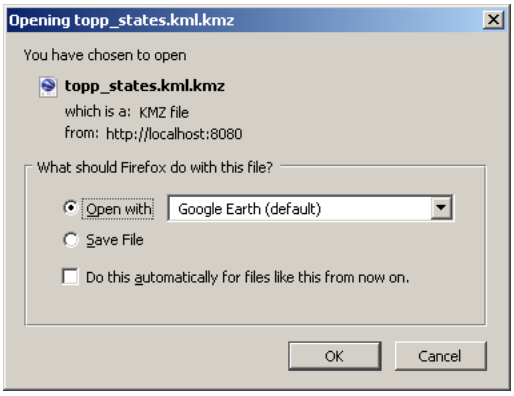
Conveniently GeoWebCache can generate super-overlays automatically. With the standalone GeoWebCache it takes minimal amount of configuration. Please see the GeoWebCache documentation for more information on the standalone version of GeoWebCache.

We are going to use the plug in version of GeoWebCache where there is no configuration need. For this tutorial we are also using the topp:states layer. Using the GeoWebCache plug in with super-overlays

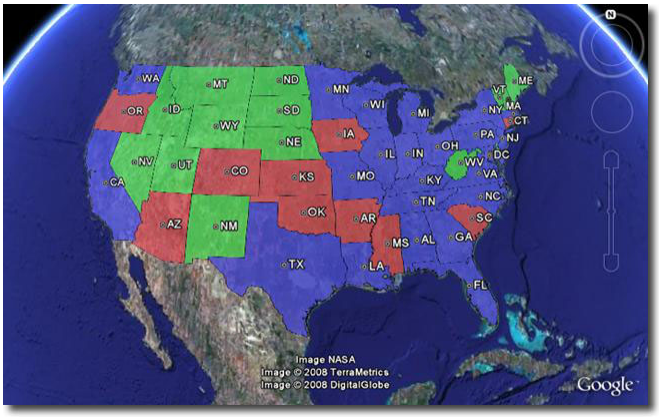
To access GWC from GeoServer go to http://localhost:8080/geoserver/gwc/demo/. This should return a layer list of similar to below.



To use a super-overlay in GeoWebCache select the KML (vector) option display for each layer. Lets select topp:states.The url would be http://localhost:8080/geoserver/gwc/service/kml/topp:states.kml.kmz After doing so you will be presented with a open option dialog, choose Google Earth.



When Google Earth finishes loading you should be viewing a the topp:states layers.



19.5 Features

This section delves into greater detail about the various functionality and options possible with KML output and Google Earth.

19.5.1 KML Reflector

Standard WMS requests can be quite long and cumbersome. The following is an example of a request for KML output from GeoServer:

```
http://localhost:8080/geoserver/ows?service=WMS&request=GetMap&version=1.1.1&format=application/vnd.g
```

GeoServer includes an alternate way of requesting KML, and that is to use the **KML reflector**. The KML reflector is a simpler URL-encoded request that uses sensible defaults for many of the parameters in a standard WMS request. Using the KML reflector one can shorten the above request to:

```
http://localhost:8080/geoserver/wms/kml?layers=topp:states
```

Using the KML reflector

The only mandatory parameter is the `layers` parameter. The syntax is as follows:

```
http://GEOSERVER_URL/wms/kml?layers=<layer>
```

where `GEOSERVER_URL` is the URL of your GeoServer instance, and `<layer>` is the name of the feature-type to be served.

The following table lists the default assumptions:

Key	Value
request	GetMap
service	wms
version	1.1.1
srs	EPSG:4326
format	applcation/vnd.google-earth.kmz+xml
width	2048
height	2048
bbox	<layer bounds>
kmattr	true
kmplacemark	false
kmscore	40
styles	[default style for the featuretype]

Any of these defaults can be changed when specifying the request. For instance, to specify a particular style, one can append `styles=population` to the request:

```
http://localhost:8080/geoserver/wms/kml?layers=topp:states&styles=population
```

To specify a different bounding box, append the parameter to the request:

```
http://localhost:8080/geoserver/wms/kml?layers=topp:states&bbox=-124.73,24.96,-66.97,49.37
```

Reflector modes

The KML reflector can operate in one of three modes: **refresh**, **superoverlay**, and **download**.

The mode is set by appending the following parameter to the URL:

```
mode=<mode>
```

where `<mode>` is one of the three reflector modes. The details for each mode are as follows:

Mode	Description
refresh	(*default for all versions except 1.7.1 through 1.7.5*) Returns dynamic KML that can be refreshed/updated by the Google Earth client. Data is refreshed and new data/imagery is downloaded when zooming/panning stops. This mode can return either vector or raster (placemark or overlay) The decision to return either vector or raster data is determined by the value of `kmscore`. Please see the section on *KML Scoring* for more information.
superoverlay	(*default for versions 1.7.1 through 1.7.5*) Returns KML as a super-overlay. A super-overlay is a form of KML in which data is broken up into regions. Please see the section on *KML Super-Overlays* for more information.
download	Returns KML which contains the entire data set. In the case of a vector layer, this will include a series of KML placemarks. With raster layers, this will include a single KML ground overlay. This is the only mode that doesn't dynamically request new data from the server, and thus is self-contained KML.

More about the "superoverlay" mode

When requesting KML using the `superoverlay` mode, there are four additional submodes available regarding how and when data is requested. These options are set by appending the following parameter to the KML reflector request:

```
superoverlay_mode=<submode>
```

where `<submode>` is one of the following options:

Sub-mode	Description
`auto`	(*default*) Always returns vector features if the original data is in vector form, and returns raster imagery if the original data is in raster form. This can sometimes be less than optimal if the geometry of the features are very complicated, which can slow down Google Earth.
`raster`	Always returns raster imagery, regardless of the original data. This is almost always faster, but all vector information is lost in this view.
`overview`	Displays either vector or raster data depending on the view. At higher zoom levels, raster imagery will be displayed, and at lower zoom levels, vector features will be displayed. The determination for when to switch between vector and raster is made by the regionation parameters set on the server. See the section on *KML Regionation* for more information.
`hybrid`	Displays both raster and vector data at all times.

19.5.2 Toggling Placemarks

Vector Placemarks

When GeoServer generates KML for a vector dataset, it attaches information from the data to each feature that is created. When clicking on a vector feature, a pop-up window is displayed. This is called a **placemark**. By default this is a simple list which displays attribute data, although this information can be customized using Freemarker templates.

If you would like this information not to be shown when a feature is clicked (either for security reasons, or simply to have a cleaner user interface), it is possible to disable this functionality. To do so, use the `kmattr` parameter in a KML request to turn off attribution.

The syntax for `kmattr` is as follows:

```
format_options=kmattr:[true|false]
```

Note that `kmattr` is a "format option", so the syntax is slightly different from the usual key-value pair. For example:

```
http://localhost:8080/geoserver/wms/kml?layers=topp:states&format_options=kmattr:false
```

Raster Placemarks

Unlike vector features, where the placemark is enabled by default, placemarks are disabled by default with raster images of features. To enable this feature, you can use the `kmplacemark` format option in your KML request. The syntax is similar to the `kmattr` format option specified above:

```
format_options=kmplacemark:[true|false]
```

For example, using the KML reflector, the syntax would be:

```
http://localhost:8080/geoserver/wms/kml?layers=topp:states&format_options=kmplacemark:true
```

19.5.3 Customizing Placemarks

KML output can leverage some powerful visualization abilities in Google Earth. **Titles** can be displayed on top of the features. **Descriptions** (custom HTML shown when clicking on a feature) can be added to customize the views of the attribute data. In addition, using Google Earth's time slider, **time**-based animations can be created. Finally, **height** of features can be set, as opposed to the default ground overlay. All of these can be accomplished by creating Freemarker templates. Freemarker templates are text files (with limited HTML code), saved in the *GeoServer Data Directory*, that utilize variables that link to specific attributes in the data.

Titles

Specifying labels via a template involves creating a special text file called `title.ftl` and placing it into the featuretypes directory inside the *GeoServer Data Directory* for the dataset to be labeled. For instance, to create a template to label the `states` layer by state name, one would create the file: `<data_dir>/workspaces/topp/states_shapefile/states/title.ftl`. The content of the file would be:

```
${STATE_NAME.value}
```

Descriptions

When working with KML, each feature is linked to a description, accessible when the feature is clicked on. By default, GeoServer creates a list of all the attributes and values for the particular feature.

It is possible to modify this default behavior. Much like with featuretype titles, which are edited by creating a `title.ftl` template, specifying descriptions via a template involves creating a special text file called `description.ftl` and placing it into the featuretypes directory inside the *GeoServer Data Directory* for the dataset to be labeled. For instance, a sample description template would be saved here: `<data_dir>/workspaces/topp/states_shapefile/states/description.ftl`. The content of the file could be:

```
This is the state of ${STATE_NAME.value}.
```

The resulting description will look like this:

> **Warning:** Add SS: A custom description

It is also possible to create one description template for all layers in a given namespace. To do this, create a `description.ftl` file as above, and save it here:

```
<data_dir>/templates/<namespace>/description.ftl.
```

Please note that if a description template is created for a specific layer that also has an associated namespace description template, the layer template (i.e. the most specific template) will take priority.

19.5.4 KML Height and Time

Height

GeoServer by default creates two dimensional overlays in Google Earth. However, GeoServer can output features with height information (also called "KML extrudes") if desired. This can have the effect of having features "float" above the ground, or create bar graph style structures in the shape of the features. The height of features can be linked to an attribute of the data.

Setting the height of features is determined by using a KML Freemarker template. Create a file called `height.ftl`, and save it in the same directory as the featuretype in your *GeoServer Data Directory*. For example, to create a height template for the `states` layer, the file should be saved in `<data_dir>/workspaces/topp/states_shapefile/states/height.ftl`.

To set the height based on an attribute, the syntax is:

```
${ATTRIBUTE.value}
```

Replace the word `ATTRIBUTE` with the name of the height attribute in your data set. For a complete tutorial on working with the height templates see *Heights Templates*.

Time

Google Earth also contains a "time slider", which can allow animations of data, and show changes over time. As with height, time can be linked to an attribute of the data, as long as the data set has a date/time attribute. Linking this date/time attribute to the time slider in Google Earth is accomplished by creating a Freemarker template. Create a file called `time.ftl`, and save it in the same directory that contains your data's `info.xml`.

To set the time based on an attribute the syntax is:

```
${DATETIME_ATTRIBUTE.value}
```

Replace the word `DATETIME_ATTRIBUTE` with the name of the date/time attribute. When creating KML, GeoServer will automatically link the data to the time element in Google Earth. If set successfully, the time slider will automatically appear.

For a full tutorial on using GeoServer with Google Earth's time slider see *Time*

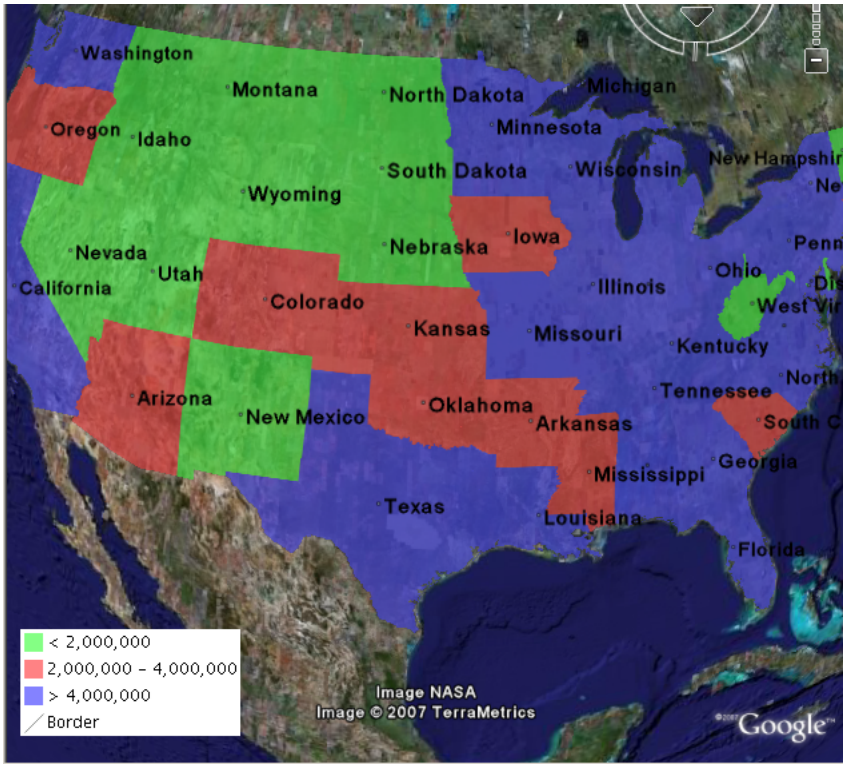
19.5.5 KML Legends

WMS includes a `GetLegendGraphic` operation which allows a WMS client to obtain a legend graphic from the server for a particular layer. Combining the legend with KML overlays allows the legend to be viewed inside Google Earth.

To get GeoServer to include a legend with the KML output, append `legend=true` to the KML reflector request. For example:

```
http://localhost:8080/geoserver/wms/kml?layers=topp:states&legend=true
```

The resulting Google Earth output looks like this:



19.5.6 Filters

Though not specific to Google Earth, GeoServer has the ability to filter data returned from the *Web Map Service*. The KML Reflector will pass through any WMS `filter` or `cql_filter` parameter to GeoServer to constrain the response.

Note: Filters are basically a translation of a SQL "WHERE" statement into web form. Though limited to a single table, this allows users to do logical filters like "AND" and "OR" to make very complex queries, leveraging numerical and string comparisons, geometric operations ("bbox", "touches", "intersects", "disjoint"), "LIKE" statements, nulls, and more.

Filter

There simplest filter is very easy to include. It is called the `featureid` filter, and it lets you filter to a single feature by its ID. The syntax is:

```
featureid=<feature>
```

where <feature> is the feature and its ID. An example would be:

```
http://localhost:8080/geoserver/wms/kml?layers=topp:states&featureid=states.5
```

This request will output only the state of Maryland. The feature IDs of your data are most easily found by doing WFS or KML requests and examining the resulting output.

CQL Filter

Using filters in a URL can be very unwieldy, as one needs to include URL-encoded XML:

```
http:/localhost:8080/geoserver/wms/kml?layers=topp:states&FILTER=%3CFilter%3E%3CPropertyIsBetween%3E%
```

Instead, one can use Common Query Language (CQL), which allows one to specify the same statement more succinctly:

```
http://localhost:8080/geoserver/wms/kml?layers=topp:states&CQL_FILTER=LAND_KM+BETWEEN+100000+AND+150
```

This query will return all the states in the US with areas between 100,000 and 150,000 km^2.

19.5.7 KML Super-Overlays

Super-overlays are a form of KML in which data is broken up into regions. This allows Google Earth to refresh/request only particular regions of the map when the view area changes. Super-overlays are used to efficiently publish large sets of data. (Please see Google's page on super-overlays for more information.)

GeoServer supports two types of super-overlays: **raster** and **vector**. With raster super-overlays, GeoServer intelligently produces imagery appropriate to the current zoom level and dynamically outputs new imagery when the zoom level changes. With vector super-overlays, feature data is requested for only the visible features and new features are dynamically loaded as necessary. Raster super-overlays require less resources on the client, but vector super-overlays have a higher output quality.

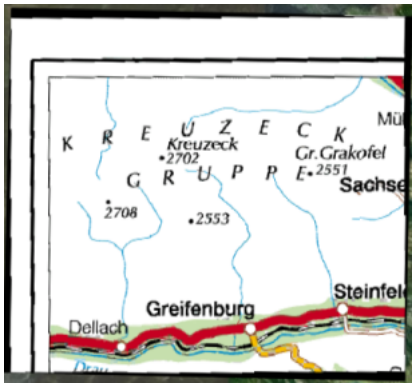
When using the *KML Reflector*, super-overlays are enabled by default, whether the data in question is raster or vector. For more information on the various options for KML super-overlay output, please see the page on the *KML Reflector*.

Raster Super-Overlays

Consider this image, which is generated from GeoServer. When zoomed out, the data is at a small size.

When zooming in, the image grows larger, but since the image is at low resolution (orignially designed to be viewed small), the quality degrades.

However, in a super-overlay, the KML document requests a new image from GeoServer of a higher resolution for that zoom level. As the new image is downloaded, the old image is replaced by the new image.



Raster Super-Overlays and GeoWebCache

GeoServer implements super-overlays in a way that is compatible with the WMS Tiling Client Recommendation. Super-overlays are generated such that the tiles of the super-overlay are the same tiles that a WMS tiling client would request. One can therefore use existing tile caching mechanisms and reap a potentially large performance benefit.

The easiest way to tile cache a raster super overlay is to use GeoWebCache which is built into GeoServer:

```
http://GEOSERVER_URL/gwc/service/kml/<layername>.<imageformat>.kmz
```

where `GEOSERVER_URL` is the URL of your GeoServer instance.

Vector Super-Overlays

GeoServer can include the feature information directly in the KML document. This has lots of benefits. It allows the user to select (click on) features to see descriptions, toggle the display of individual features, as well as have better rendering, regardless of zoom level. For large datasets, however, the feature information can take a long time to download and use a lot of client-side resources. Vector super-overlays allow the client to only download part of a dataset, and request more features as necessary.

Vector super-overlays can use the process of *KML Regionation* to organize features into a hierarchy. The regionation process can operate in a variety of modes. Most of the modes require a "regionation attribute" which is used to determine which features should be visible at a particular zoom level. Please see the *KML Regionation* page for more details.

Vector Super-Overlays and GeoWebCache

As with raster super-overlays, it is possible to cache vector super-overlays using GeoWebCache. Below is the syntax for generating a vector super-overlay KML document via GeoWebCache:

```
http://GEOSERVER_URL/gwc/service/kml/<layername>.kml.kmz
```

where `GEOSERVER_URL` is the URL of your GeoServer instance.

Unlike generating a super-overlay with the standard *KML Reflector*, it is not possible to specify the regionation properties as part of the URL. These parameters must be set in the *Layers* configuration which can be navigated to by clicking on 'Layers' in the left hand sidebar and then selecting your vector layer.

19.5.8 KML Regionation

Displaying vector features on Google Earth is a very powerful way of creating nicely-styled maps. However, it is not always optimal to display all features at all times. Displaying too many features can create an unsightly map, and can adversely affect Google Earth's performance. To combat this, GeoServer's KML output includes the ability to limit features based on certain criteria. This process is known as **regionation**. Regionation is active by default when using the superoverlay KML reflector mode.

Regionation Attributes

The most important aspect of regionation is to decide how to determine which features show up more prominently than others. This can be done either **by geometry**, or **by attribute**. One should choose the option that best exemplifies the relative "importance" of the feature. When choosing to regionate by geometry, only the larger lines and polygons will be displayed at higher zoom levels, with smaller ones being displayed when zooming in. When regionating by an attribute, the higher value of this attribute will make those features show up at higher zoom levels. (Choosing an attribute with a non-numeric value will be ignored, and will instead default to regionation by geometry.)

Regionation Strategies

Regionation strategies sets how to determine which features should be shown at any given time or zoom level. There are five types of regionation strategies:

Strategy	Description
`best_guess`	(*default*) The actual strategy is determined by the type of data being operated on. If the data consists of points, the `random` strategy is used. If the data consists of lines or polygons, the `geometry` strategy is used.
`external-sorting`	Creates a temporary auxiliary database within GeoServer. It takes slightly extra time to build the index upon first request.
`native-sorting`	Uses the default sorting algorithm of the backend where the data is hosted. It is faster than external-sorting, but will only work with PostGIS datastores.
`geometry`	Externally sorts by length (if lines) or area (if polygons).
`random`	Uses the existing order of the data and does not sort.

In most cases, the **best_guess** strategy is sufficient.

Setting Regionation Parameters

Regionation strategies and attributes are featuretype-specific, and therefore are set in the *Layers* editing page of the *Web Administration Interface*. This can be navigated to by selecting 'Layers' on the left sidebar.

19.5.9 KML Scoring

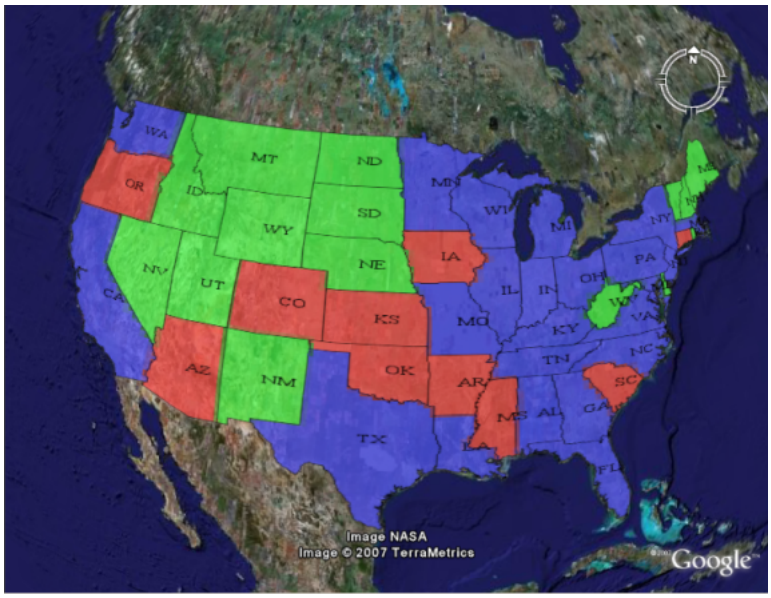
Note: KML scoring only applies when using the super-overlay mode `refresh`. See *KML Super-Overlays* for more information.

GeoServer can return KML in one of two forms. The first is as a number of placemark elements (vectors). Each placemark corresponds to a feature in the underlying dataset. This form only applies to vector datasets.

The second form is as an overlay (image). In this form the rendering is done by the GeoServer WMS and only the resulting graphic is sent to Google Earth. This is the only form available for raster datasets, but can be applied to vector datasets as well.

There are advantages to and disadvantages to each output mode when rendering vector data. Placemarks look nicer, but there can be performance problems with Google Earth if the data set is large. Overlays put less of a strain on Google Earth, but aren't as nice looking.

The following shows the same dataset rendered in Placemark form on the top and Overlay form on the bottom.



KML scoring is the process of determing whether to render features as rasters or as vectors.

The kmscore attribute

GeoServer makes the determination on whether to render a layer as raster or vector based on how many features are in the data set and an attribute called `kmscore`. The `kmscore` attribute determines the maximum amount of vector features rendered. It is calculated by this formula:

```
maximum number of features = 10^(kmscore/15)
```

The following table shows the values of this threashold for various values of the `kmscore` parameter:

kmscore	**Maximum # of features**
0	Force overlay/raster output
10	4
20	21
30	100
40	Approx. 450
50	(*default*) Approx. 2150
60	Approx. 10,000
70	Approx. 45,000
80	Approx. 200,000
90	Approx. 1,000,000
100	Force placemark/vector output

The syntax for specifying `kmscore` is:

```
kmscore=<value>
```

where `<value>` is an integer between 0 and 100. For example:

```
http://localhost:8080/geoserver/wms/kml?layers=topp:states&mode=refresh&kmscore=20
```

The `kmscore` attribute will be ignored if using a reflector mode other than `refresh`.

CHAPTER 20

Extensions

Extensions are modules that add functionality to GeoServer. They are installed as add-ons to the bsae GeoServer installation.

This section describes most of the extensions available for GeoServer. For information about extensions that add support for additional data formats, such as ArcSDE or SQL Server, see the *Working with Vector Data*, *Working with Raster Data*, and *Working with Databases* sections.

20.1 Control flow module

The `control-flow` module for GeoServer allows the administrator to control the amount of concurrent requests actually executing inside the server. This kind of control is useful for a number of reasons:

- *Performance*: tests show that, with local data sources, the maximum throughput in *GetMap* requests is achieved when allowing at most 2 times the number of CPU cores requests to run in parallel.
- *Resource control*: requests such as *GetMap* can use a significant amount of memory. The *WMS request limits* allow to control the amount of memory used per request, but an `OutOfMemoryError` is still possible if too many requests run in parallel. By controlling also the amount of requests executing it's possible to limit the total amount of memory used below the memory that was actually given to the Java Virtual Machine.
- *Fairness*: a single user should not be able to overwhelm the server with a lot of requests, leaving other users with tiny slices of the overall processing power.

The control flow method does not normally reject requests, it just queues up those in excess and executes them late. However, it's possible to configure the module to reject requests that have been waited in queue for too long.

20.1.1 Rule syntax reference

The current implementation of the control flow module reads its rules from a `controlflow.properties` property file located in the *GeoServer data directory*.

Total OWS request count

The global number of OWS requests executing in parallel can be specified with:

```
ows.global=<count>
```

Every request in excess will be queued and executed when other requests complete leaving some free execution slot.

Per request control

A per request type control can be demanded using the following syntax:

```
ows.<service>[.<request>[.<outputFormat>]]=<count>
```

Where:

- `<service>` is the OWS service in question (at the time of writing can be `wms`, `wfs`, `wcs`)
- `<request>`, optional, is the request type. For example, for the `wms` service it can be `GetMap`, `GetFeatureInfo`, `DescribeLayer`, `GetLegendGraphics`, `GetCapabilities`
- `<outputFormat>`, optional, is the output format of the request. For example, for the `wms GetMap` request it could be `image/png`, `image/gif` and so on

A few examples:

```
# don't allow more than 16 WCS requests in parallel
ows.wcs=16
# don't allow more than 8 GetMap requests in parallel
ows.wms.getmap=8
# don't allow more than 2 WFS GetFeature requests with Excel output format
ows.wfs.getfeature.application/msexcel=2
```

Per user control

There are two mechanisms to identify user requests. The first one is cookie based, so it will work fine for browsers but not as much for other kinds of clients. The second one is ip based, which works for any type of client but that can limit all the users sitting behind the same router

This avoids a single user (as identified by a cookie) to make too many requests in parallel:

```
user=<count>
```

Where `<count>` is the maximum number of requests a single user can execute in parallel.

The following avoids a single ip address from making too many requests in parallel:

```
ip=<count>
```

Where `<count>` is the maximum number of requests a single ip address can execute in parallel.

It is also possible to make this a bit more specific and throttle a single ip address instead by using the following:

```
ip.<ip_addr>=<count>
```

Where `<count>` is the maximum number of requests the ip speficied in `<ip_addr>` will execute in parallel.

To reject requests from a list of ip addresses:

```
ip.blacklist=<ip_addr1>,<ip_addr2>,...
```

Timeout

A request timeout is specified with the following syntax:

```
timeout=<seconds>
```

where `<seconds>` is the number of seconds a request can stay queued waiting for execution. If the request does not enter execution before the timeout expires it will be rejected.

20.1.2 Throttling tile requests (WMS-C, TMS, WMTS)

GeoWebCache contributes three cached tiles services to GeoServer: WMS-C, TMS, and WMTS. It is also possible to use the Control flow module to throttle them, by adding the following rule to the configuration file:

```
ows.gwc=<count>
```

Where `<count>` is the maximum number of concurrent tile requests that will be delivered by GeoWebCache at any given time.

Note also that tile request are sensitive to the other rules (user based, ip based, timeout, etc).

20.1.3 A complete example

Assuming the server we want to protect has 4 cores a sample configuration could be:

```
# if a request waits in queue for more than 60 seconds it's not worth executing,
# the client will  likely have given up by then
timeout=60
# don't allow the execution of more than 100 requests total in parallel
ows.global=100
# don't allow more than 10 GetMap in parallel
ows.wms.getmap=10
# don't allow more than 4 outputs with Excel output as it's memory bound
ows.wfs.getfeature.application/msexcel=4
# don't allow a single user to perform more than 6 requests in parallel
# (6 being the Firefox default concurrency level at the time of writing)
user=6
# don't allow the execution of more than 16 tile requests in parallel
# (assuming a server with 4 cores, GWC empirical tests show that throughput
# peaks up at 4 x number of cores. Adjust as appropriate to your system)
ows.gwc=16
```

20.2 CSS Styling

The `css` module for GeoServer adds an alternative style editor to GeoServer that uses a CSS-derived language instead of SLD. These CSS styles are internally converted to SLD, which is then used as normal by GeoServer. The CSS syntax is duplicated from SVG styling where appropriate, but extended to avoid losing facilities provided by SLD when possible. As an example, it provides facilities for extracting feature attributes to use in labelling, sizing point markers according to data values, etc.

Read on for information about:

20.2.1 Installing the CSS Module

The CSS extension is listed among the other extension downloads on the GeoServer download page. Please ensure that you download a version of the extension that corresponds to the version of GeoServer that you use.

The installation process is similar to other GeoServer plugins:

1. Download the ZIP archive. Please verify that the version number in the filename corresponds to the one reported in GeoServer's admin UI.
2. Extract the contents of the ZIP archive into the `/WEB-INF/lib/` direcotry in the GeoServer webapp. For example, if you have installed the GeoServer binary to `/opt/geoserver-2.4.0/`, you should place the CSS extension's JAR files in `/opt/geoserver-2.4.0/webapps/geoserver/WEB-INF/lib/`.
3. After extracting the extension, restart GeoServer in order for the changes to take effect. All further configuration can be done through the GeoServer web UI.

After installation, you may find the following useful to get you started - *Tutorial: Styling Data with CSS*.

Nightly builds

For those interested in trying out new features and other experimental changes, nightly builds are available from the GeoServer continuous integration system at http://ares.boundlessgeo.com/geoserver/ . After downloading the ZIP archive, the steps to install are the same as above.

20.2.2 Tutorial: Styling Data with CSS

This tutorial will walk through using CSS to style the (USA) states example data that is included with the default GeoServer installation. It also shows you the equivalent SLD code.

What you need before starting this tutorial:

- An installed copy of GeoServer 2.0 or greater. See *Installation* if you have not already installed GeoServer.
- The states layer from the default GeoServer configuration
- The CSS plugin installed. See *Installing the CSS Module* if you have not already installed the plugin.

What's in the Box?

The CSS extension adds a page to the GeoServer web UI, linked from the sidebar. This page is only visible to logged-in administrators since it can modify the styles in GeoServer.

After loading the CSS page, you can view any of the layers and styles in GeoServer by selecting them in the drop-down boxes directly beneath the map, then clicking the *Switch* link. You can overwrite any style by entering CSS into the form, but it is recommended that you avoid editing pre-existing styles since existing SLD styles are not reflected in the CSS. The *Create* link allows creating a new style with a CSS file attached to it.

Creating a States Style

The SLD file for the default states layer looks like this:

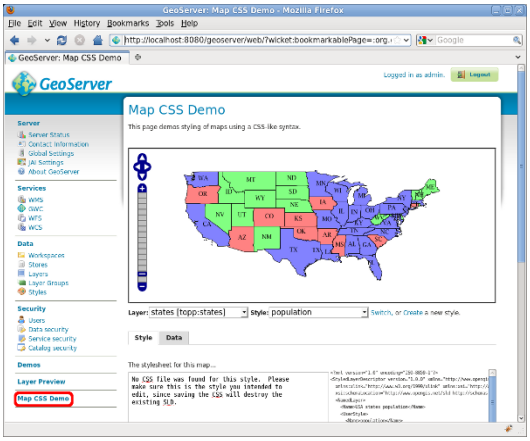


Figure 20.1: *The CSS demo page can be used to switch between layers and styles. Note the sidebar link, highlighted in red.*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StyledLayerDescriptor
  version="1.0.0"
  xmlns="http://www.opengis.net/sld"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:gml="http://www.opengis.net/gml"
  xsi:schemaLocation="http://www.opengis.net/sld
    http://schemas.opengis.net/sld/1.0.0/StyledLayerDescriptor.xsd
">
  <NamedLayer>
    <Name>USA states population</Name>
    <UserStyle>
      <Name>population</Name>
      <Title>Population in the United States</Title>
      <Abstract>A sample filter that filters the United States into three
        categories of population, drawn in different colors</Abstract>
      <FeatureTypeStyle>
        <Rule>
          <Title>&lt; 2M</Title>
          <ogc:Filter>
            <ogc:PropertyIsLessThan>
             <ogc:PropertyName>PERSONS</ogc:PropertyName>
             <ogc:Literal>2000000</ogc:Literal>
            </ogc:PropertyIsLessThan>
          </ogc:Filter>
          <PolygonSymbolizer>
             <Fill>
                <!-- CssParameters allowed are fill (the color) and fill-opacity -->
                <CssParameter name="fill">#4DFF4D</CssParameter>
                <CssParameter name="fill-opacity">0.7</CssParameter>
             </Fill>
          </PolygonSymbolizer>
        </Rule>
        <Rule>
          <Title>2M - 4M</Title>
          <ogc:Filter>
```

```
      <ogc:PropertyIsBetween>
        <ogc:PropertyName>PERSONS</ogc:PropertyName>
        <ogc:LowerBoundary>
          <ogc:Literal>2000000</ogc:Literal>
        </ogc:LowerBoundary>
        <ogc:UpperBoundary>
          <ogc:Literal>4000000</ogc:Literal>
        </ogc:UpperBoundary>
      </ogc:PropertyIsBetween>
    </ogc:Filter>
    <PolygonSymbolizer>
       <Fill>
          <!-- CssParameters allowed are fill (the color) and fill-opacity -->
          <CssParameter name="fill">#FF4D4D</CssParameter>
          <CssParameter name="fill-opacity">0.7</CssParameter>
       </Fill>
    </PolygonSymbolizer>
  </Rule>
  <Rule>
    <Title>&gt; 4M</Title>
    <!-- like a linesymbolizer but with a fill too -->
    <ogc:Filter>
      <ogc:PropertyIsGreaterThan>
       <ogc:PropertyName>PERSONS</ogc:PropertyName>
       <ogc:Literal>4000000</ogc:Literal>
      </ogc:PropertyIsGreaterThan>
    </ogc:Filter>
    <PolygonSymbolizer>
       <Fill>
          <!-- CssParameters allowed are fill (the color) and fill-opacity -->
          <CssParameter name="fill">#4D4DFF</CssParameter>
          <CssParameter name="fill-opacity">0.7</CssParameter>
       </Fill>
    </PolygonSymbolizer>
  </Rule>
  <Rule>
    <Title>Boundary</Title>
    <LineSymbolizer>
      <Stroke>
        <CssParameter name="stroke-width">0.2</CssParameter>
      </Stroke>
    </LineSymbolizer>
    <TextSymbolizer>
      <Label>
        <ogc:PropertyName>STATE_ABBR</ogc:PropertyName>
      </Label>
      <Font>
        <CssParameter name="font-family">Times New Roman</CssParameter>
        <CssParameter name="font-style">Normal</CssParameter>
        <CssParameter name="font-size">14</CssParameter>
      </Font>
      <LabelPlacement>
        <PointPlacement>
          <AnchorPoint>
            <AnchorPointX>0.5</AnchorPointX>
            <AnchorPointY>0.5</AnchorPointY>
          </AnchorPoint>
        </PointPlacement>
```

```
            </LabelPlacement>
          </TextSymbolizer>
        </Rule>
      </FeatureTypeStyle>
    </UserStyle>
    </NamedLayer>
</StyledLayerDescriptor>
```

Now, let's start on a CSS file that accomplishes the same thing. First, use the *Create* link to start a new style.

This creates an example style with the following source:

```
* {
  fill: lightgrey;
}
```

This demonstrates the basic elements of a CSS style:

A **selector** that identifies some part of the data to style. Here, the selector is `*`, indicating that all data should use the style properties.

Properties inside curly braces (`{}`) which specify how the affected features should be styled. Properties consist of name/value pairs separated by colons (`:`).

We can also see the basics for styling a polygon (`fill`), line (`stroke`), or point marker (`mark`). Note that while the stroke and fill use colors, the marker simply identifies a Well-Known Mark with the `symbol` function.

See also:

The *Filter Syntax* and *Property Listing* pages in this manual provide more information about the options available in CSS styles.

Let's use these basics to start translating the states style. The first Rule in the SLD applies to states where the PERSONS field is less than two million:

```
<Rule>
  <Title>&lt; 2M</Title>
  <ogc:Filter>
    <ogc:PropertyIsLessThan>
     <ogc:PropertyName>PERSONS</ogc:PropertyName>
     <ogc:Literal>2000000</ogc:Literal>
    </ogc:PropertyIsLessThan>
  </ogc:Filter>
  <PolygonSymbolizer>
     <Fill>
        <!-- CssParameters allowed are fill (the color) and fill-opacity -->
        <CssParameter name="fill">#4DFF4D</CssParameter>
        <CssParameter name="fill-opacity">0.7</CssParameter>
     </Fill>
  </PolygonSymbolizer>
</Rule>
```

Using a *CQL*-based selector, and copying the names and values of the CssParameters over, we get:

```
[PERSONS < 2000000] {
  fill: #4DFF4D;
  fill-opacity: 0.7;
}
```

For the second style, we have a `PropertyIsBetween` filter, which doesn't directly translate to CSS:

```
<Rule>
  <Title>2M - 4M</Title>
  <ogc:Filter>
    <ogc:PropertyIsBetween>
      <ogc:PropertyName>PERSONS</ogc:PropertyName>
      <ogc:LowerBoundary>
        <ogc:Literal>2000000</ogc:Literal>
      </ogc:LowerBoundary>
      <ogc:UpperBoundary>
        <ogc:Literal>4000000</ogc:Literal>
      </ogc:UpperBoundary>
    </ogc:PropertyIsBetween>
  </ogc:Filter>
  <PolygonSymbolizer>
     <Fill>
        <!-- CssParameters allowed are fill (the color) and fill-opacity -->
        <CssParameter name="fill">#FF4D4D</CssParameter>
        <CssParameter name="fill-opacity">0.7</CssParameter>
     </Fill>
  </PolygonSymbolizer>
</Rule>
```

However, `PropertyIsBetween` can easily be replaced by a combination of two comparison selectors. In CSS, you can apply multiple selectors to a rule by simply placing them one after the other. Selectors separated by only whitespace must ALL be satisfied for a style to apply. Multiple such groups can be attached to a rule by separating them with commas (`,`). If a feature matches any of the comma-separated groups for a rule then that style is applied. Thus, the CSS equivalent of the second rule is:

```
[PERSONS > 2000000] [PERSONS < 4000000] {
  fill: #FF4D4D;
  fill-opacity: 0.7;
}
```

The third rule can be handled in much the same manner as the first:

```
[PERSONS > 4000000] {
  fill: #4D4DFF;
  fill-opacity: 0.7;
}
```

The fourth and final rule is a bit different. It applies a label and outline to all the states:

```
<Rule>
  <Title>Boundary</Title>
  <LineSymbolizer>
    <Stroke>
      <CssParameter name="stroke-width">0.2</CssParameter>
    </Stroke>
  </LineSymbolizer>
  <TextSymbolizer>
    <Label>
      <ogc:PropertyName>STATE_ABBR</ogc:PropertyName>
    </Label>
    <Font>
      <CssParameter name="font-family">Times New Roman</CssParameter>
      <CssParameter name="font-style">Normal</CssParameter>
      <CssParameter name="font-size">14</CssParameter>
    </Font>
    <LabelPlacement>
```

```
      <PointPlacement>
        <AnchorPoint>
          <AnchorPointX>0.5</AnchorPointX>
          <AnchorPointY>0.5</AnchorPointY>
        </AnchorPoint>
      </PointPlacement>
    </LabelPlacement>
  </TextSymbolizer>
</Rule>
```

This introduces the idea of rendering an extracted value (`STATE_ABBR`) directly into the map, unlike all of the rules thus far. For this, you can use a CQL expression wrapped in square braces (`[]`) as the value of a CSS property. It is also necessary to surround values containing whitespace, such as `Times New Roman`, with single- or double-quotes (`",'`). With these details in mind, let's write the rule:

```
* {
  stroke-width: 0.2;
  label: [STATE_ABBR];
  label-anchor: 0.5 0.5;
  font-family: "Times New Roman";
  font-style: normal;
  font-size: 14;
}
```

Putting it all together, you should now have a style that looks like:

```
[PERSONS < 2000000] {
  fill: #4DFF4D;
  fill-opacity: 0.7;
}

[PERSONS > 2000000] [PERSONS < 4000000] {
  fill: #FF4D4D;
  fill-opacity: 0.7;
}

[PERSONS > 4000000] {
  fill: #4D4DFF;
  fill-opacity: 0.7;
}

* {
  stroke-width: 0.2;
  label: [STATE_ABBR];
  label-anchor: 0.5 0.5;
  font-family: "Times New Roman";
  font-style: normal;
  font-size: 14;
}
```

Press the *Submit* button at the bottom of the CSS form to save your changes and see your style applied to the states layer.

Surprise! The borders are missing. What happened? In the GeoServer CSS module, each type of symbolizer has a "key" property which controls whether it is applied. Without these "key" properties, subordinate properties are ignored. These "key" properties are:

- **fill**, which controls whether or not Polygon fills are applied. This specified the color or graphic to use for the fill.

- **stroke**, which controls whether or not Line and Polygon outline strokes are applied. This specifies the color (or graphic fill) of the stroke.
- **mark**, which controls whether or not point markers are drawn. This identifies a Well-Known Mark or image URL to use.
- **label**, which controls whether or not to draw labels on the map. This identifies the text to use for labeling the map, usually as a CQL expression.
- **halo-radius**, which controls whether or not to draw a halo around labels. This specifies how large such halos should be.

See also:

The *Property Listing* page in this manual for information about the other properties.

Since we don't specify a `stroke` color, no stroke is applied. Let's add it, so that that last rule ends up looking like:

```
* {
  stroke: black;
  stroke-width: 0.2;
  label: [STATE_ABBR];
  label-anchor: 0.5 0.5;
  font-family: "Times New Roman";
  font-style: normal;
  font-size: 14;
}
```

Refining the Style

Removing Duplicated Properties

The style that we have right now is only 23 lines, a nice improvement over the 103 lines of XML that we started with. However, we are still repeating the `fill-opacity` attribute everywhere. We can move it into the `*` rule and have it applied everywhere. This works because the GeoServer CSS module emulates **cascading**, the "C" part of "CSS". While SLD uses a painter's model where each rule is processed independently, a cascading style allows you to provide general style properties and override only specific properties for particular features. Anyway, this takes the style down to only 21 lines:

```
[PERSONS < 2000000] {
  fill: #4DFF4D;
}

[PERSONS > 2000000] [PERSONS < 4000000] {
  fill: #FF4D4D;
}

[PERSONS > 4000000] {
  fill: #4D4DFF;
}

* {
  fill-opacity: 0.7;
  stroke-width: 0.2;
  label: [STATE_ABBR];
  label-anchor: 0.5 0.5;
  font-family: "Times New Roman";
```

```
  font-style: normal;
  font-size: 14;
}
```

Scale-Dependent Styles

The labels for this style are nice, but at lower zoom levels they seem a little crowded. We can easily move the labels to a rule that doesn't activate until the scale denominator is below 2000000. We do want to keep the stroke and fill-opacity at all zoom levels, so we can separate them from the label properties:

```
* {
  fill-opacity: 0.7;
  stroke-width: 0.2;
}

[@scale < 20000000] {
  label: [STATE_ABBR];
  label-anchor: 0.5 0.5;
  font-family: "Times New Roman";
  font-style: normal;
  font-size: 14;
}
```

Setting Titles for the Legend

So far, we haven't set titles for any of the style rules. This doesn't really cause any problems while viewing maps, but GeoServer uses the title in auto-generating legend graphics. Without the titles, GeoServer falls back on the names, which in the CSS module are generated from the filters for each rule. Titles are not normally a part of CSS, so GeoServer looks for them in specially formatted comments before each rule. We can add titles like so:

```
/* @title Population < 2M */
[PERSONS < 2000000] {
  fill: #4DFF4D;
  fill-opacity: 0.7;
}

/* @title 2M < Population < 4M */
[PERSONS > 2000000] [PERSONS < 4000000] {
  fill: #FF4D4D;
  fill-opacity: 0.7;
}

/* @title Population > 4M */
[PERSONS > 4000000] {
  fill: #4D4DFF;
  fill-opacity: 0.7;
}

/* @title Boundaries */
* {
  stroke-width: 0.2;
  label: [STATE_ABBR];
  label-anchor: 0.5 0.5;
  font-family: "Times New Roman";
```

```
  font-style: normal;
  font-size: 14;
}
```

Because of the way that CSS is translated to SLD, each SLD rule is a combination of several CSS rules. This is handled by combining the titles with the word "with". If the title is omitted for a rule, then it is simply not included in the SLD output.

20.2.3 Filter Syntax

Filters limit the set of features affected by a rule's properties. There are several types of simple filters, which can be combined to provide more complex filters for rules.

Combining Filters

Combination is done in the usual CSS way. A rule with two filters separated by a comma affects any features that match *either* filter, while a rule with two filters separated by only whitespace affects only features that match *both* filters. Here's an example using a basic attribute filter (described below):

```
/* Matches places where the lake is flooding */
[rainfall>12] [lakes>1] {
    fill: black;
}

/* Matches wet places */
[rainfall>12], [lakes>1] {
    fill: blue;
}
```

Filtering on Data Attributes

An attribute filter matches some attribute of the data (for example, a column in a database table). This is probably the most common type of filter. An attribute filter takes the form of an attribute name and a data value separated by some predicate operator (such as the less-than operator <).

Supported predicate operators include the following:

Operator	Meaning
=	The property must be exactly *equal* to the specified value.
<>	The property must not be exactly equal to the specified value.
>	The property must be greater than (or alphabetically later than) the specified value.
>=	The property must be greater than or equal to the specified value.
<	The property must be less than (or alphabetically earlier than) the specified value.
<=	The property must be less than or equal to the specified value.
LIKE	The property must match the pattern described by the specified value. Patterns use _ to indicate a single unspecified character and % to indicate an unknown number of unspecified characters.

For example, to only render outlines for the states whose names start with letters in the first half of the alphabet, the rule would look like:

```
[STATE_NAME<='M'] {
    stroke: black;
}
```

Note: The current implementation of property filters uses ECQL syntax, described on the GeoTools documentation.

Filtering on Type

When dealing with data from multiple sources, it may be useful to provide rules that only affect one of those sources. This is done very simply; just specify the name of the layer as a filter:

```
states {
    stroke: black;
}
```

Filtering by ID

For layers that provide feature-level identifiers, you can style specific features simply by specifying the ID. This is done by prefixing the ID with a hash sign (#):

```
#states.2 {
    stroke: black;
}
```

Note: In CSS, the `.` character is not allowed in element ids; and the `#states.foo` selector matches the element with id `states` only if it also has the class `foo`. Since this form of identifier comes up so frequently in GeoServer layers, the CSS module deviates from standard CSS slightly in this regard. Future revisions may use some form of munging to avoid this deviation.

Filtering by Rendering Context (Scale)

Often, there are aspects of a map that should change based on the context in which it is being viewed. For example, a road map might omit residential roads when being viewed at the state level, but feature them prominently at the neighborhood level. Details such as scale level are presented as pseudo-attributes; they look like property filters, but the property names start with an `@` symbol:

```
[roadtype='Residential'][@scale>100000] {
    stroke: black;
}
```

The context details that are provided are as follows:

Pseudo-Attribute	Meaning
@scale	The scale denominator for the current rendering. More explicitly, this is the ratio of real-world distance to screen/rendered distance.

Note: While property filters (currently) use the more complex ECQL syntax, pseudo-attributes cannot use complex expressions and MUST take the form of <PROPERTY><OPERATOR><LITERAL>.

Filtering Symbols

When using symbols to create graphics inline, you may want to apply some styling options to them. You can specify style attributes for built-in symbols by using a few special selectors:

PseudoSelector	Meaning
:mark	specifies that a rule applies to symbols used as point markers
:stroke	specifies that a rule applies to symbols used as stroke patterns
:fill	specifies that a rule applies to symbols used as fill patterns
:symbol	specifies that a rule applies to any symbol, regardless of which context it is used in
:nth-mark(n)	specifies that a rule applies to the symbol used for the nth stacked point marker on a feature.
:nth-stroke(n)	specifies that a rule applies to the symbol used for the nth stacked stroke pattern on a feature.
:nth-fill(n)	specifies that a rule applies to the symbol used for the nth stacked fill pattern on a feature.
:nth-symbol(n)	specifies that a rule applies to the symbol used for the nth stacked symbol on a feature, regardless of which context it is used in.

For more discussion on using these selectors, see *Styled Marks in CSS*.

Not Filtering at All

Sometimes it is useful to have a rule that matches all features, for example, to provide some default styling for your map (remember, by default nothing is rendered). This is accomplished using a single asterisk * in place of the usual filter. This catch-all rule can be used in complex expressions, which may be useful if you want a rule to provide defaults as well as overriding values for some features:

```
* {
    stroke: black;
}
```

20.2.4 Providing Metadata

One feature that appears in SLD that has no analog in CSS is the ability to provide *metadata* for styles and style rules. For example, this SLD embeds a title for its single rule:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StyledLayerDescriptor version="1.0.0"
    xmlns="http://www.opengis.net/sld"
    xmlns:ogc="http://www.opengis.net/ogc"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:gml="http://www.opengis.net/gml"
    xsi:schemaLocation="http://www.opengis.net/sld
        http://schemas.opengis.net/sld/1.0.0/StyledLayerDescriptor.xsd"
>
  <NamedLayer>
    <Name>Country Borders</Name>
    <UserStyle>
      <Name>borders</Name>
      <Title>Country Borders</Title>
      <Abstract>
          Borders of countries, in an appropriately sovereign aesthetic.
      </Abstract>
```

```
      <FeatureTypeStyle>
        <Rule>
          <Title>Borders</Title>
          <LineSymbolizer>
            <Stroke>
              <CssParameter name="stroke-width">0.2</CssParameter>
            </Stroke>
          </LineSymbolizer>
        </Rule>
      </FeatureTypeStyle>
    </UserStyle>
  </NamedLayer>
</StyledLayerDescriptor>
```

Software such as GeoServer can use this metadata to automatically generate nice legend images directly from the style. You don't have to give up this ability when styling maps in CSS; just add comment *before* your rules including lines that start with '@title' and '@abstract'. Here is the analogous style in CSS:

```
/*
 * @title This is a point layer.
 * @abstract This is an abstract point layer.
 */
* {
    mark: mark(circle);
}
```

Rules can provide either a title, an abstract, both, or neither. The SLD Name for a rule is autogenerated based on the filters from the CSS rules that combined to form it, for aid in troubleshooting.

Combined Rules

One thing to keep in mind when dealing with CSS styles is that multiple rules may apply to the same subset of map features, especially as styles get more complicated. Metadata is inherited similarly to CSS properties, but metadata fields are **combined** instead of overriding less specific rules. That means that when you have a style like this:

```
/* @title Borders */
* {
    stroke: black;
}

/* @title Parcels */
[category='parcel'] {
    fill: blue;
}
```

The legend entry for parcels will have the title `'Parcels with Borders'`. If you don't like this behavior, then only provide titles for the most specific rules in your style. (Or, suggest something better in an issue report!) Rules that don't provide titles are simply omitted from title aggregation.

20.2.5 Multi-Valued Properties

When rendering maps, it is sometimes useful to draw the same feature multiple times. For example, you might want to stroke a roads layer with a thick line and then a slimmer line of a different color to create a halo effect.

In GeoServer's `css` module, all properties may have multiple values. There is a distinction between complex properties, and multi-valued properties. Complex properties are separated by spaces, while multi-valued properties are separated by commas. So, this style fills a polygon once:

```
* {
    fill: url("path/to/img.png") red;
}
```

Using `red` as a fallback color if the image cannot be loaded. If you wanted to draw red on top of the image, you would have to style like so:

```
* {
    fill: url("path/to/img.png"), red;
    /* set a transparency for the second fill,
       leave the first fully opaque. */
    fill-opacity: 100%, 20%;
}
```

For each type of symbolizer (`fill`, `mark`, `stroke`, and `label`) the number of values determines the number of times the feature will be drawn. For example, you could create a bulls-eye effect by drawing multiple circles on top of each other with decreasing sizes:

```
* {
    mark: symbol(circle), symbol(circle), symbol(circle), symbol(circle);
    mark-size: 40px, 30px, 20px, 10px;
}
```

If you do not provide the same number of values for an auxiliary property, the list will be repeated as many times as needed to finish. So:

```
* {
    mark: symbol(circle), symbol(circle), symbol(circle), symbol(circle);
    mark-size: 40px, 30px, 20px, 10px;
    mark-opacity: 12%;
}
```

makes all those circles 12% opaque. (Note that they are all drawn on top of each other, so the center one will appear 4 times as solid as the outermost one.)

Inheritance

For purposes of inheritance/cascading, property lists are treated as indivisible units. For example:

```
* {
    stroke: red, green, blue;
    stroke-width: 10px, 6px, 2px;
}

[type='special'] {
    stroke: pink;
}
```

This style will draw the 'special' features with only one outline. It has `stroke-width: 10px, 6px, 2px;` so that outline will be 10px wide.

20.2.6 Property Listing

This page lists the supported rendering properties. See *CSS Value Types* for more information about the value types for each.

Point Symbology

Property	Type	Meaning	Accepts Express -ion?
mark	url, symbol	The image or well-known shape to render for points	yes
mark-mime	string (MIME Type)	The type of the image referenced by a url()	No, defaults to 'image/jpeg'
mark-geometry	expression	An expression to use for the geometry when rendering features	yes
mark-size	length	The width to assume for the provided image. The height will be adjusted to preserve the source aspect ratio.	yes
mark-rotation	angle	A rotation to be applied (clockwise) to the mark image.	yes
-gt-mark-label-obstacle	boolean	If true the point symbol will be consider an obstable for labels, no label will overlap it	no

Line Symbology

Property	Type	Meaning	Accepts Expression?
stroke	color, url, symbol	The color, graphic, or well-known shape to use to stroke lines or outlines	yes
stroke-geometry	expression	An expression to use for the geometry when rendering features.	yes
stroke-mime	string (MIME Type)	The type of the image referenced by a url()	No, defaults to 'image/jpeg'
stroke-opacity	percentage	A value in the range of 0 (fully transparent) to 1.0 (fully opaque)	yes
stroke-width	length	The width to use for stroking the line.	yes
stroke-size	length	An image or symbol used for the stroke pattern will be stretched or squashed to this size before rendering. If this value differs from the stroke-width, the graphic will be repeated or clipped as needed.	yes
stroke-rotation	angle	A rotation to be applied (clockwise) to the stroke image. See also the stroke- repeat property.	yes
stroke-linecap	keyword: butt, square, round	The style to apply to the ends of lines drawn	yes
stroke-linejoin	keyword: miter, round, bevel	The style to apply to the "elbows" where segments of multi-line features meet.	yes
stroke-dasharray	list of lengths	The lengths of segments to use in a dashed line.	no
stroke-dashoffset	length	How far to offset the dash pattern from the ends of the lines.	yes |
stroke-repeat	keyword: repeat, stipple	How to use the provided graphic to paint the line. If repeat, then the graphic is repeatedly painted along the length of the line (rotated appropriately to match the line's direction). If stipple, then the line is treated as a polygon to be filled.	yes
-gt-stroke-label-obstacle	boolean	If true the line will be consider an obstable for labels, no label will overlap it	no

Polygon Symbology

Property	Type	Meaning	Accepts Expression?
fill	color, url, symbol	The color, graphic, or well-known shape to use to stroke lines or outlines	yes
fill-geometry	expression	An expression to use for the geometry when rendering features.	yes
fill-mime	string (MIME Type)	The type of the image referenced by a url()	No, defaults to 'image/jpeg'
fill-opacity	percentage	A value in the range of 0 (fully transparent) to 1.0 (fully opaque)	yes
fill-size	length	The width to assume for the image or graphic provided.	yes
fill-rotation	angle	A rotation to be applied (clockwise) to the fill image.	yes
-gt-fill-label-obstacle	boolean	If true the polygon will be consider an obstable for labels, no label will overlap it	no
-gt-graphic-marging	List of lengths	A list of 1 to 4 values, specifying the space between repeated graphics in a texture paint. One value is uniform spacing in all directions, two values are considered top/bottom and right/left, three values are considered top, right/left, bottom, four values are read as top,right,bottom,left.	no
-gt-graphic-marging	List of lengths	A list of 1 to 4 values, specifying the space between repeated graphics in a texture paint. One value is uniform spacing in all directions, two values are considered top/bottom and right/left, three values are considered top, right/left, bottom, four values are read as top,right,bottom,left.	no
-gt-random	none,grid,free	Activates random distribution of symbols in a texture fill tile. See *Fills with randomized symbols* for details. Defaults to "none"	no
-gt-random-seed	integer number	The seed for the random generator. Defaults to 0	no
-gt-random-rotation	none/free	When set to "free" activates random rotation of the symbol in addition to random distribution. Defaults to "none"	no
-gt-random-symbol-count	positive integer number	Number of suymbols to be placed in the texture fill tile. May not be respected due to location conflicts (no two symbols are allowed to overlap). Defaults to 16.	no
-gt-random-tile-size	positive integer number	Size of the texture paint tile that will be filled with the random symbols. Defaults to 256.	no

Text Symbology (Labeling)

Property	Type	Meaning	Accepts Expression?
label	string	The text to display as labels for features	yes
label-geometry	expression	An expression to use for the geometry when rendering features.	yes
label-anchor	expression	The part of the label to place over the point or middle of the polygon. This takes 2 values - x y where x=0 is the left edge of the label, x=1 is the right edge. y=0 is the bottom edge of the label, y=1 is the top edge. Specify 0.5 0.5 to centre a label.	yes
label-offset	expression	This is for fine-tuning label-anchor. x and y values specify pixels to adjust the label position. For lines, a single value will make the label be parallel to the line, at the given distance, while two values will force a point style placement, with the label painted horizonally at the center of the line (plus the given offsets)	yes
label-rotation	expression	Clockwise rotation of label in degrees.	yes
label-z-index	expression	Used to determine which labels are drawn on top of other labels. Lower z-indexes are drawn on top.	yes
shield	mark, symbol	A graphic to display behind the label, such as a highway shield.	yes
shield-mime	string (MIME Type)	The type of the image referenced by a url()	No, defaults to 'image/jpeg'
font-family	string	The name of the font or font family to use for labels	yes
font-fill	fill	The fill to use when rendering fonts	yes
font-style	keyword: normal, italic, oblique	The style for the lettering	yes
font-weight	keyword: normal, bold	The weight for the lettering	yes
font-size	length	The size for the font to display.	yes
halo-radius	length	The size of a halo to display around the lettering (to enhance readability). This is *required* to activate the halo feature.	yes
halo-color	color	The color for the halo	yes
halo-opacity	percentage	The opacity of the halo, from 0 (fully transparent) to 1.0 (fully opaque).	yes
-gt-label-padding	length	The amount of 'padding' space to provide around labels. Labels will not be rendered closer together than this threshold. This is equivalent to the *spaceAround* vendor parameter.	no
-gt-label-group	one of: `true` or `false`	If true, the render will treat features with the same label text as a single feature for the purpose of labeling. This is equivalent to the *group* vendor parameter.	no
-gt-label-max-displacement	length	If set, this is the maximum displacement that the renderer will apply to a label. Labels that need larger displacements to avoid collisions will simply be omitted. This is equivalent to the *maxDisplacement* vendor parameter.	no

Text Symbology (Labeling) - continued

Property	Type	Meaning	Accepts Expression?
-gt-label-min-group-distance	length	This is equivalent to the minGroupDistance vendor parameter in SLD.	no
-gt-label-repeat	length	If set, the renderer will repeat labels at this interval along a line. This is equivalent to the *repeat* vendor parameter.	no
-gt-label-all-group	one of `true` or `false`	when using grouping, whether to label only the longest line that could be built by merging the lines forming the group, or also the other ones. This is equivalent to the *allGroup* vendor parameter.	no
-gt-label-remove-overlaps	one of `true` or `false`	If enabled, the renderer will remove overlapping lines within a group to avoid duplicate labels. This is equivalent to the removeOverlaps vendor parameter.	no
-gt-label-allow-overruns	one of `true` or `false`	Determines whether the renderer will show labels that are longer than the lines being labelled. This is equivalent to the allowOverrun vendor parameter.	no
-gt-label-follow-line	one of `true` or `false`	If enabled, the render will curve labels to follow the lines being labelled. This is equivalent to the *followLine* vendor parameter.	no
-gt-label-max-angle-delta	one of `true` or `false`	The maximum amount of curve allowed between two characters of a label; only applies when '-gt-follow-line: true' is set. This is equivalent to the *maxAngleDelta* vendor parameter.	no
-gt-label-auto-wrap	length	Labels will be wrapped to multiple lines if they exceed this length in pixels. This is equivalent to the *autoWrap* vendor parameter.	no
-gt-label-force-ltr	one of `true` or `false`	By default, the renderer will flip labels whose normal orientation would cause them to be upside-down. Set this parameter to false if you are using some icon character label like an arrow to show a line's direction. This is equivalent to the *forceLeftToRight* vendor parameter.	no
-gt-label-conflict-resolution	one of `true` or `false`	Set this to false to disable label conflict resolution, allowing overlapping labels to be rendered. This is equivalent to the *conflictResolution* vendor parameter.	no
-gt-label-fit-goodness	scale	The renderer will omit labels that fall below this "match quality" score. The scoring rules differ for each geometry type. This is equivalent to the *goodnessOfFit* vendor parameter.	no
-gt-label-priority	expression	Specifies an expression to use in determining which features to prefer if there are labeling conflicts. This is equivalent to the *Priority* SLD extension.	yes

Text Symbology (Labeling) - continued

Property	Type	Meaning	Accepts Expression?
-gt-shield-resize	string, one of `none`, `stretch`, or `proportional`	Specifies a mode for resizing label graphics (such as highway shields) to fit the text of the label. The default mode, 'none', never modifies the label graphic. In `stretch` mode, GeoServer will resize the graphic to exactly surround the label text, possibly modifying the image's aspect ratio. In `proportional` mode, GeoServer will expand the image to be large enough to surround the text while preserving its original aspect ratio.	none
-gt-shield-margin	list of lengths, one to four elements long.	Specifies an extra margin (in pixels) to be applied to the label text when calculating label dimensions for use with the `-gt-shield-resize` option. Similar to the `margin` shorthand property in CSS for HTML, its interpretation varies depending on how many margin values are provided: 1 = use that margin length on all sides of the label 2 = use the first for top & bottom margins and the second for left & right margins. 3 = use the first for the top margin, second for left & right margins, third for the bottom margin. 4 = use the first for the top margin, second for the right margin, third for the bottom margin, and fourth for the left margin.	none

Raster Symbology

Property	Type	Meaning	Accepts Expression?
raster-channels	string	The list of raster channels to be used in the output. It can be "auto" to make the renderer choose the best course of action, or a list of band numbers, a single one will generate a gray image, three will generate an RGB one, four will generate a RGBA one. E.g., "1 3 7" to choose the first, third and seventh band of the input raster to make a RGB image	no
raster-geometry	expression	The attribute containing the raster to be painted. Normally not needed, but it would work if you had a custom vector data source that contains a GridCoverage attribute, in order to select it	yes
raster-opacity	floating point	A value comprised between 0 and 1, 0 meaning completely transparent, 1 meaning completely opaque. This controls the whole raster trasparency.	no
raster-contrast-enhancement	string	Allows to stretch the range of data/colors in order to enhance tiny differences. Possible values are 'normalize', 'histogram' and 'none'	no
raster-gamma	floating point	Gamma adjustment for the output raster	no
raster-z-index	integer	Controls the z ordering of the raster output	no
raster-color-map	string	Applies a color map to single banded input. The contents is a space separate list of `color-map-entry(color, value)` (opacity assumed to be 1), or `color-map-entry(color, value, opacity)`. The values must be provided in increasing order.	no
raster-color-map-type	string	Controls how the color map entries are interpreted, the possible values are "ramp", "intervals" and "values", with ramp being the default if no "raster-color-map-type" is provided. The default "ramp" behavior is to linearly interpolate color between the provided values, and assign the lowest color to all values below the lowest value, and the highest color to all values above the highest value. The "intervals" behavior instead assigns solid colors between values, whilst "values" only assigns colors to the specified values, every other value in the raster is not painted at all	no

Shared

Property	Type	Meaning	Accepts Expression?
geometry	expression	An expression to use for the geometry when rendering features. This provides a geometry for all types of symbology, but can be overridden by the symbol-specific geometry properties.	yes

Symbol Properties

These properties are applied only when styling built-in symbols. See *Styled Marks in CSS* for details.

Property	Type	Meaning	Accepts Expression?
size	length	The size at which to render the symbol.	yes
rotation	angle	An angle through which to rotate the symbol.	yes

20.2.7 CSS Value Types

This page presents a brief overview of CSS types as used by this project. Note that these can be repeated as described in *Multi-Valued Properties*.

Numbers

Numeric values consist of a number, or a number annotated with a measurement value. In general, it is wise to use measurement annotations most of the time, to avoid ambiguity and protect against potential future changes to the default units.

Currently, the supported units include:

- Length
  - `px` pixels
  - `m` meters
  - `ft` feet
- Angle
  - `deg` degrees
- Ratio
  - `%` percentage

When using expressions in place of numeric values, the first unit listed for the type of measure is assumed.

Since the CSS module translates styles to SLD before any rendering occurs, its model of unit-of-measure is tied to that of SLD. In practice, this means that for any particular symbolizer, there only one unit-of-measure applied for the style. Therefore, the CSS module extracts that unit-of-measure from one special property for each symbolizer type. Those types are listed below for reference:

- `fill-size` determines the unit-of-measure for polygon symbolizers (but that doesn't matter so much since it is the only measure associated with fills)
- `stroke-width` determines the unit-of-measure for line symbolizers
- `mark-size` determines the unit-of-measure for point symbolizers
- `font-size` determines the unit-of-measure for text symbolizers and the associated halos

Strings

String values consist of a small snippet of text. For example, a string could be a literal label to use for a subset of roads:

```
[lanes>20] {
        label: "Serious Freaking Highway";
}
```

Strings can be enclosed in either single or double quotes. It's easiest to simply use whichever type of quotes are not in your string value, but you can escape quote characters by prefixing them with a backslash `\`. Backslash characters themselves must also be prefixed. For example, `'\\\''` is a string value consisting of a single backslash followed by a single single quote character.

Labels

While labels aren't really a special type of value, they deserve a special mention since labels are more likely to require special string manipulation than other CSS values.

If a label is a simple string value, then it works like any other string would:

```
[lanes > 20] {
    label: "Serious Freaking Highway";
}
```

However, if a label has multiple values, all of those values will be concatenated to form a single label:

```
[lanes > 20] {
   label: "Serious " "Freaking " "Highway";
}
```

Note the whitespace within the label strings here; *no whitespace is added* when concatenating strings, so you must be explicit about where you want it included. You can also mix CQL expressions in with literal string values here:

```
states {
   label: [STATE_NAME] " (" [STATE_ABBR] ")";
}
```

Note: This automatic concatenation is currently a special feature only provided for labels. However, string concatenation is also supported directly in CQL expressions by using the `strConcat` filter function:

```
* { fill: [strConcat('#', color_hex)]; }
```

This form of concatenation works with any property that supports expressions.

Colors

Color values are relatively important to styling, so there are multiple ways to specify them.

Format	Interpretation
`#RRGGBB`	A hexadecimal-encoded color value, with two digits each for red, green, and blue.
`#RGB`	A hexadecimal-encoded color value, with one digits each for red, green, and blue. This is equivalent to the two-digit-per-channel encoding with each digit duplicated.
`rgb(r, g, b)`	A three-part color value with each channel represented by a value in the range 0 to 1, or in the range 0 to 255. 0 to 1 is used if any of the values include a decimal point, otherwise it is 0 to 255.
Simple name	The simple English name of the color. A full list of the supported colors is available at http://www.w3.org/TR/SVG/types.html#ColorKeywords

External References

When using external images to decorate map features, it is necessary to reference them by URL. This is done by a call to the `url` function. The URL value may be wrapped in single or double-quotes, or not at all.

The same escaping rules as for string values. The `url` function is also a special case where the surrounding quote marks can usually be omitted. Some examples:

```
/* These properties are all equivalent. */

* {
    stroke: url("http://example.com/");
    stroke: url('http://example.com/');
    stroke: url(http://example.com/);
}
```

Note: While relative URLs are supported, they will be fully resolved during the conversion process to SLD and written out as absolute URLs. This may be cause problems when relocating data directories, etc. The style can be regenerated with the current correct URL by opening it in the demo editor and using the Submit button there.

Well-Known Marks

As defined in the SLD standard, GeoServer's `css` module also allows using a certain set of well-known mark types without having to provide graphic resources explicitly. These include:

- `circle`
- `square`
- `cross`
- `star`
- `arrow`

And others. Additionally, vendors can provide an extended set of well-known marks, a facet of the standard that is exploited by some GeoTools plugins to provide dynamic map features such as using characters from TrueType fonts as map symbols, or dynamic charting. In support of these extended mark names, the css module provides a `symbol` function similar to `url`. The syntax is the same, aside from the function name:

```
* {
    mark: symbol(circle);
    mark: symbol('ttf://Times+New+Roman&char=0x19b2');
    mark: symbol("chart://type=pie&x&y&z");
}
```

20.2.8 Styled Marks in CSS

GeoServer's CSS module provides a collection of predefined symbols that you can use and combine to create simple marks, strokes, and fill patterns without needing an image editing program. You can access these symbols via the symbol() CSS function. For example, the built-in circle symbol makes it easy to create a simple 'dot' marker for a point layer:

```
* {
  mark: symbol(circle);
}
```

Symbols work anywhere you can use a `url()` to reference an image (ie, you can use symbols for stroke and fill patterns as well as markers.)

Symbol Names

GeoServer extensions can add extra symbols (such as the `chart://` symbol family which allows the use of charts as symbols via a naming scheme similar to the Google Charts API). However, there are a few symbols that are always available:

- circle
- square
- triangle
- arrow
- cross
- star
- x
- shape://horizline
- shape://vertline
- shape://backslash
- shape://slash
- shape://plus
- shape://times

Symbol Selectors

Symbols offer some additional styling options beyond those offered for image references. To specify these style properties, just add another rule with a special selector. There are 8 "pseudoclass" selectors that are used to style selectors:

- `:mark` specifies that a rule applies to symbols used as point markers
- `:shield` specifies that a rule applies to symbols used as label shields (icons displayed behind label text)
- `:stroke` specifies that a rule applies to symbols used as stroke patterns
- `:fill` specifies that a rule applies to symbols used as fill patterns
- `:symbol` specifies that a rule applies to any symbol, regardless of which context it is used in
- `:nth-mark(n)` specifies that a rule applies to the symbol used for the nth stacked point marker on a feature.
- `:nth-shield(n)` specifies that a rule applies to the symbol used for the background of the nth stacked label on a feature
- `:nth-stroke(n)` specifies that a rule applies to the symbol used for the nth stacked stroke pattern on a feature.
- `:nth-fill(n)` specifies that a rule applies to the symbol used for the nth stacked fill pattern on a feature.
- `:nth-symbol(n)` specifies that a rule applies to the symbol used for the nth stacked symbol on a feature, regardless of which context it is used in.

Symbol Styling Properties

Styling a built-in symbol is similar to styling a polygon feature. However, the styling options are slightly different from those available to a true polygon feature:

- The `mark` and `label` families of properties are unavailable for symbols.
- Nested symbol styling is not currently supported.
- Only the first `stroke` and `fill` will be used.
- Additional `size` (as a length) and `rotation` (as an angle) properties are available. These are analogous to the `(mark|stroke|fill)-size` and `(mark|stroke|fill)-rotation` properties available for true geometry styling.

Note: The various prefixed '-size' and '-rotation' properties on the containing style override those for the symbol if they are present.

Example Styled Symbol

As an example, consider a situation where you are styling a layer that includes data about hospitals in your town. You can create a simple hospital logo by placing a red cross symbol on top of a white circle background:

```
[usage='hospital'] {
  mark: symbol('circle'), symbol('cross');
}

[usage='hospital'] :nth-mark(1) {
  size: 16px;
  fill: white;
  stroke: red;
}

[usage='hospital'] :nth-mark(2) {
  size: 12px;
  fill: red;
}
```

20.2.9 CSS Cookbook

The CSS Cookbook is a collection of CSS "recipes" for creating various types of map styles. Wherever possible, each example is designed to show off a single CSS feature so that code can be copied from the examples and adapted when creating CSS styles of your own. Most examples are shared with the SLD Cookbook, to make a comparison between the two syntaxes immediate.

The CSS Cookbook is divided into four sections: the first three for each of the vector types (points, lines, and polygons) and the fourth section for rasters. Each example in every section contains a screen-shot showing actual GeoServer WMS output and the full CSS code for reference.

Each section uses data created especially for the Cookbooks (both CSS and SLD), with shapefiles for vector data and GeoTIFFs for raster data. The projection for data is EPSG:4326. All files can be easily loaded into GeoServer in order to recreate the examples.

Data Type	Shapefile
Point	`sld_cookbook_point.zip`
Line	`sld_cookbook_line.zip`
Polygon	`sld_cookbook_polygon.zip`
Raster	`sld_cookbook_raster.zip`

Points

While points are seemingly the simplest type of shape, possessing only position and no other dimensions, there are many different ways that a point can be styled in CSS.

Example points layer

The `points layer` used for the examples below contains name and population information for the major cities of a fictional country. For reference, the attribute table for the points in this layer is included below.

fid (Feature ID)	**name** (City name)	**pop** (Population)
point.1	Borfin	157860
point.2	Supox City	578231
point.3	Ruckis	98159
point.4	Thisland	34879
point.5	Synopolis	24567
point.6	San Glissando	76024
point.7	Detrania	205609

`Download the points shapefile`

Simple point

This example specifies points be styled as red circles with a diameter of 6 pixels.

Figure 20.2: *Simple point*

Code

```
1  * {
2    mark: symbol(circle);
3    mark-size: 6px;
```

```
4   }
5
6   :mark {
7     fill: red;
8   }
```

Details There are two rules in this CSS, the first one (**lines 1-4**) matches all features, and asks them to be depicted with a circular mark, 6 pixels wide. The second rule uses a symbol selector, `:mark`, which selects all marks in the previous rules, and allows to specify how to fill the contents of the circle, in this case, with a solid red fill (a stand alone fill property would have been interpreted as the request to fill all polygons in the input with solid red instead).

Simple point with stroke

This example adds a stroke (or border) around the *Simple point*, with the stroke colored black and given a thickness of 2 pixels.

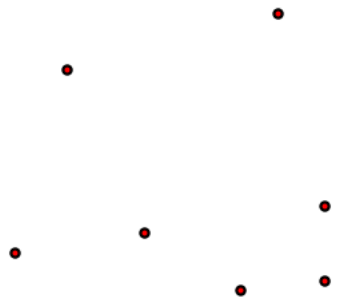


Figure 20.3: *Simple point with stroke*

Code

```
1   * {
2     mark: symbol(circle);
3     mark-size: 6px;
4   }
5
6   :mark {
7     fill: red;
8     stroke: black;
9     stroke-width: 2px;
10  }
```

Details This example is similar to the *Simple point* example, in this case a stroke and a stroke width have been specified in the mark selector in order to apply them to the circle symbols.

Rotated square

This example creates a square instead of a circle, colors it green, sizes it to 12 pixels, and rotates it by 45 degrees.

Figure 20.4: *Rotated square*

Code

```
1  * {
2    mark: symbol(square);
3    mark-size: 12px;
4    mark-rotation: 45;
5  }
6
7  :mark {
8    fill: #009900;
9  }
```

Details In this example, **line 2** sets the shape to be a square, with **line 8** setting the color to a dark green (`#009900`). **Line 3** sets the size of the square to be 12 pixels, and **line 4** set the rotation is to 45 degrees.

Transparent triangle

This example draws a triangle, creates a black stroke identical to the *Simple point with stroke* example, and sets the fill of the triangle to 20% opacity (mostly transparent).

Code

```
1  * {
2    mark: symbol(triangle);
3    mark-size: 12;
4  }
5
6  :mark {
7    fill: #009900;
8    fill-opacity: 0.2;
9    stroke: black;
```

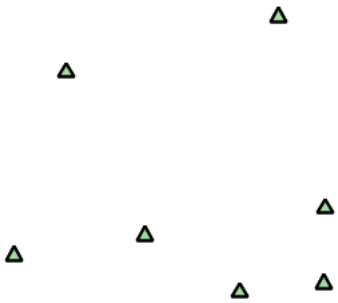


Figure 20.5: *Transparent triangle*

```
10      stroke-width : 2px;
11    }
```

Details In this example, **line 2** once again sets the shape, in this case to a triangle, where **line 3** sets the mark size to 12 pixels. **Line 6** sets the fill color to a dark green (`#009900`) and **line 7** sets the opacity to 0.2 (20% opaque). An opacity value of 1 means that the shape is drawn 100% opaque, while an opacity value of 0 means that the shape is drawn 0% opaque, or completely transparent. The value of 0.2 (20% opaque) means that the fill of the points partially takes on the color and style of whatever is drawn beneath it. In this example, since the background is white, the dark green looks lighter. Were the points imposed on a dark background, the resulting color would be darker. **Line 8** set the stroke color to black and width to 2 pixels.

Point as graphic

This example styles each point as a graphic instead of as a simple shape.

Figure 20.6: *Point as graphic*

Code

```
1  * {
2    mark: url(smileyface.png);
3    mark-mime: "image/png";
4  }
```

Details This style uses a graphic instead of a simple shape to render the points. **Line 2** sets the path and file name of the graphic, while **line 3** indicates the format (MIME type) of the graphic (image/png). In this example, the graphic is contained in the same directory as the SLD, so no path information is necessary, although a full URL could be used if desired.

Figure 20.7: *Graphic used for points*

Point with default label

This example shows a text label on the *Simple point* that displays the "name" attribute of the point. This is how a label will be displayed in the absence of any other customization.

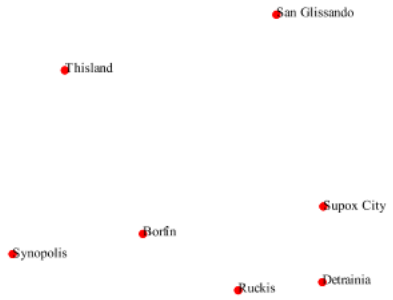


Figure 20.8: *Point with default label*

Code

```
1  * {
2    mark: symbol(circle);
3    mark-size: 6px;
4    label: [name];
5    font-fill: black;
6  }
7
8  :mark {
9    fill: red;
10 }
```

Details This style is quite similar to the *Simple point*, but two new properties have been added to specify the labelling options. **Line 4** indicates that the label contents come from the "name" attribute (anything in

square brackets is a CQL expression, the attribute name being the simplest case) while **Line 5** sets the label color to black.

Point with styled label

This example improves the label style from the *Point with default label* example by centering the label above the point and providing a different font name and size.

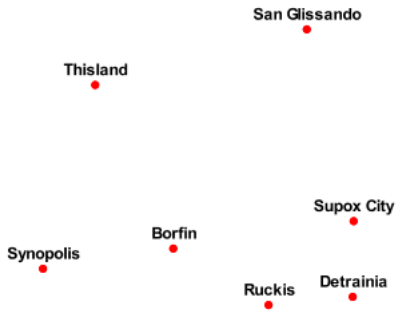


Figure 20.9: *Point with styled label*

Code

```
1  * {
2    mark: symbol(circle);
3    mark-size: 6px;
4    label: [name];
5    font-fill: black;
6    font-family: Arial;
7    font-size: 12;
8    font-weight: bold;
9    label-anchor: 0.5 0;
10   label-offset: 0 5;
11 }
12
13 :mark {
14   fill: red;
15 }
```

Details This example expands on *Point with default label* and specifies the font attributes, in particular, the text is Aria, bold, 12px wide. Moreover, the label is moved on top of the point, by specifying an anchor of `0.5 0`, which sets the point to be centered (0.5) horizontally axis and bottom aligned (0.0) vertically with the label, and an offset which moves the label 5 pixels up vertically.

The result is a centered bold label placed slightly above each point.

Point with rotated label

This example builds on the previous example, *Point with styled label*, by rotating the label by 45 degrees, positioning the labels farther away from the points, and changing the color of the label to purple.

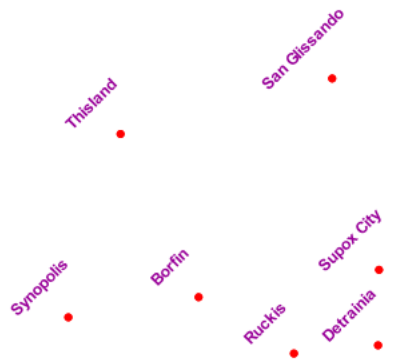


Figure 20.10: *Point with rotated label*

Code

```
1  * {
2    mark: symbol(circle);
3    mark-size: 6px;
4    label: [name];
5    font-fill: #990099;
6    font-family: Arial;
7    font-size: 12;
8    font-weight: bold;
9    label-anchor: 0.5 0;
10   label-offset: 0 25;
11   label-rotation: -45;
12 }
13
14 :mark {
15   fill: red;
16 }
```

Details This example is similar to the *Point with styled label*, but there are three important differences. **Line ** specifies 25 pixels of vertical displacement. **Line 11** specifies a rotation of "-45" or 45 degrees counter-clockwise. (Rotation values increase clockwise, which is why the value is negative.) Finally, **line 5** sets the font color to be a shade of purple (`#990099`).

Note that the displacement takes effect before the rotation during rendering, so in this example, the 25 pixel vertical displacement is itself rotated 45 degrees.

Attribute-based point

This example alters the size of the symbol based on the value of the population ("pop") attribute.

Code

```
1  * {
2    mark: symbol(circle);
3  }
4
5  :mark {
6    fill: #0033CC;
```

Figure 20.11: *Attribute-based point*

```
7   }
8
9   [pop < 50000] {
10    mark-size: 8;
11  }
12
13  [pop >= 50000] [pop < 100000] {
14    mark-size: 12;
15  }
16
17  [pop >= 100000] {
18    mark-size: 16;
19  }
```

Details

Note: Refer to the *Example points layer* to see the attributes for this data. This example has eschewed labels in order to simplify the style, but you can refer to the example *Point with styled label* to see which attributes correspond to which points.

This style shows how the basic mark setup (red circle, default size) can be overridden via cascading, changing the size depending on the pop attribute value, with smaller values yielding a smaller circle, and larger values yielding a larger circle.

The three rules are designed as follows:

Rule order	**Rule name**	**Population** ("pop")	**Size**
1	SmallPop	Less than 50,000	8
2	MediumPop	50,000 to 100,000	12
3	LargePop	Greater than 100,000	16

The result of this style is that cities with larger populations have larger points. In particular, the rule at **Line 9** matches all features whose "pop" attribute is less than 50000, the rule at **Line 13** matches all features whose "pop" attribute is between 50000 and 100000 (mind the space between the two predicates, it is equivalent to and AND, if we had used a comma it would have been an OR instead), while the rule at **Line 17** matches all features with more than 100000 inhabitants.

Zoom-based point

This example alters the style of the points at different zoom levels.

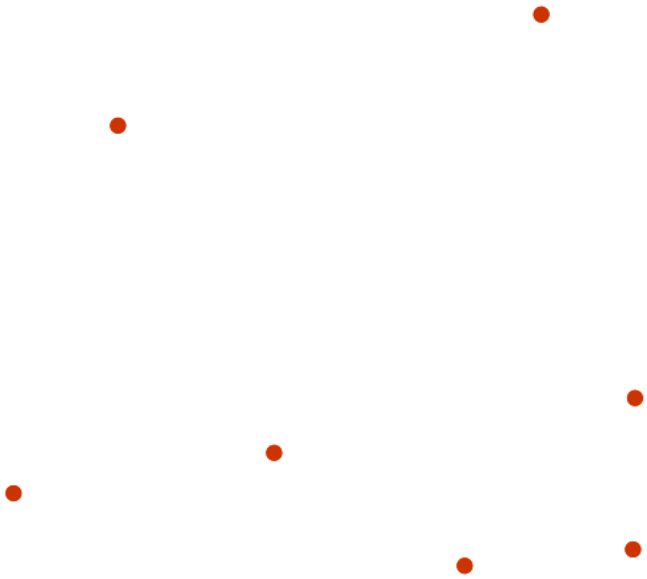


Figure 20.12: *Zoom-based point: Zoomed in*

Figure 20.13: *Zoom-based point: Partially zoomed*

Code

```
1  * {
2    mark: symbol(circle);
3  }
4
5  :mark {
```

Figure 20.14: *Zoom-based point: Zoomed out*

```
6       fill: #CC3300;
7     }
8
9     [@scale < 16000000] {
10      mark-size: 12;
11    }
12
13    [@scale > 16000000] [@scale < 32000000] {
14      mark-size: 8;
15    }
16
17    [@scale > 32000000] {
18      mark-size: 4;
19    }
```

Details It is often desirable to make shapes larger at higher zoom levels when creating a natural-looking map. This example styles the points to vary in size based on the zoom level (or more accurately, scale denominator). Scale denominators refer to the scale of the map. A scale denominator of 10,000 means the map has a scale of 1:10,000 in the units of the map projection.

Note: Determining the appropriate scale denominators (zoom levels) to use is beyond the scope of this example.

This style contains three rules matching the scale. The three rules are designed as follows:

Rule order	Rule name	Scale denominator	Point size
1	Large	1:16,000,000 or less	12
2	Medium	1:16,000,000 to 1:32,000,000	8
3	Small	Greater than 1:32,000,000	4

The order of these rules does not matter since the scales denominated in each rule do not overlap.

The rules use the "@scale" pseudo-attribute, which refers to the current scale denominator, and which can be compared using the '<' and '>' operators only (using any other operator or function will result in errors).

The result of this style is that points are drawn larger as one zooms in and smaller as one zooms out.

Lines

While lines can also seem to be simple shapes, having length but no width, there are many options and tricks for making lines display nicely.

Example lines layer

The `lines layer` used in the examples below contains road information for a fictional country. For reference, the attribute table for the points in this layer is included below.

fid (Feature ID)	**name** (Road name)	**type** (Road class)
line.1	Latway	highway
line.2	Crescent Avenue	secondary
line.3	Forest Avenue	secondary
line.4	Longway	highway
line.5	Saxer Avenue	secondary
line.6	Ridge Avenue	secondary
line.7	Holly Lane	local-road
line.8	Mulberry Street	local-road
line.9	Nathan Lane	local-road
line.10	Central Street	local-road
line.11	Lois Lane	local-road
line.12	Rocky Road	local-road
line.13	Fleet Street	local-road
line.14	Diane Court	local-road
line.15	Cedar Trail	local-road
line.16	Victory Road	local-road
line.17	Highland Road	local-road
line.18	Easy Street	local-road
line.19	Hill Street	local-road
line.20	Country Road	local-road
line.21	Main Street	local-road
line.22	Jani Lane	local-road
line.23	Shinbone Alley	local-road
line.24	State Street	local-road
line.25	River Road	local-road

`Download the lines shapefile`

Simple line

This example specifies lines be colored black with a thickness of 3 pixels.

Code

```
1  * {
2    stroke: black;
3    stroke-width: 3px;
4  }
```

Details The only rule asks for a black stroke (this attribute is mandatory to get strokes to actually show up), 3 pixels wide.

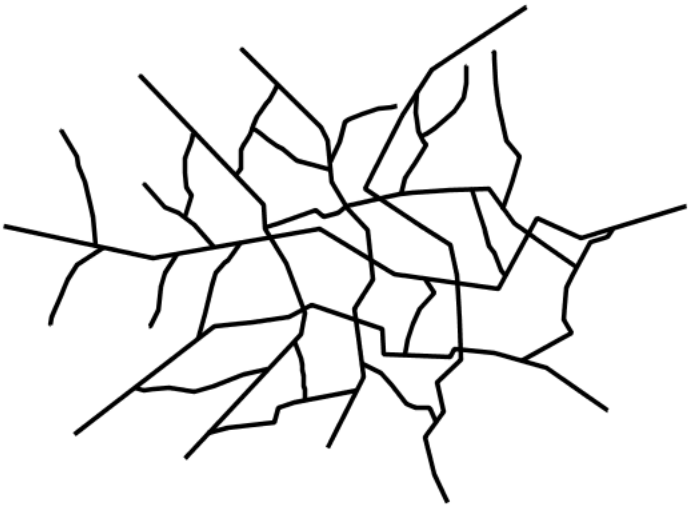


Figure 20.15: *Simple line*

Line with border

This example shows how to draw lines with borders (sometimes called "cased lines"). In this case the lines are drawn with a 3 pixel blue center and a 1 pixel wide gray border.

Code

```
1  * {
2    stroke: #333333, #6699FF;
3    stroke-width: 5px, 3px;
4    stroke-linecap: round;
5    z-index: 0, 1;
6  }
```

Details Lines in CSS have no notion of a "fill", only "stroke". Thus, unlike points or polygons, it is not possible to style the "edge" of the line geometry. It is, however, possible to achieve this effect by drawing each line twice: once with a certain width and again with a slightly smaller width. This gives the illusion of fill and stroke by obscuring the larger lines everywhere except along the edges of the smaller lines.

The style uses the "multi-valued properties" CSS support by specifying two strokes and two stroke-widths. This causes each feature to be painted twice, first with a dark gray (`#333333`) line 5 pixels wide, and then a thinner blue (`#6699FF`) line 3 pixels wide.

Since every line is drawn twice, the order of the rendering is *very* important. Without the z-index indication, each feature would first draw the gray stroke and then the blue one, and then the rendering engine would move to the next feature, and so on. This would result in ugly overlaps when lines do cross. By using the z-index property (**Line 3**) instead, all gray lines will be painted first, and then all blue lines will painted on top, thus making sure the blue lines visually connect.

The "stroke-linecap" property is the only one having a single value, this is because the value is the same for both the gray and blue line.

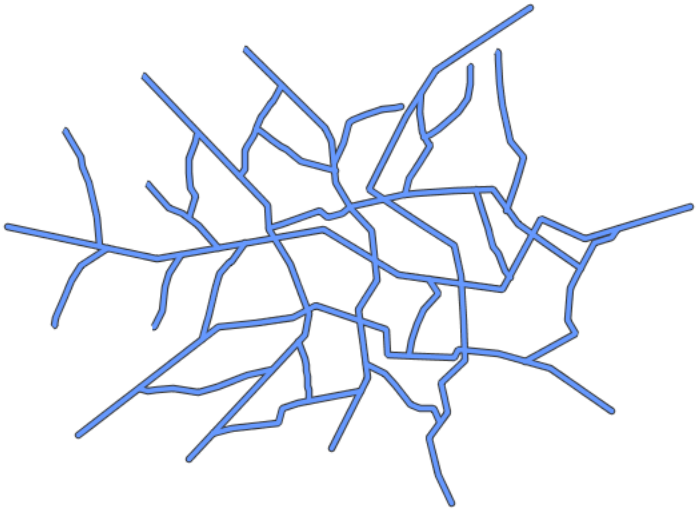


Figure 20.16: *Line with border*

The result is a 3 pixel blue line with a 1 pixel gray border, since the 5 pixel gray line will display 1 pixel on each side of the 3 pixel blue line.

Dashed line

This example alters the *Simple line* to create a dashed line consisting of 5 pixels of drawn line alternating with 2 pixels of blank space.

Code

```
1  * {
2    stroke: blue;
3    stroke-width: 3px;
4    stroke-dasharray: 5 2;
5  }
```

Details In this example the we create a blue line, 3 pixels wide, and specify a dash array with value "5 2", which creates a repeating pattern of 5 pixels of drawn line, followed by 2 pixels of omitted line.

Railroad (hatching)

This example uses hatching to create a railroad style. Both the line and the hatches are black, with a 2 pixel thickness for the main line and a 1 pixel width for the perpendicular hatches.

Code

```
1  * {
2    stroke: #333333, symbol("shape://vertline");
3    stroke-width: 3px;
4  }
```

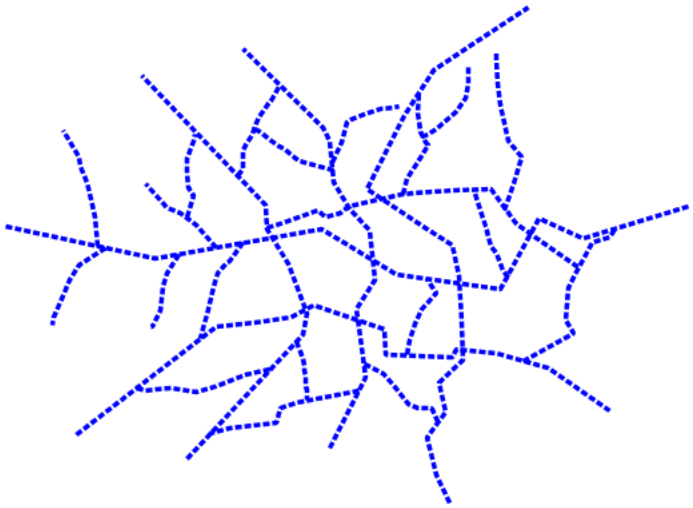


Figure 20.17: *Dashed line*

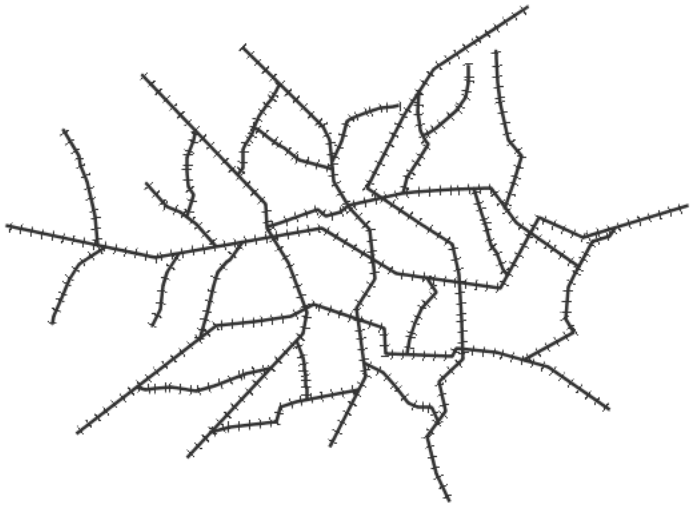


Figure 20.18: *Railroad (hatching)*

```
5
6   :nth-stroke(2) {
7     size: 12;
8     stroke: #333333;
9     stroke-width: 1px;
10  }
```

Details In this example a multi-valued stroke is used: the fist value makes the renderer paint a dark gray line (3 pixels wide, according to the "stroke-width" attribute), whilst the second value makes the line be painted by repeating the "shape://vertline" symbol over and over, creating the hatching effect.

In order to specify how the symbol itself should be painted, the ":nth-stroke(2)" pseudo-selector is used at **Line 6** to specify the options for the repeated symbol: in particular with are instructing the renderer to create a 12px wide symbol, with a dark gray stroke 1 pixel wide.

Spaced graphic symbols

This example uses a graphic stroke along with dash arrays to create a "dot and space" line type. Adding the dash array specification allows to control the amount of space between one symbol and the next one. Without using the dash array the lines would be densely populated with dots, each one touching the previous one.

Figure 20.19: *Spaced symbols along a line*

Code

```
1   * {
2     stroke: symbol(circle);
3     stroke-dasharray: 4 6;
4   }
5
6   :stroke {
7     size: 4;
8     fill: #666666;
9     stroke: #333333;
10    stroke-width: 1px;
11  }
```

Details This example, like others before, uses `symbol(circle)` to place a graphic symbol along a line.

The symbol details are specified in the rule at **Line 6** using the ":stroke" pseudo-selector, creating a gray fill circle, 4 pixels wide, with a dark gray outline.

The spacing between symbols is controlled with the `stroke-dasharray` at **line 3**, which specifies 4 pixels of pen-down (just enough to draw the circle) and 6 pixels of pen-up, to provide the spacing.

Alternating symbols with dash offsets

This example shows how to create a complex line style which alternates a dashed line and a graphic symbol. The code builds on features shown in the previous examples:

- `stroke-dasharray` controls pen-down/pen-up behavior to generate dashed lines
- `symbol(...)` places symbols along a line combining the two allows control of symbol spacing

This also shows the usage of a *dash offset*, which controls where rendering starts in the dash array. For example, with a dash array of `5 10` and a dash offset of `7` the renderer starts drawing the pattern 7 pixels from the beginning. It skips the 5 pixels pen-down section and 2 pixels of the pen-up section, then draws the remaining 8 pixels of pen-up, then 5 down, 10 up, and so on.

The example shows how to use these features to create two synchronized sequences of dash arrays, one drawing line segments and the other symbols.

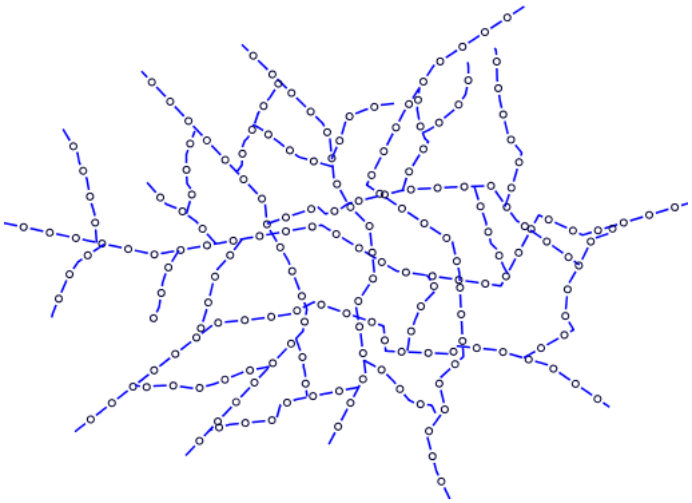


Figure 20.20: *Alternating dash and symbol*

Code

```
1  * {
2    stroke: blue, symbol(circle);
3    stroke-width: 1px;
4    stroke-dasharray: 10 10, 5 15;
5    stroke-dashoffset: 0, 7.5;
6  }
7
8  :nth-stroke(2) {
9    stroke: #000033;
10   stroke-width: 1px;
```

```
    size: 5px;
  }
```

Details

This example uses again multi-valued properties to create two subsequent strokes applied to the same lines.
The first stroke is a solid blue line, 1 pixel wide, with a dash array of "10 10".
The second one instead is a repeated circle, using a dash array of "5 15" and with a dash offset of 7.5. This makes the sequence start with 12.5 pixels of white space, then a circle (which is then centered between the two line segments of the other pattern), then 15 pixels of white space, and so on.

The circle portrayal details are specified using the pseudo selector "nth-stroke(2)" at **line 8**, asking for circles that are 5 pixels wide, not filled, and with a dark blue outline.

Line with default label

This example shows a text label on the simple line. This is how a label will be displayed in the absence of any other customization.

Figure 20.21: *Line with default label*

Code

```
  * {
    stroke: red;
    label: [name];
    font-fill: black;
  }
```

Details This example paints lines with a red stroke, and then adds horizontal black labels at the center of the line, using the "name" attribute to fill the label.

_css_line_

Labels along line with perpendicular offset

This example shows a text label on the simple line, just like the previous example, but will force the label to be parallel to the lines, and will offset them a few pixels away.

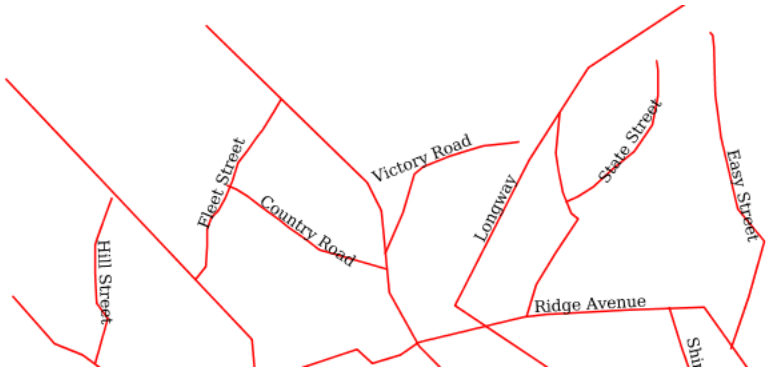


Figure 20.22: *Line with default label*

Code

```
* {
  stroke: red;
  label: [name];
  label-offset: 7px;
  font-fill: black;
}
```

Details This example is line by line identical to the previous one, but it add a new attribute "label-offset", which in the case of lines, when having a single value, is intepreted as a perpendicular offset from the line. The label is painted along a straight line, parallel to the line orientation in the center point of the label.

Label following line

This example renders the text label to follow the contour of the lines.

Code

```
* {
  stroke: red;
  label: [name];
  font-fill: black;
  -gt-label-follow-line: true;
}
```

Figure 20.23: *Label following line*

Details As the *Line with default label* example showed, the default label behavior isn't optimal.

This example is similar to the *Line with default label* example with the exception of **line 5** where the "-gt-label-follow-line" option is specified, which forces the labels to strickly follow the line.

Not all labels are visible partly because of conflict resolution, and partly because the renderer cannot find a line segment long and "straight" enough to paint the label (labels are not painted over sharp turns by default).

Optimized label placement

This example optimizes label placement for lines such that the maximum number of labels are displayed.

Code

```
1  * {
2    stroke: red;
3    label: [name];
4    font-fill: black;
5    -gt-label-follow-line: true;
6    -gt-label-max-angle-delta: 90;
7    -gt-label-max-displacement: 400;
8    -gt-label-repeat: 150;
9  }
```

Details This example is similar to the previous example, *Label following line*. The only differences are contained in **lines 6-8**. **Line 6** sets the maximum angle that the label will follow. This sets the label to never bend more than 90 degrees to prevent the label from becoming illegible due to a pronounced curve or angle. **Line 7** sets the maximum displacement of the label to be 400 pixels. In order to resolve conflicts with overlapping labels, GeoServer will attempt to move the labels such that they are no longer overlapping. This value sets how far the label can be moved relative to its original placement. Finally, **line 8** sets the

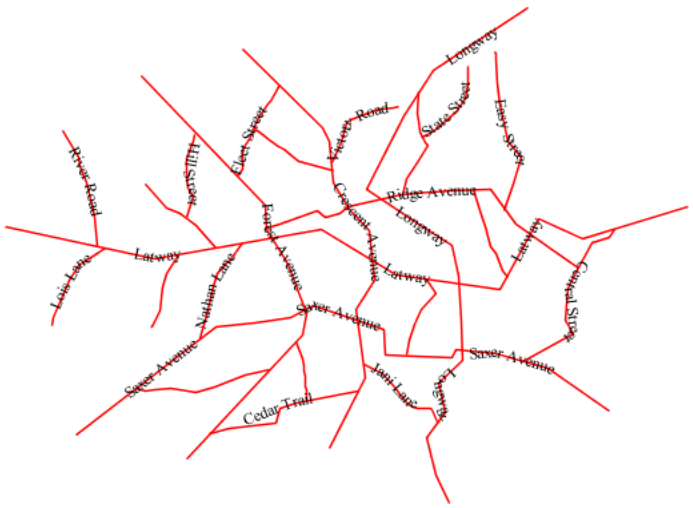


Figure 20.24: *Optimized label*

labels to be repeated every 150 pixels. A feature will typically receive only one label, but this can cause confusion for long lines. Setting the label to repeat ensures that the line is always labeled locally.

Optimized and styled label

This example improves the style of the labels from the *Optimized label placement* example.

Figure 20.25: *Optimized and styled label*

Code

```
* {
  stroke: red;
  label: [name];
  font-family: Arial;
  font-weight: bold;
  font-fill: black;
  font-size: 10;
```

```
    halo-color: white;
    halo-radius: 1;
    -gt-label-follow-line: true;
    -gt-label-max-angle-delta: 90;
    -gt-label-max-displacement: 400;
    -gt-label-repeat: 150;
  }
```

Details This example is similar to the *Optimized label placement*. The only differences are:

- The font family and weight have been specified
- In order to make the labels easier to read, a white "halo" has been added. The halo draws a thin 1 pixel white border around the text, making it stand out from the background.

Attribute-based line

This example styles the lines differently based on the "type" (Road class) attribute.

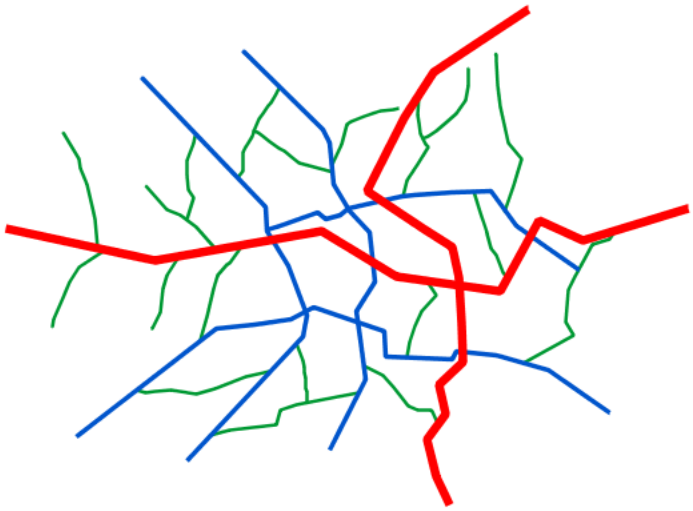


Figure 20.26: *Attribute-based line*

Code

```
[type = 'local-road'] {
  stroke: #009933;
  stroke-width: 2;
  z-index: 0;
}

[type = 'secondary'] {
  stroke: #0055CC;
  stroke-width: 3;
  z-index: 1;
```

```
11  }
12
13  [type = 'highway'] {
14    stroke: #FF0000;
15    stroke-width: 6;
16    z-index: 2;
17  }
```

Details

Note: Refer to the *Example lines layer* to see the attributes for the layer. This example has eschewed labels in order to simplify the style, but you can refer to the example *Optimized and styled label* to see which attributes correspond to which points.

There are three types of road classes in our fictional country, ranging from back roads to high-speed freeways: "highway", "secondary", and "local-road". In order to make sure the roads are rendered in the proper order of importance, a "z-index" attribute has been placed in each rule.

The three rules are designed as follows:

Rule order	Rule name / type	Color	Size
1	local-road	`#009933` (green)	2
2	secondary	`#0055CC` (blue)	3
3	highway	`#FF0000` (red)	6

Lines 1-5 comprise the first rule, the filter matches all roads that the "type" attribute has a value of "local-road". If this condition is true for a particular line, the rule renders it dark green, 2 pixels wide. All these lines are rendered first, and thus sit at the bottom of the final map.

Lines 7-11 match the "secondary" roads, painting them dark blue, 3 pixels wide. Given the "z-index" is 1, they are rendered after the local roads, but below the highways.

Lines 13-17 match the "highway" roads, painting them red 6 pixels wide. These roads are pained last, thus, on top of all others.

Zoom-based line

This example alters the *Simple line* style at different zoom levels.

Code

```
1   * {
2     stroke: #009933;
3   }
4
5   [@scale < 180000000] {
6     stroke-width: 6;
7   }
8
9   [@scale > 180000000] [@scale < 360000000] {
10    stroke-width: 4;
11  }
12
13  [@scale > 360000000] {
14    stroke-width: 2;
15  }
```

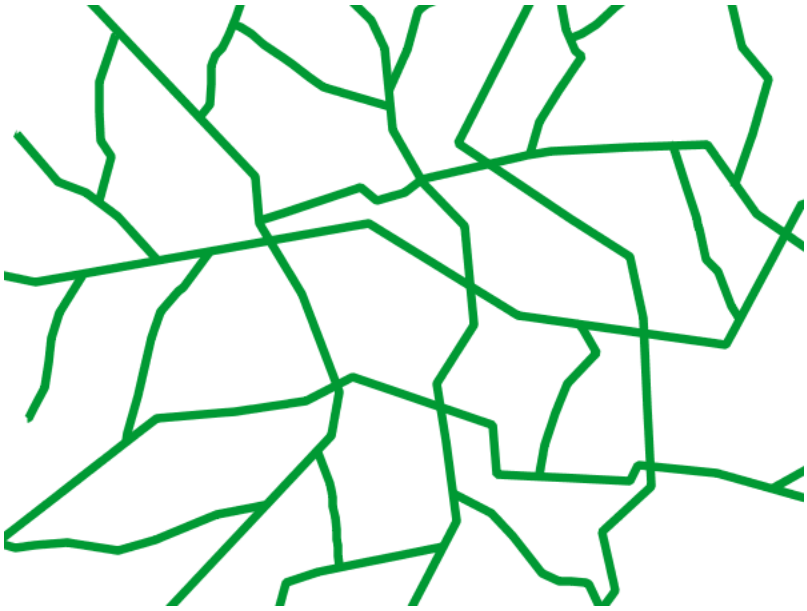


Figure 20.27: *Zoom-based line: Zoomed in*

Figure 20.28: *Zoom-based line: Partially zoomed*

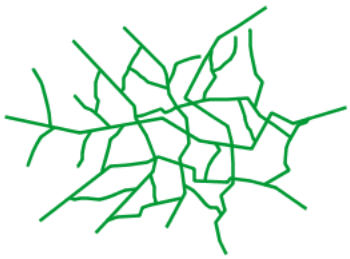


Figure 20.29: *Zoom-based line: Zoomed out*

Details It is often desirable to make shapes larger at higher zoom levels when creating a natural-looking map. This example varies the thickness of the lines according to the zoom level (or more accurately, scale denominator). Scale denominators refer to the scale of the map. A scale denominator of 10,000 means the map has a scale of 1:10,000 in the units of the map projection.

Note: Determining the appropriate scale denominators (zoom levels) to use is beyond the scope of this example.

This style contains three rules. The three rules are designed as follows:

Rule order	Rule name	Scale denominator	Line width
1	Large	1:180,000,000 or less	6
2	Medium	1:180,000,000 to 1:360,000,000	4
3	Small	Greater than 1:360,000,000	2

The order of these rules does not matter since the scales denominated in each rule do not overlap.

The first rule provides the stroke color used at all zoom levels, dark gray, while the other three rules cascade over it applying the different stroke widths based on the current zoom level leveraging the "@scale" pseudo attribute. The "@scale" pseudo attribute can only be compared using the "<" and ">" operators, using any other operator will result in errors.

The result of this style is that lines are drawn with larger widths as one zooms in and smaller widths as one zooms out.

Polygons

Polygons are two dimensional shapes that contain both an outer edge (or "stroke") and an inside (or "fill"). A polygon can be thought of as an irregularly-shaped point and is styled in similar ways to points.

Example polygons layer

The `polygons layer` used below contains county information for a fictional country. For reference, the attribute table for the polygons is included below.

fid (Feature ID)	**name** (County name)	**pop** (Population)
polygon.1	Irony County	412234
polygon.2	Tracker County	235421
polygon.3	Dracula County	135022
polygon.4	Poly County	1567879
polygon.5	Bearing County	201989
polygon.6	Monte Cristo County	152734
polygon.7	Massive County	67123
polygon.8	Rhombus County	198029

`Download the polygons shapefile`

Simple polygon

This example shows a polygon filled in blue.

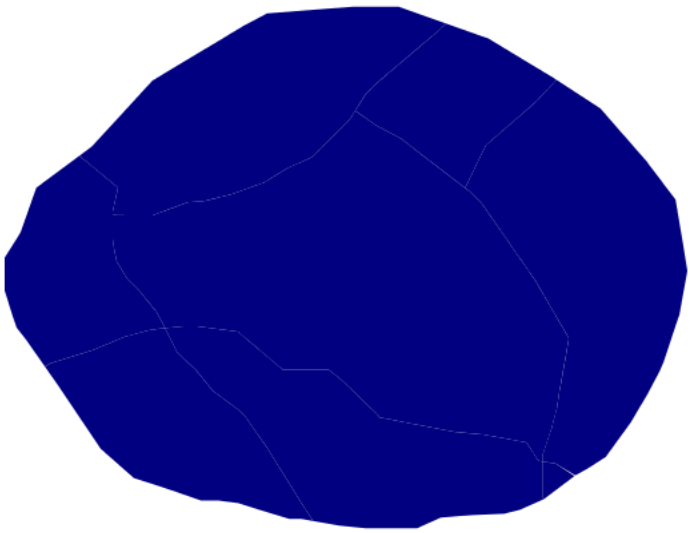


Figure 20.30: *Simple polygon*

Code

```
1  * {
2    fill: #000080;
3  }
```

Details This simple rule applies a dark blue (`#000080`) fill to all the polygons in the dataset.

Note: The light-colored borders around the polygons in the figure are artifacts of the renderer caused by the polygons being adjacent. There is no border in this style.

Simple polygon with stroke

This example adds a 2 pixel white stroke to the *Simple polygon* example.

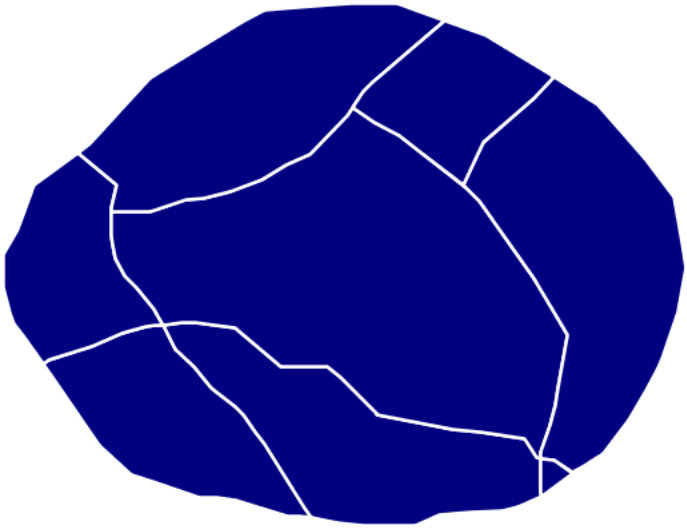


Figure 20.31: *Simple polygon with stroke*

Code

```
* {
  fill: #000080;
  stroke: #FFFFFF;
  stroke-width: 2;
}
```

Details This example is similar to the *Simple polygon* example above, with the addition of the "stroke" and "stroke-width" attributes, that add a white, 2 pixels wide border around each polygon.

Transparent polygon

This example builds on the *Simple polygon with stroke* example and makes the fill partially transparent by setting the opacity to 50%.

Code

```
* {
  fill: #000080;
  fill-opacity: 0.5;
  stroke: #FFFFFF;
  stroke-width: 2;
}
```

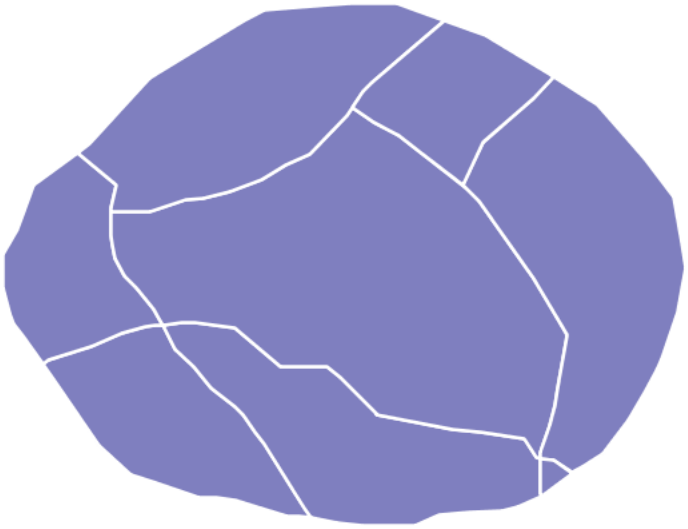


Figure 20.32: *Transparent polygon*

Details This example is similar to the *Simple polygon with stroke* example, save for defining the fill's opacity in **line 3**. The value of 0.5 results in partially transparent fill that is 50% opaque. An opacity value of 1 would draw the fill as 100% opaque, while an opacity value of 0 would result in a completely transparent (0% opaque) fill. In this example, since the background is white, the dark blue looks lighter. Were the points imposed on a dark background, the resulting color would be darker.

Graphic fill

This example fills the polygons with a tiled graphic.

Code

```
1  * {
2    fill: url("colorblocks1.png");
3    fill-mime: 'image/png';
4  }
```

Details This style fills the polygon with a tiled graphic. The graphic is selected providing a url for the fill, which in this case is meant to the relative to the `styles` directory contained within the data directory (an absolute path could have been provided, as well as a internet reference). **Line 3** specifies that the image itself is a png (by default the code assumes jpegs are used and will fail to parse the file unless we specify its mime type). The size of the image is not specified, meaning the native size is going to be used. In case a rescale is desired, the "fill-size" attribute can be used to force a different size.

Hatching fill

This example fills the polygons with a hatching pattern.

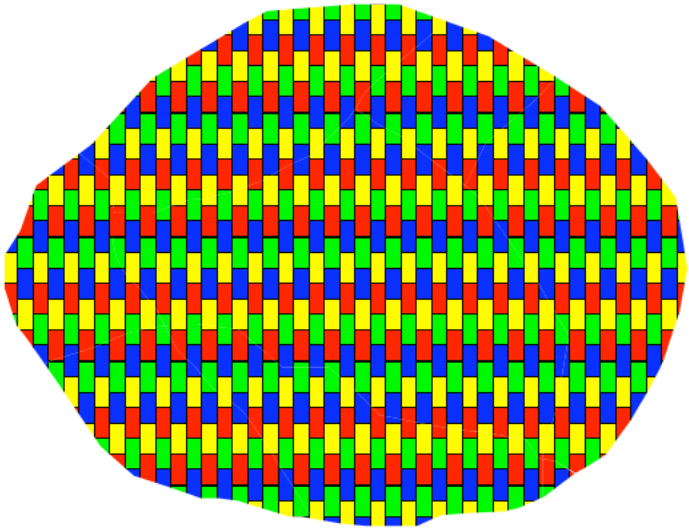


Figure 20.33: *Graphic fill*

Figure 20.34: *Graphic used for fill*

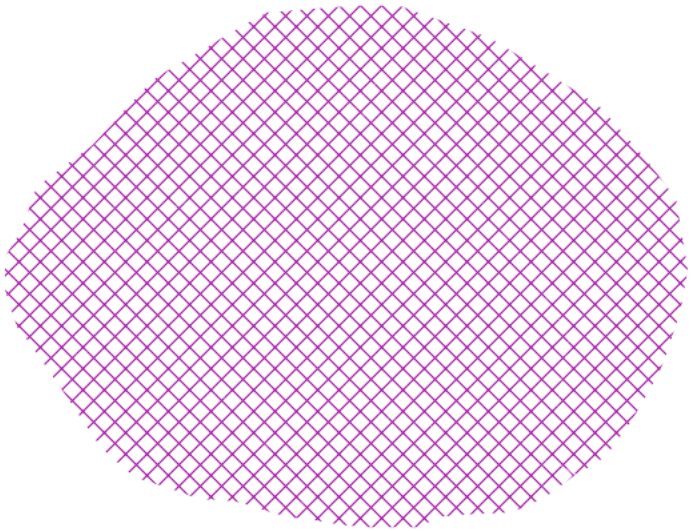


Figure 20.35: *Hatching fill*

Code

```
* {
  fill: symbol("shape://times");
}

:nth-fill(1) {
  size: 16;
  stroke: #990099;
  stroke-width: 1px;
}
```

Details In this example the fill is specified to be the "shape://times" symbol, which is going to be tiled creating a cross-hatch effect.

The details of the hatch are specified at **line 5***, where the pseudo-selector ":nth-fill(1)" is used to match the contents of the first fill, and specify that we want a symbol large 16 pixels (the larger the symbol, the coarser the cross hatch will be), and painted with a 1 pixel wide purple stroke.

Polygon with default label

This example shows a text label on the polygon. In the absence of any other customization, this is how a label will be displayed.

Figure 20.36: *Polygon with default label*

Code

```
* {
  fill: #40FF40;
  stroke: white;
  stroke-width: 2;
  label: [name];
```

```
6     font-fill: black;
7   }
```

Details The single rule in the CSS applies to all feature: first it fills all polygons a light green with white outline, and thn applies the "name" attribute as the label, using the default font (Times), with black color and default font size (10 px).

Label halo

This example alters the look of the *Polygon with default label* by adding a white halo to the label.

Figure 20.37: *Label halo*

Code

```
1   * {
2     fill: #40FF40;
3     stroke: white;
4     stroke-width: 2;
5     label: [name];
6     font-fill: black;
7     halo-color: white;
8     halo-radius: 3;
9   }
```

Details This example builds on *Polygon with default label*, with the addition of a halo around the labels on **lines 7-8**. A halo creates a color buffer around the label to improve label legibility. **Line 9** sets the radius of the halo, extending the halo 3 pixels around the edge of the label, and **line 8** sets the color of the halo to white. Since halos are most useful when set to a sharp contrast relative to the text color, this example uses a white halo around black text to ensure optimum readability.

Polygon with styled label

This example improves the label style from the *Polygon with default label* example by centering the label on the polygon, specifying a different font name and size, and setting additional label placement optimizations.

Figure 20.38: *Polygon with styled label*

Code

```
* {
  fill: #40FF40;
  stroke: white;
  stroke-width: 2;
  label: [name];
  font-family: Arial;
  font-size: 11px;
  font-style: normal;
  font-weight: bold;
  font-fill: black;
  label-anchor: 0.5 0.5;
  -gt-label-auto-wrap: 60;
  -gt-label-max-displacement: 150;
}
```

Details This example is similar to the *Polygon with default label* example, with additional styling options for the labels.

The font is setup to be Arial, 11 pixels, "normal" (as opposed to "italic") and bold.

The "label-anchor" affects where the label is placed relative to the centroid of the polygon, centering the label by positioning it 50% (or 0.5) of the way horizontally along the centroid of the polygon, as well as vertically in exactly the same way.

Finally, there are two added touches for label placement optimization: The "gt-label-auto-wrap" attribute ensures that long labels are split across multiple lines by setting line wrapping on the labels to 60 pixels, whilst the "-gt-label-max-displacement" allows the label to be displaced by up to 150 pixels. This ensures that labels are compacted and less likely to spill over polygon boundaries. Notice little Massive County in the corner, whose label is now displayed.

Attribute-based polygon

This example styles the polygons differently based on the "pop" (Population) attribute.

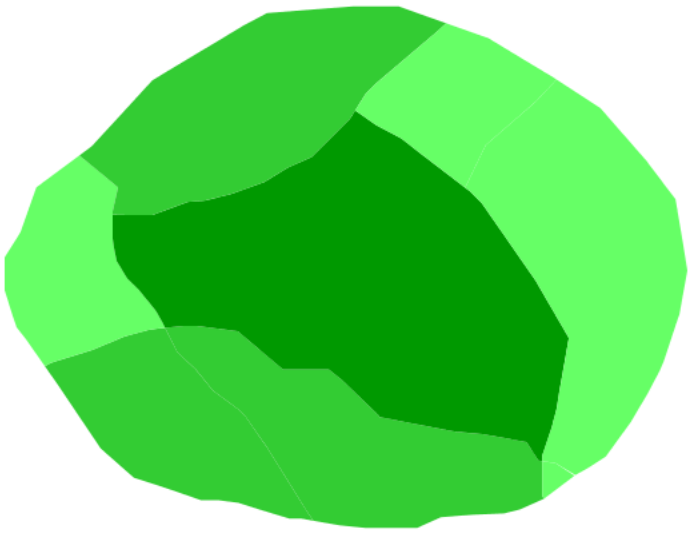


Figure 20.39: *Attribute-based polygon*

Code

```
[parseLong(pop) < 200000] {
  fill: #66FF66;
}

[parseLong(pop) >= 200000] [parseLong(pop) < 500000] {
  fill: #33CC33;
}

[parseLong(pop) >= 500000] {
  fill: #009900;
}
```

Details

Note: Refer to the *Example polygons layer* to see the attributes for the layer. This example has eschewed labels in order to simplify the style, but you can refer to the example *Polygon with styled label* to see which attributes correspond to which polygons.

Each polygon in our fictional country has a population that is represented by the population ("pop") attribute. This style contains three rules that alter the fill based on the value of "pop" attribute, with smaller

values yielding a lighter color and larger values yielding a darker color.

The three rules are designed as follows:

Rule order	Rule name	Population ("pop")	Color
1	SmallPop	Less than 200,000	#66FF66
2	MediumPop	200,000 to 500,000	#33CC33
3	LargePop	Greater than 500,000	#009900

The order of the rules does not matter in this case, since each shape is only rendered by a single rule.

The first rule fills light green all polygons whose "pop" attribute is below 200,000, the second paints medium green all poygons whose "pop" attribute is between 200,000 and 500,000, while the third rule paints dark green the remaining polygons.

What's interesting in the filters is the use of the "parseLong" filter function: this function is necessary because the "pop" attribute is a string, leaving it as is we would have a string comparison, whilst the function turns it into a number, ensuring proper numeric comparisons instead.

Zoom-based polygon

This example alters the style of the polygon at different zoom levels.

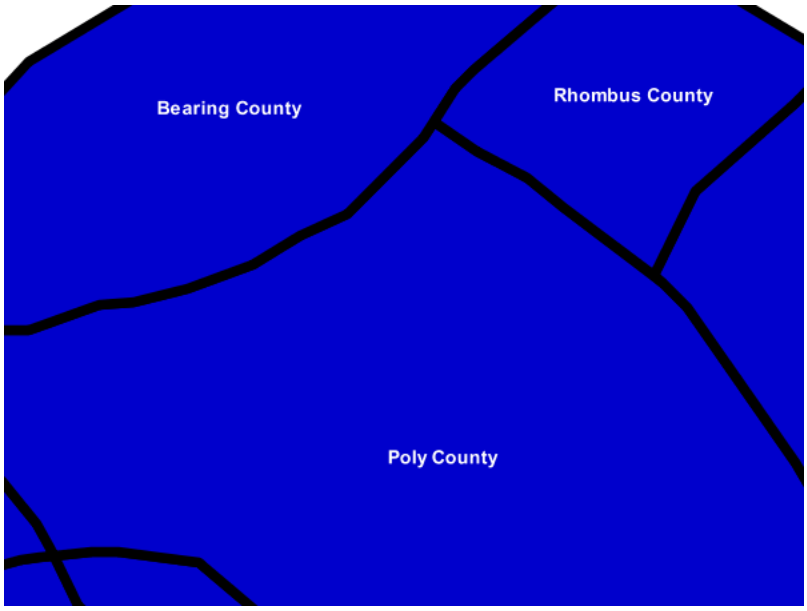


Figure 20.40: *Zoom-based polygon: Zoomed in*

Code

```
* {
  fill: #0000CC;
  stroke: black;
}

[@scale < 100000000] {
   stroke-width: 7;
   label: [name];
```

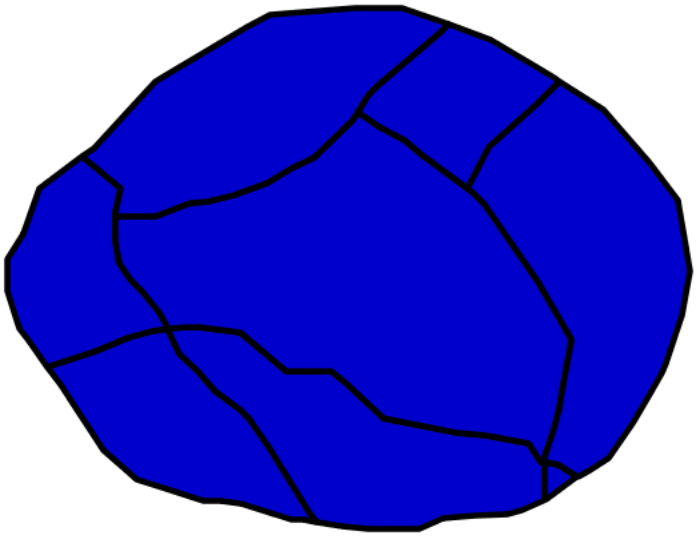


Figure 20.41: *Zoom-based polygon: Partially zoomed*

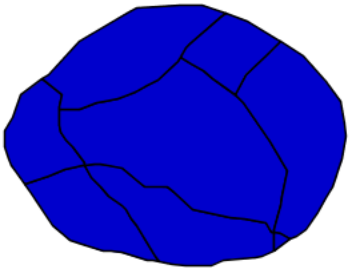


Figure 20.42: *Zoom-based polygon: Zoomed out*

```
9       label-anchor: 0.5 0.5;
10      font-fill: white;
11      font-family: Arial;
12      font-size: 14;
13      font-weight: bold;
14   }
15
16   [@scale > 100000000] [@scale < 200000000] {
17      stroke-width: 4;
18   }
19
20   [@scale > 200000000] {
21      stroke-width: 1;
22   }
```

Details It is often desirable to make shapes larger at higher zoom levels when creating a natural-looking map. This example varies the thickness of the lines according to the zoom level. Polygons already do this by nature of being two dimensional, but another way to adjust styling of polygons based on zoom level is to adjust the thickness of the stroke (to be larger as the map is zoomed in) or to limit labels to only certain zoom levels. This is ensures that the size and quantity of strokes and labels remains legible and doesn't overshadow the polygons themselves.

Zoom levels (or more accurately, scale denominators) refer to the scale of the map. A scale denominator of 10,000 means the map has a scale of 1:10,000 in the units of the map projection.

Note: Determining the appropriate scale denominators (zoom levels) to use is beyond the scope of this example.

This style contains three rules, defined as follows:

Rule order	Rule name	Scale denominator	Stroke width	Label display?
1	Large	1:100,000,000 or less	7	Yes
2	Medium	1:100,000,000 to 1:200,000,000	4	No
3	Small	Greater than 1:200,000,000	2	No

The first rule (**lines 1-4**) defines the attributes that are not scale dependent: dark blue fill, black outline.

The second (**lines 6-14**) rule provides specific overrides for the higher zoom levels, asking for a large stroke (7 pixels) and a label, which is only visible at this zoom level. The label is white, bold, Arial 14 pixels, its contents are coming form the "name" attribute.

The third rule (**lines 16-18**) specifies a stroke width of 4 pixels for medium zoom levels, whilst for low zoom levels the stroke width is set to 1 pixel by the last rule (**lines 20-22**).

The resulting style produces a polygon stroke that gets larger as one zooms in and labels that only display when zoomed in to a sufficient level.

Rasters

Rasters are geographic data displayed in a grid. They are similar to image files such as PNG files, except that instead of each point containing visual information, each point contains geographic information in numerical form. Rasters can be thought of as a georeferenced table of numerical values.

One example of a raster is a Digital Elevation Model (DEM) layer, which has elevation data encoded numerically at each georeferenced data point.

Example raster

The `raster layer` that is used in the examples below contains elevation data for a fictional world. The data is stored in EPSG:4326 (longitude/latitude) and has a data range from 70 to 256. If rendered in grayscale, where minimum values are colored black and maximum values are colored white, the raster would look like this:

Figure 20.43: *Raster file as rendered in grayscale*

`Download the raster file`

Two-color gradient

This example shows a two-color style with green at lower elevations and brown at higher elevations.

Figure 20.44: *Two-color gradient*

Code

```
* {
  raster-channels: auto;
  raster-color-map:
                    color-map-entry(#008000, 70)
                    color-map-entry(#663333, 256);
}
```

Details There is a single rule which applies a color map to the raster data.

The "raster-channels" attribute activates raster symbolization, the "auto" value is indicates that we are going to use the default choice of bands to symbolize the output (either gray or RBG/RGBA depending on the input data). There is also the possibility of providing a band name or a list of band names in case we want to choose specific bands out of a multiband input, e.g., "1" or "1 3 7".

The "raster-color-map" attribute builds a smooth gradient between two colors corresponding to two elevation values. Each "color-map-entry" represents one entry or anchor in the gradient:

- The first argument is the color
- The second argument is the value at which we anchor the color
- An optional third argument could specify the opacity of the pixels, as a value between 0 (fully transparent) and 1 (fully opaque). The default, when not specified, is 1, fully opaque.

Line 4 sets the lower value of 70, which is styled a opaque dark green (`#008000`), and **line 5** sets the upper value of 256, which is styled a opaque dark brown (`#663333`). All data values in between these two quantities will be linearly interpolated: a value of 163 (the midpoint between 70 and 256) will be colored as the midpoint between the two colors (in this case approximately `#335717`, a muddy green).

Transparent gradient

This example creates the same two-color gradient as in the *Two-color gradient* as in the example above but makes the entire layer mostly transparent by setting a 30% opacity.

Figure 20.45: *Transparent gradient*

Code

```
1  * {
2    raster-channels: auto;
3    raster-opacity: 0.3;
4    raster-color-map: color-map-entry(#008000, 70)
5                      color-map-entry(#663333, 256);
6  }
```

Details This example is similar to the *Two-color gradient* example save for the addition of **line 3**, which sets the opacity of the layer to 0.3 (or 30% opaque). An opacity value of 1 means that the shape is drawn 100% opaque, while an opacity value of 0 means that the shape is rendered as completely transparent. The value of 0.3 means that the the raster partially takes on the color and style of whatever is drawn beneath it. Since the background is white in this example, the colors generated from the "raster-color-map" look lighter, but were the raster imposed on a dark background the resulting colors would be darker.

Brightness and contrast

This example normalizes the color output and then increases the brightness by a factor of 2.

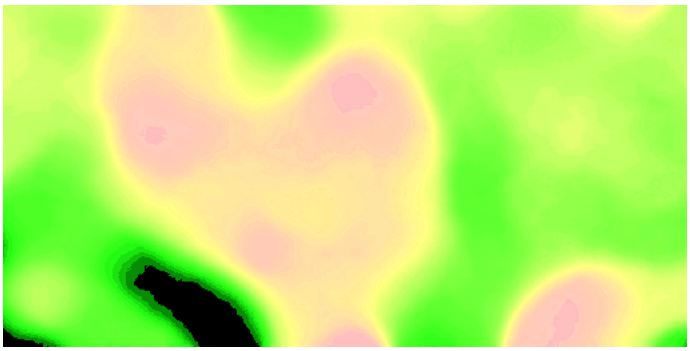


Figure 20.46: *Brightness and contrast*

Code

```
1       * {
2         raster-channels: auto;
3         raster-contrast-enhancement: normalize;
4         raster-gamma: 0.5;
5         raster-color-map: color-map-entry(#008000, 70)
6                           color-map-entry(#663333, 256);
7       }
```

Details This example is similar to the *Two-color gradient*, save for the addition of the contrast enhancement and gamma attributes on **lines 3-4**. **Line 3** normalizes the output by increasing the contrast to its maximum extent. **Line 4** then adjusts the brightness by a factor of 0.5. Since values less than 1 make the output brighter, a value of 0.5 makes the output twice as bright.

Three-color gradient

This example creates a three-color gradient in primary colors. In addition, we want to avoid displaying data outside of the chosen range, leading some data not to be rendered at all.

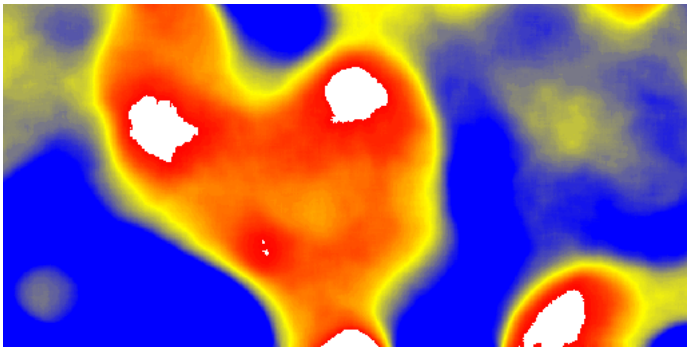


Figure 20.47: *Three-color gradient*

Code

```
1       * {
2         raster-channels: auto;
3         raster-color-map:
4                           color-map-entry(black, 150, 0)
5                           color-map-entry(blue, 150)
6                           color-map-entry(yellow, 200)
7                           color-map-entry(red, 250)
8                           color-map-entry(black, 250, 0)
9       }
```

Details This example creates a three-color gradient, with two extra rules to make ranges of color disappear. The color map behavior is such that any value below the lowest entry gets the same color as that entry, and any value above the last entry gets the same color as the last entry, while everything in between is linearly interpolated (all values must be provided from lower to higher). **Line 4** associates value 150 and below with a transparent color (0 opacity, that is, fully transparent), and so does **line 8**, which makes transparent every value above 250. The lines in the middle create a gradient going from blue, to yellow, to red.

Alpha channel

This example creates an "alpha channel" effect such that higher values are increasingly transparent.

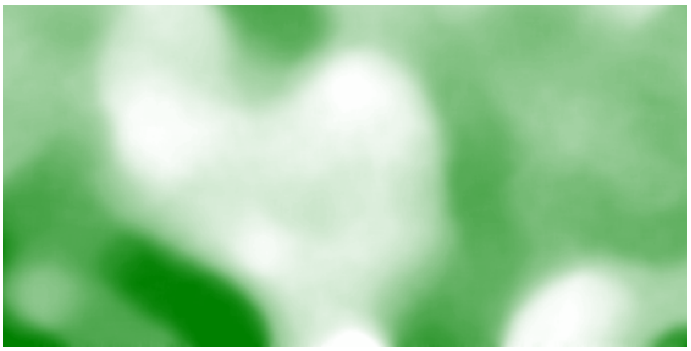


Figure 20.48: *Alpha channel*

Code

```
1       * {
2         raster-channels: auto;
3         raster-color-map: color-map-entry(#008000, 70)
4                           color-map-entry(#663333, 256, 0);
5       }
```

Details An alpha channel is another way of referring to variable transparency. Much like how a gradient maps values to colors, each entry in a "raster-color-map" can have a value for opacity (with the default being 1.0 or completely opaque).

In this example, there is a "raster-color-map" with two entries: **line 3** specifies the lower bound of 70 be colored dark green (`#008000`), while **line 4** specifies the upper bound of 256 also be colored dark green but with an opacity value of 0. This means that values of 256 will be rendered at 0% opacity (entirely

transparent). Just like the gradient color, the opacity is also linearly interpolated such that a value of 163 (the midpoint between 70 and 256) is rendered at 50% opacity.

Discrete colors

This example shows a gradient that is not linearly interpolated but instead has values mapped precisely to one of three specific colors.

Figure 20.49: *Discrete colors*

Code

```
1       * {
2         raster-channels: auto;
3         raster-color-map-type: intervals;
4         raster-color-map: color-map-entry(#008000, 150)
5                           color-map-entry(#663333, 256);
6       }
```

Details Sometimes color bands in discrete steps are more appropriate than a color gradient. The "raster-color-map-type: intervals" attribute sets the display to output discrete colors instead of a gradient. The values in each entry correspond to the upper bound for the color band such that colors are mapped to values less than the value of one entry but greater than or equal to the next lower entry. For example, **line 4** colors all values less than 150 to dark green (`#008000`) and **line 5** colors all values less than 256 but greater than or equal to 150 to dark brown (`#663333`).

Many color gradient

This example shows a gradient interpolated across eight different colors.

Code

```
1       * {
2         raster-channels: auto;
3         raster-color-map:
4                 color-map-entry(black, 95)
5                 color-map-entry(blue, 110)
6                 color-map-entry(green, 135)
7                 color-map-entry(red, 160)
8                 color-map-entry(purple, 185)
```

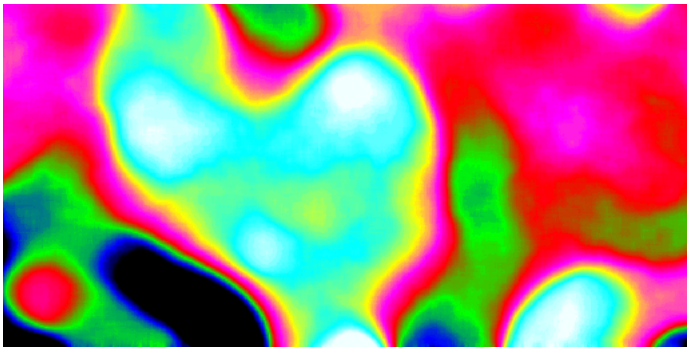


Figure 20.50: *Many color gradient*

```
9                   color-map-entry(yellow, 210)
10                  color-map-entry(cyan, 235)
11                  color-map-entry(white, 256)
12      }
```

Details This example is similar to the previous ones, and creates a color gradient between 8 colors as reported in the following table

Entry number	Value	Color
1	95	Black
2	110	Blue
3	135	Green
4	160	Red
5	185	Purple
6	210	Yellow
7	235	Cyan
8	256	White

20.2.10 CSS Styling Examples

The following pages contain CSS styling examples, groped by topics

Fills with randomized symbols

Starting with GeoServer 2.4.2 it is possible to generate fills by randomly repeating a symbol in the polygons to be filled. Please refer to the *equivalent SLD chapter* for details on the meaning of the various options.

Simple random distribution

Here is an example distributing up to 50 small "slash" symbols in a 100x100 pixel tile (in case of conflicts the symbol will be skipped), enabling random symbol rotation), and setting the seed to "5" to get a distribution different than the default one:

```
* {
  fill: symbol("shape://slash");
  stroke: black;
  -gt-fill-random: grid;
```

```
  -gt-fill-random-seed: 5;
  -gt-fill-random-rotation: free;
  -gt-fill-random-symbol-count: 50;
  -gt-fill-random-tile-size: 100;
}


:fill {
  size: 8;
  stroke: blue;
  stroke-width: 4;
  stroke-linecap: round;
}
```

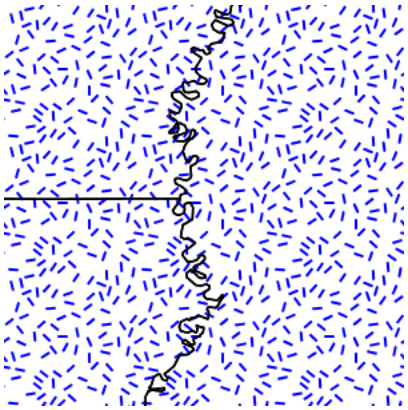


Figure 20.51: *Random distribution of a diagonal line*

Thematic map using point density

Randomized distributions can also be used for thematic mapping, for example, here is the SLD for a version of topp:states that displays the number of inhabitantìs varying the density of a random point distribution:

```
* {
  fill: symbol("circle");
  stroke: black;
  -gt-fill-random: grid;
  -gt-fill-random-tile-size: 100;
}

:fill {
  size: 2;
  fill: darkgray;
}

[PERSONS < 2000000] {
   -gt-fill-random-symbol-count: 50;
}

[PERSONS >= 2000000] [PERSONS < 4000000] {
   -gt-fill-random-symbol-count: 150;
}
```

```
[PERSONS >= 4000000] {
   -gt-fill-random-symbol-count: 500;
}
```

Figure 20.52: *Thematic map via point density approach*

Detecting switch from raster to vector representation in KML

GeoServer 2.4 added a new icon server that KML output uses to make sure the point symbolisers look the same as in a normal WMS call no matter what scale they are looked at.

This may pose some issue when working in the default KML generation mode, where the map is a ground overlay up to a certain scale, and switches to a vector, clickable representation once the number of features in the visualization fall below a certain scale (as controlled by the `KMSCORE` parameter): the end user is not informed "visually" that the switch happened.

There is however a custom enviroment variable, set by the KML generator, that styles can leverage to know whether the KML generation is happening in ground overlay or vector mode.

The following example leverages this function to show a larger point symbol when points become clickable:

```
* {
  mark: symbol("circle");
}

:mark [env('kmlOutputMode') = 'vector'] {
  size: 8;
}

:mark {
  size: 4;
  fill: yellow;
  stroke: black;
}
```

This will result in the following output:

One important bit about the above CSS is that the order of the rules is important. The CSS to SLD translator uses specificity to decide which rule overrides which other one, and the specificity is driven, at the time of writing, only by scale rules and access to attributes. The filter using the `kmlOutputMode` filter is not actually using any feature attribute, so it has the same specificity as the catch all `:mark` rule. Putting it first ensures that it overrides the catch all rule anyways, while putting it second would result in the output size being always 4.

Figure 20.53: *Raster output, points are not yet clickable*

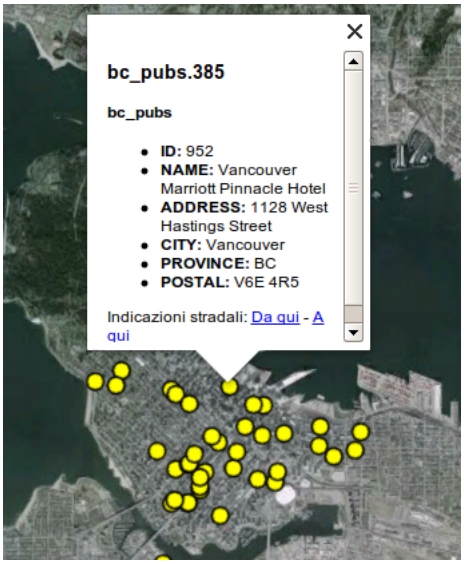


Figure 20.54: *Vector output, points are clickable and painted as larger icons*

Assorted Short Examples

Markers Sized by an Attribute Value

The following produces square markers at each point, but these are sized such that the area of each marker is proprtional to the `REPORTS` attribute. When zoomed in (when there are less points in view) the size of the markers is doubled to make the smaller points more noticable.

```
* {
  mark: symbol(square);
}

[@scale > 1000000] :mark {
  size: [sqrt(REPORTS)];
}

/* So that single-report points can be more easily seen */
[@scale < 1000000] :mark {
  size: [sqrt(REPORTS)*2];
}
```

This example uses the `sqrt` function. There are many functions available for use in CSS and SLD. For more details read - *Filter Function Reference*

Specifying a Geometry Attribute

In some cases, typically when using a database table with multiple geometry columns, it's necessary to specify which geometry to use. For example, let's suppose you have a table containing routes `start` and `end` both containing point geometries. The following CSS will style the start with a triangle mark, and the end with a square.

```
* {
    geometry: [start],          [end];
    mark:     symbol(triangle), symbol(square);
}
```

Generating a Geometry (Geometry Transformations)

Taking the previous example a bit further, we can also perform computations on-the-fly to generate the geometries that will be drawn. Any operation that is available for GeoServer *Geometry transformations in SLD* is also available in CSS styles. To use them, we simply provide a more complex expression in the `geometry` property. For example, we could mark the start and end points of all the paths in a line layer (you can test this example out with any line layer, such as the `sf:streams` layer that is included in GeoServer's default data directory.)

```
* {
    geometry: [startPoint(the_geom)], [endPoint(the_geom)];
    mark:     symbol(triangle),       symbol(square);
}
```

Rendering Different Geometry Types (lines/points) with a Single Style

As one more riff on the geometry examples, we'll show how to render both the original line and the start/endpoints in a single style. This is accomplished by using `stroke-geometry` and `mark-geometry` to specify that different geometry expressions should be used for symbols compared with strokes.

```
* {
    stroke-geometry: [the_geom];
    stroke:          blue;
    mark-geometry: [startPoint(the_geom)], [endPoint(the_geom)];
    mark:          symbol(triangle),       symbol(square);
}
```

20.3 Excel WFS Output Format

The GeoServer Excel plugin adds the ability to output WFS responses in either Excel 97-2003 (`.xls`) or Excel 2007 (`.xlsx`) formats.

20.3.1 Installation

1. Download the Excel plugin for your version of GeoServer from the download page.
2. Unzip the archive into the WEB-INF/lib directory of the GeoServer installation.
3. Restart GeoServer.

20.3.2 Usage

When making a WFS request, set the `outputFormat` to `excel` (for Excel 97-2003) or `excel2007` (for Excel 2007).

20.3.3 Examples

Excel 97-2003 GET: http://localhost:8080/geoserver/wfs?request=GetFeature&version=1.1.0&typeName=topp:states&ou

Excel 2007 GET: http://localhost:8080/geoserver/wfs?request=GetFeature&version=1.1.0&typeName=topp:states&outpu

Excel 97-2003 POST:

```
<wfs:GetFeature service="WFS" version="1.1.0"
  outputFormat="excel"
  xmlns:topp="http://www.openplans.org/topp"
  xmlns:wfs="http://www.opengis.net/wfs"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.opengis.net/wfs
                      http://schemas.opengis.net/wfs/1.1.0/wfs.xsd">
  <wfs:Query typeName="topp:states" />
</wfs:GetFeature>
```

20.3.4 Limitations

Excel 97-2003 files are stored in a binary format and are thus space-efficient, but have inherent size limitations (65,526 rows per sheet; 256 columns per sheet).

Excel 2007 files are XML-based, and have much higher limits (1,048,576 rows per sheet; 16,384 columns per sheet). However, because they are text files Excel 2007 files are usually larger than Excel 97-2003 files.

If the number of rows in a sheet or characters in a cell exceeds the limits of the chosen Excel file format, warning text is inserted to indicate the truncation.

20.4 GeoSearch

20.4.1 GeoSearch Indexing Module

The GeoSearch indexing module adds support to GeoServer for exposing your data to Google's GeoSearch. This makes it so more people can find your data, by searching directly on Google Maps or Google Earth. The format exposed is KML, so other search engines will also be able to crawl it when they are ready - Google is just the first to support it for sure. By default no data is published, but we highly encourage you to if your data can be publicly available, to help grow the wider geospatial web. Publishing is easy, as it is a part of the administration interface. For more information about geosearch see this blog.

20.4.2 How It Works

The GeoSearch module adds a sitemap.xml endpoint in the GeoServer REST API; that is, http://localhost:8080/geoserver/rest/sitemap.xml is your sitemap. By submitting the sitemap through Google's webmaster tools, you can get your map layers to show up in searches on http://maps.google.com/.

20.4.3 Step By Step

A more explicit guide to using the GeoSearch module follows.

1. Load your data as normal.
2. Go to the Layer configuration page in GeoServer's admin console for each layer you would like to expose, and check the 'enable searching' checkbox on the *Publishing* tab.
3. Submit your sitemap.xml using Google's webmaster tools. From your dashboard, pick the domain on which your server lives. In the menu on the left, click on "Sitemaps" and then "Add Sitemap". You are adding a "General Web Sitemap", and provide the URL equivalent http://localhost:8080/geoserver/rest/sitemap.xml .

The reason we are using "General Web Sitemap", as opposed to a "Geo Sitemap", is that sitemap.xml is really a sitemap index that links to a geo sitemap for each layer.

20.4.4 Behind the Scenes

GeoServer already has support for breaking up a dataset into regionated tiles. The information about what features belong in each tile is stored in an H2 database in $GEOSERVER_DATA_DIR/geosearch . We use this information when creating the sitemaps for Google. However, since the hierarchy may not be fully explored by the time a sitemap is submitted, the sitemaps also contain links to tiles deeper in the hierarchy, thereby expanding it. Some of these tiles may be empty, in which case Googlebot will receive a 204 response.

20.4.5 Big datasets

If you are making big datasets available, more than 50 000 individual features, up to 2,000,000, you should consider doing the following. The main burden is to sort the features according to an attribute, so that they are output in order of importance and included in exactly one tile.

1. Use a backend that supports queries, such as Postgis. You can use shp2psql to convert from a Shapefile to a SQL format supported by Postgis. Be sure to specify that you want a GIST (geospatial index) to be created, and provide the SRS. (-I and -s)
2. Make sure your database has a primary index (an auto-incrementing integer is fine) and a spatial index on the geometry column
3. Put an index on the column that you are going to sort the feature by. If you are using the size of the geometry, consider making an auxilliary column that contains the precalculated value and put an index on that. Note that GeoServer always sorts in descending order, so features you consider important should have a high value.
4. In GeoServer's feature type configuration, be sure to use "native-sorting" for the regionating strategy, and your chosen column as the regionating attribute.
5. KML Feature Limit should generally be set to 50. It's a balancing act between too much information per tile (Googlebot prefers document that are less than 1 megabyte) and a big hierarchy that takes long to build.

20.5 Imagemap

HTML ImageMaps have been used for a long time to create interactive images in a light way. Without using Flash, SVG or VML you can simply associate different links or tooltips to different regions of an image. Why can't we use this technique to achieve the same result on a GeoServer map? The idea is to combine a raster map (png, gif, jpeg, ...) with an HTML ImageMap overlay to add links, tooltips, or mouse events behavior to the map.

An example of an ImageMap adding tooltips to a map:

```
<img src="..." usemap="#mymap/>
<map name="mymap">
     <area shape="poly" coords="536,100 535,100 534,101 533,101 532,102"  title="This is a tooltip"/>
     <area shape="poly" coords="518,113 517,114 516,115 515,114"  title="Another tooltip"/>
</map>
```

An example of an ImageMap adding links to a map:

```
<img src="..." usemap="#mymap/>
<map name="mymap">
     <area shape="poly" coords="536,100 535,100 534,101 533,101 532,102"  href="http://www.mylink.com
     <area shape="poly" coords="518,113 517,114 516,115 515,114"  href="http://www.mylink2.com"/>
</map>
```

A more complex example adding interactive behaviour on mouse events:

```
<img src="..." usemap="#mymap/>
<map name="mymap">
     <area shape="poly" coords="536,100 535,100 534,101 533,101 532,102"  onmouseover="onOver('<featu
     <area shape="poly" coords="518,113 517,114 516,115 515,114"   onmouseover="onOver('<featureid>')
</map>
```

To realize this in GeoServer some great community contributors developed an HTMLImageMap GetMap-Producer for GeoServer, able to render an HTMLImageMap in response to a WMS GetMap request.

The GetMapProducer is associated to the text/html mime type. It produces, for each requested layer, a <map>...</map> section containing the geometries of the layer as distinct <area> tags. Due to the limitations in the shape types supported by the <area> tag, a single geometry can be split into multiple ones. This way almost any complex geometry can be rendered transforming it into simpler ones.

To add interactive attributes we use styling. In particular, an SLD Rule containing a TextSymbolizer with a Label definition can be used to define dynamic values for the <area> tags attributes. The Rule name will be used as the attribute name.

As an example, to define a title attribute (associating a tooltip to the geometries of the layer) you can use a rule like the following one:

```
<Rule>
   <Name>title</Name>
   <TextSymbolizer>
      <Label><PropertyName>MYPROPERTY</PropertyName></Label>
   </TextSymbolizer>
</Rule>
```

To render multiple attributes, just define multiple rules, with different names (href, onmouseover, etc.)

Styling support is not limited to TextSymbolizers, you can currently use other symbolizers to detail <area> rendering. For example you can:

- use a PointSymbolizer with a Size property to define point sizes.
- use LineSymbolizer with a stroke-width CssParameter to create thick lines.

20.6 INSPIRE

The INSPIRE extension allows GeoServer to be compliant with the View Service specification put forth by the **Infrastructure for Spatial Information in the European Community** (INSPIRE) directive.

In a practical sense, the INSPIRE plugin extends the WMS capabilities document to include the following extra information: **Metadata URL**, or the link to the metadata associated with the WMS layers; and **SupportedLanguages**, for detailing the default language.

Note: The current INSPIRE extension fulfills "Scenario 1" of the View Service extended metadata requirements. "Scenario 2" is not currently supported in GeoServer, but is certainly possible to implement. If you are interested in implementing or funding this, please raise the issue on the *GeoServer mailing list*.

For more information on the INSPIRE directive, please see the European Commission's INSPIRE website.

20.6.1 Installing the INSPIRE extension

The INSPIRE extension is a official extension available at GeoServer download pages (starting with GeoServer 2.3.2).

1. Download the inspire zip release file from the download page of your version of GeoServer
2. Extract the archive and copy the contents into the `<GEOSERVER_ROOT>/WEB-INF/lib` directory.
3. Restart GeoServer.

To verify that the extension was installed successfully, please see the next section on *Using the INSPIRE extension*.

20.6.2 Using the INSPIRE extension

When the INSPIRE extension has been properly installed, there will be two changes to GeoServer.

1. The GeoServer WMS 1.3.0 capabilities document, as well as the WFS 1.1 and 2.0 will contain extra content relevant to INSPIRE.
2. The *WMS* and *WFS* sections of the *Web Administration Interface* will show extra configuration options.

Extended WMS Capabilities

Note: The INSPIRE directive is relevant to WMS 1.3.0 only, so please make sure that you are viewing the correct capabilities document.

The WMS 1.3.0 capabilities document will be extended once the INSPIRE extension is installed. Those changes are:

1. Two additional entries in the `xsi:schemaLocation` of the root `<WMS_Capabilities>` tag:
   - `http://inspire.ec.europa.eu/schemas/inspire_vs/1.0`
   - `http://<GEOSERVER_ROOT>/www/inspire/inspire_vs.xsd`
2. An additional ExtendedCapabilities block. This tag block shows up in between the tags for `<Exception>` and `<Layer>`. It contains the following information:
   - Metadata URL and MIME type
   - Default Language
   - Supported Language(s)
   - Response Language(s)

   By default, this block will contain the following content:

```
<inspire_vs:ExtendedCapabilities>
  <inspire_common:MetadataUrl xsi:type="inspire_common:resourceLocatorType">
    <inspire_common:URL/>
    <inspire_common:MediaType>application/vnd.iso.19139+xml</inspire_common:MediaType>
  </inspire_common:MetadataUrl>
  <inspire_common:SupportedLanguages xsi:type="inspire_common:supportedLanguagesType">
    <inspire_common:DefaultLanguage>
      <inspire_common:Language>eng</inspire_common:Language>
    </inspire_common:DefaultLanguage>
    <inspire_common:SupportedLanguage>
      <inspire_common:Language>eng</inspire_common:Language>
    </inspire_common:SupportedLanguage>
  </inspire_common:SupportedLanguages>
  <inspire_common:ResponseLanguage>
    <inspire_common:Language>eng</inspire_common:Language>
  </inspire_common:ResponseLanguage>
</inspire_vs:ExtendedCapabilities>
```

This information can be changed via the *WMS* section of the *Web Administration Interface*.

Note: If you do not see this content in the WMS 1.3.0 capabilities document, the INSPIRE extension may not be installed properly. Reread the section on *Installing the INSPIRE extension* and verify that the correct file was saved to the correct directory.

Extended WMS configuration

As with the WMS 1.3.0 capabilities document, the WMS configuration in the *Web Administration Interface* is also extended to allow for changing the above published information. INSPIRE-specific configuration is accessed on the main *WMS* page in the *Web Administration Interface*. This is accessed by clicking on the *WMS* link on the sidebar.

Note: You must be logged in as an administrator to edit WMS configuration.

Once on the WMS configuration page, there will be a block titled *INSPIRE*. This section will have three settings:

- *Language* combo box, for setting the Supported, Default, and Response languages
- *ISO 19139 Service Metadata URL* field, a URL containing the location of the metadata associated with the WMS
- *Service Metadata Type* combo box, for detailing whether the metadata came from a CSW (Catalog Service) or a standalone metadata file

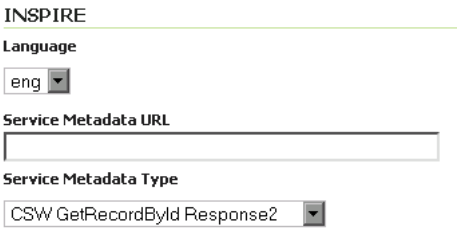


Figure 20.55: *INSPIRE-related options*

Note: If you do not see this content in the WMS configuration page, the INSPIRE extension may not be installed properly. Reread the section on *Installing the INSPIRE extension* and verify that the correct file was saved to the correct directory.

After clicking *Submit* on this page, any changes will be immediately reflected in the WMS 1.3.0 capabilities document.

Note: Currently GeoServer does not offer the ability to configure alternate languages, as there is no way for an administrator to configure multiple responses. There is an open issue on the GeoServer issue tracker that we are hoping to secure funding for. If you are interested in implementing or funding this improvement, please raise the issue on the *GeoServer mailing list*.

Extended WFS Capabilities

Note: The INSPIRE directive is relevant to WFS 1.1 and 2.0 only, so please make sure that you are viewing the correct capabilities document.

The WFS 1.1.0 capabilities document will be extended once the INSPIRE extension is installed. Those changes are:

1. Two additional entries in the `xsi:schemaLocation` of the root element tag:

   ```
   http://inspire.ec.europa.eu/schemas/common/1.0/common.xsd
   http://inspire.ec.europa.eu/schemas/inspire_dls/1.0/inspire_dls.xsd
   ```

2. An additional ExtendedCapabilities block with the following information:

   - Metadata URL and MIME type
   - Default Language
   - Supported Language(s)
   - Response Language(s)
   - Spatial data identifiers

   By default, this block will contain the following content:

```
<inspire_vs:ExtendedCapabilities>
  <inspire_common:MetadataUrl xsi:type="inspire_common:resourceLocatorType">
    <inspire_common:URL/>
    <inspire_common:MediaType>application/vnd.iso.19139+xml</inspire_common:MediaType>
  </inspire_common:MetadataUrl>
  <inspire_common:SupportedLanguages xsi:type="inspire_common:supportedLanguagesType">
    <inspire_common:DefaultLanguage>
      <inspire_common:Language>eng</inspire_common:Language>
    </inspire_common:DefaultLanguage>
    <inspire_common:SupportedLanguage>
      <inspire_common:Language>eng</inspire_common:Language>
    </inspire_common:SupportedLanguage>
  </inspire_common:SupportedLanguages>
  <inspire_common:ResponseLanguage>
    <inspire_common:Language>eng</inspire_common:Language>
  </inspire_common:ResponseLanguage>
</inspire_vs:ExtendedCapabilities>
```

The spatial data identifiers section is mandatory, but cannot be filled by default, it is your duty to provide at least one spatial dataset identifier (see the INSPIRE download service technical guidelines for more information).

This information can be changed via the *WFS* section of the *Web Administration Interface*.

Note: If you do not see this content in the WFS 1.1/2.0 capabilities document, the INSPIRE extension may not be installed properly. Reread the section on *Installing the INSPIRE extension* and verify that the correct file was saved to the correct directory.

Extended WFS configuration

As with the WFS capabilities document, the WFS configuration in the *Web Administration Interface* is also extended to allow for changing the above published information. INSPIRE-specific configuration is accessed on the main *WFS* page in the *Web Administration Interface*. This is accessed by clicking on the *WFS* link on the sidebar.

Note: You must be logged in as an administrator to edit WFS configuration.

Once on the WFS configuration page, there will be a block titled *INSPIRE*. This section will have three settings:

- *Language* combo box, for setting the Supported, Default, and Response languages
- *ISO 19139 Service Metadata URL* field, a URL containing the location of the metadata associated with the WFS
- *Service Metadata Type* combo box, for detailing whether the metadata came from a CSW (Catalog Service) or a standalone metadata file
- *Spatial dataset identifers* table, where you can specify a code (mandatory) and a namespace (optional) for each spatial data set the WFS server is offering

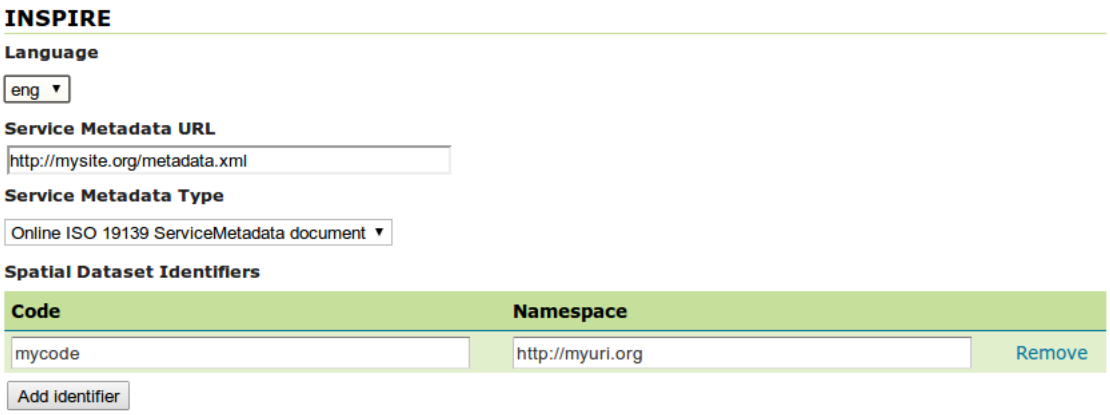


Figure 20.56: *INSPIRE-related options*

Note: If you do not see this content in the WFS configuration page, the INSPIRE extension may not be installed properly. Reread the section on *Installing the INSPIRE extension* and verify that the correct file was saved to the correct directory.

After clicking *Submit* on this page, any changes will be immediately reflected in the WFS 1.1 and WFS 2.0 capabilities documents.

Note: Currently GeoServer does not offer the ability to configure alternate languages, as there is no way for an administrator to configure multiple responses. There is an open issue on the GeoServer issue tracker that we are hoping to secure funding for. If you are interested in implementing or funding this improvement, please raise the issue on the *GeoServer mailing list*.

More information

A tutorial on setting up GeoServer with the INSPIRE extension is available at: http://location.defra.gov.uk/2011/07/data-publisher-how-to-guides/. See the section on *Setting up GeoServer on a Windows Machine*.

20.7 Monitoring

The monitor extension tracks requests made against a GeoServer instance. With the extension request data can be persisted to a database, used to generate simple reports , and routed to a customized request audit

log.

To get the extension proceed to *Installing the Monitor Extension*. To learn more about how it works jump to the *Monitoring Overview* section.

20.7.1 Installing the Monitor Extension

Note: If performing an upgrade of the monitor extension please see *Upgrading*.

The monitor extension is not part of the GeoServer core and must be installed as a plug-in. To install:

1. Navigate to the GeoServer download page.
2. Find the page that matches the version of the running GeoServer.
3. Download the monitor extension. The download link will be in the *Extensions* section under *Other*.

   Note: The *Database Persistence* function is packaged as a separate extension. If you plan to use it both the core "monitor" and "monitor-hibernate" extensions must be installed.

4. Extract the files in this archive to the `WEB-INF/lib` directory of your GeoServer installation.
5. Restart GeoServer

Verifying the Installation

There are two ways to verify that the monitoring extension has been properly installed.

1. Start GeoServer and open the *Web Administration Interface*. Log in using the administration account. If successfully installed, there will be a *Monitor* section on the left column of the home page.

Figure 20.57: *Monitoring section in the web admin interface*

1. Start GeoServer and navigate to the current *GeoServer Data Directory*. If successfully installed, a new directory named `monitoring` will be created in the data directory.

20.7.2 Upgrading

The monitoring extension uses Hibernate to persist request data. Changes to the extension over time affect the structure of the underlying database, which should be taken into consideration before performing an upgrade. Depending on the nature of changes in an upgrade, it may involve manually making changes to the underlying database before deploying a new version of the extension.

The sections below provides a history of such changes, and recommended actions that should be taken as part of the upgrade. Upgrades are grouped into two categories:

- **minor** upgrades that occur during a minor GeoServer version change, for example going from 2.1.2 to 2.1.3. These changes are backward compatible in that no action is specifically required but potentially

recommended. In these cases performing an upgrade without any action still result in the monitoring extension continuing to function.

- **major** upgrades that occur during a major GeoServer version change, for example going from 2.1.2 to 2.2.0. These changes *may* be backward compatible, but not necessarily. In these cases performing an upgrade without any action could potentially result in the monitoring extension ceasing to function, and may result in significant changes to the underlying database.

For each change the following information is maintained:

- The released version containing the change
- The date of the change
- The subversion revision of the change
- The jira issue referring to the change

The date and subversion revision are especially useful if a nightly build of the extension is being used.

Minor upgrades

Column `resource` renamed to `name` in `request_resources` table

- *Version*: n/a, extension still community status
- *Date*: Dec 09, 2011
- *Subversion revision*: 16632
- *Reference*: GEOS-4871

Upgrading without performing any action will result in the `name` column being added to the `request_resources` table, leaving the `resource` column in tact. From that point forward the `resource` column will essentially be ignored. However no data from the `resource` column will be migrated, which will throw off reports, resource access statistics, etc... If you wish to migrate the data perform one of the following actions two actions.

The first is a *pre* upgrade action that involves simply renaming the column before deploying the new monitoring extension:

```
ALTER TABLE request_resources RENAME COLUMN resource to name;
```

Alternatively the migration may occur *post* upgrade:

```
UPDATE TABLE request_resources SET name = resource where name is NULL;
ALTER TABLE request_resources DROP COLUMN resource;
```

Column `remote_user_agent` added to `request` table

- *Version*: n/a, extension still community status
- *Date*: Dec 09, 2011
- *Subversion revision*: 16634
- *Reference*: GEOS-4872

No action should be required here as Hibernate will simply append the new column to the table. If for some reason this does not happen the column can be added manually:

```
ALTER TABLE request ADD COLUMN remote_user_agent VARCHAR(1024);
```

Major upgrades

20.7.3 Monitoring Overview

The following diagram outlines the architecture of the monitor extension:

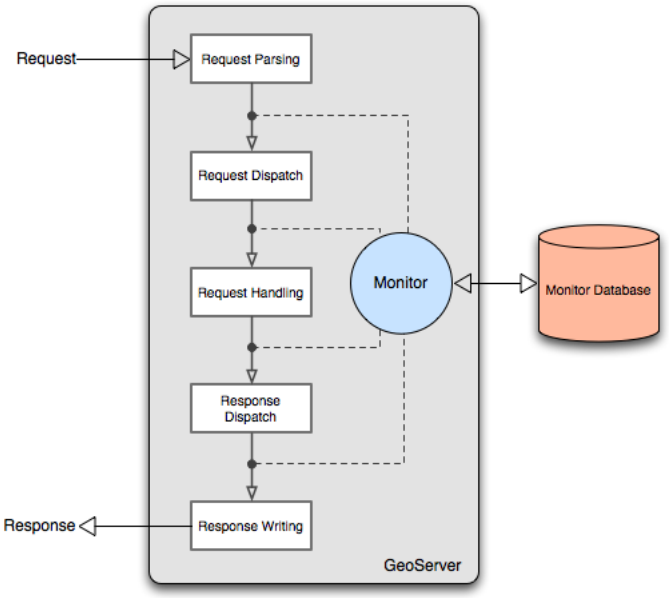


Figure 20.58: *Monitor extension architecture*

As a request is processed the monitor inserts itself at particular points in the request life cycle to capture various information about the request. Such information includes:

- Timestamp of the origin of the request
- Total ime it took for the request to complete
- Origin of the request
- HTTP information such as the body content type, header information, etc...

And more. See the *Data Reference* section for a complete list.

In addition to capturing request data the monitor extension is also capable of persisting it. Two options are provided out of the box:

- Persisting to a relational database, see *Database Persistence* for more details
- Piping to a log file, see *Audit Logging* for more details

By default the extension will do neither and simply maintain data for only the most recent requests. The data is stored in memory meaning that if the server is restarted or shutdown this information is lost.The *Monitor Configuration* section provides a comprehensive guide to configuring the monitor extension.

Stored request information is made available through a simple *query api* that allows clients to access request data through a HTTP interface.

20.7.4 Data Reference

The following is a list of all the attributes of a request that are captured by the monitor extension.

General

Attribute	Description	Type
ID	Numeric identifier of the request. Every request is assigned an identifier upon its creation.	Numeric
Status	Status of the request. See *notes* below.	String
Category	The type of request being made, for example an OGC service request, a REST call, etc... See *notes* below.	String
Start time	The time of the start of the request.	Timestamp
End time	The time of the completion of the request.	Timestamp
Total time	The total time spent handling the request, measured in milliseconds, equal to the end time - start time.	Numeric
Error message	The exception message if the request failed or resulted in an error.	String
Error	The raw exception if the message failed or resulted in an error.	Text blob

Status

The status of a request changes over it's life cycle and may have one of the following values:

- `WAITING` - The request has been received by the server, but is queued and not yet being actively handled.
- `RUNNING` - The request is in the process of being handled by the server.
- `FINISHED` - The request has been completed and finished normally.
- `FAILED` - The request has been completed but resulted in an error.
- `CANCELLED` - The request was cancelled before it could complete.
- `INTERRUPTED` - The request was interrupted before it could complete.

Category

Requests are grouped into categories that describe the nature or type of the request. The following are the list of all categories:

- `OWS` - The request is an OGC service request.
- `REST` - The request is a REST service request.
- `OTHER` - All other requests.

HTTP

The following attributes are all HTTP related.

Attribute	Description	Type
HTTP method	The HTTP method, one of `GET`, `POST`, `PUT`, or `DELETE`	String
Remote address	The IP address of the client from which the request originated.	String
Remote host	The hostname corresponding to the remote address, obtained via reverse DNS lookup.	String
Host	The hostname of the server handling the request, from the point of view of the client.	String
Internal host	The hostname of the server handling request, from the point of view of the local network. Availability depends on host and network configuration.	String
Path	The path component of the request URL, for example: "/wms", "/rest/workspaces.xml", etc...	String
Query string	The query string component of the request URL. Typically only present when the HTTP method is GET.	String
Body	The body content of the request. Typically only present when the HTTP method is PUT or POST.	Binary blob
Body content length	The total number of bytes comprising the body of the request. Typically only present when the HTTP method is PUT or POST.	Numeric
Body content type	The mime type of the body content of the request, for example: "application/json", "text/xml; subtype=gml/3.2", etc... Typically only present when the HTTP method is PUT or POST.	String
Response status	The HTTP response code, for example: 200, 401, etc...	Numeric
Response length	The total number of bytes comprising the response to the request.	Numeric
Response content type	The mime type of the response to the request.	String
Remote user	The username specified parsed of the request. Only available when request included credentials for authentication.	String
Remote user agent	The value of the `User-Agent` HTTP header.	String
Http referrer	The value of the `Referer` HTTP header.	String

OWS/OGC

The following attributes are OGC service specific.

Attribute	Description	Type
Service	The OGC service identifier, for example: “WMS”, “WFS”, etc...	String
Operation	The OGC operation name, for example: “GetMap”, “GetFeature”, etc...	String
Sub operation	The ogc sub operation (if it applies). For instance when the operation is a WFS Transaction the sub operation may be one of “Insert”, “Update”, etc...	String
OWS/OGC Version	The OGC service version, for example with WFS the version may be “1.0.0”, “1.1.0”, etc...	String
Resources	Names of resources (layers, processes, etc...) specified as part of the request.	List of String
Bounding box	The bounding box specified as part of the request. In some cases this is not possible to obtain this reliable, an example being a complex WFS query with a nested “BBOX” filter.	List of Numeric

GeoIP

The following attributes are specific to GeoIP look ups and are not captured out of the box. See *GeoIP* for more details.

Attribute	Description	Type
Remote country	Name of the country of the client from which the request originated.	String
Remote city	Name of the city from which the request originated.	String
Remote lat	The latitude from which the request originated.	Numeric
Remote lon	The longitude from which the request originated.	Numeric

20.7.5 Monitor Configuration

Many aspects of the monitor extension are configurable. All configuration files are stored in the data directory under the `monitoring` directory:

```
<data_directory>
    monitoring/
        db.properties
        filter.properties
        hibernate.properties
        monitor.properties
```

The `monitor.properties` file is the main configuration file whose contents are described in the following sections. Other configuration files include:

- **filter.properties** - Allows for *filtering* out those requests from being monitored.
- **db.properties** - Database configuration when using database persistence.
- **hibernate.properties** - Hibernate configuration when using database persistence.

Database persistence with hibernate is described in more detail in the *Database Persistence* section.

Monitor Storage

How request data is persisted is configurable via the `storage` property defined in the `monitor.properties` file. The following values are supported for the `storage` property:

- **memory** - Request data is to be persisted in memory alone.
- **hibernate** - Request data is to be persisted in a relational database via Hibernate.

The default value is `memory`.

Memory Storage

With memory storage only the most recent 100 requests are stored. And by definition this storage is volatile in that if the GeoServer instance is restarted, shutdown, or crashes this data is lost.

Hibernate Storage

Hibernate storage is described in detail in the *Database Persistence* section.

Monitor Mode

The monitor extension supports different "monitoring modes" that control how request data is captured. Currently two modes are supported:

- **history** *(Default)* - Request information updated post request only. No live information made available.
- **live** - Information about a request is captured and updated in real time.

The monitor mode is set with the `mode` property in the `monitor.properties` file. The default value is `history`.

History Mode

History mode persists information (sending it to storage) about a request after a request has completed. This mode is appropriate in cases where a user is most interested in analyzing request data after the fact and doesn't require real time updates.

Live Mode

Live mode updates request data (sending it to storage) in real time as it changes. This mode is suitable for users who care about what a service is doing now.

Bounding Box

When applicable one of the attributes the monitor extension can capture is the request bounding box. In some cases, such as WMS and WCS requests, capturing the bounding box is easy. However in other cases such as WFS it is not always possible to 100% reliably capture the bounding box. An example being a WFS request with a complex filter element.

How the bounding box is captured is controlled by the `bboxMode` property in the `monitor.properties` file. It can have one of the following values.

- **none** - No bounding box information is captured.
- **full** - Bounding box information is captured and heuristics are applied for WFS requests.
- **no_wfs** - Bounding box information is captured except for WFS requests.

Part of a bounding box is a coordinate reference system (crs).Similar to the WFS case it is not always straight forward to determine what the crs is. For this reason the `bboxCrs` property is used to configure a default crs to be used. The default value for the property is "EPSG:4326" and will be used in cases where all lookup heuristics fail to determine a crs for the bounding box.

Request Body Size

The monitor extension will capture the contents of the request body when a body is specified as is common with a PUT or POST request. However since a request body can be large the extension limits the amount captured to the first 1024 bytes by default.

A value of `0` indicates that no data from the request body should be captured. A value of `-1` indicates that no limit should be placed on the capture and the entire body content should be stored.

This limit is configurable with the `maxBodySize` property of the `monitor.properties` file.

Note: When using database persistence it is important to ensure that the size of the body field in the database can accommodate the `maxBodySize` property.

Request Filters

By default not all requests are monitored. Those requests excluded include any web admin requests or any *Monitor Query API* requests. These exclusions are configured in the `filter.properties` file:

```
/rest/monitor/**
/web/**
```

These default filters can be changed or extended to filter more types of requests. For example to filter out all WFS requests the following entry is added:

```
/wfs
```

How to determine the filter path

The contents of `filter.properties` are a series of ant-style patterns that are applied to the *path* of the request. Consider the following request:

```
http://localhost:8080/geoserver/wms?request=getcapabilities
```

The path of the above request is `/wms`. In the following request:

```
http://localhost:8080/geoserver/rest/workspaces/topp/datastores.xml
```

The path is `/rest/workspaces/topp/datastores.xml`.

In general, the path used in filters is comprised of the portion of the URL after `/geoserver` (including the preceding `/`) and before the query string `?`:

```
http://<host>:<port>/geoserver/<path>?<queryString>
```

Note: For more information about ant-style pattern matching, see the Apache Ant manual.

Samples

monitor.properties

```
# storage and mode
storage=memory
mode=history

# request body capture
maxBodySize=1024

# bounding box capture
bboxMode=no_wfs
bboxCrs=EPSG:4326
```

filter.properties

```
# filter out monitor query api requests
/rest/monitor/**

# filter out all web requests
/web
/web/**

# filter out requests for WCS service
/wcs
```

20.7.6 Database Persistence

The monitor extension is capable of persisting request data to a database via the Hibernate library.

Note: In order to utilize hibernate persistence the hibernate extension must be installed on top of the core monitoring extension. See the *Installing the Monitor Extension* for details.

Configuration

General

In order to activate hibernate persistence the `storage` parameter must be set to the value "hibernate":

```
storage=hibernate
```

The hibernate storage backend supports both the `history` and `live` modes however care should be taken when enabling the `live` mode as it results in many transactions with the database over the life of a request. Unless updating the database in real time is required the `history` mode is recommended.

Database

The file `db.properties` in the `<GEOSERVER_DATA_DIR>/monitoring` directory specifies the Hibernate database. By default an embedded H2 database located in the `monitoring` directory is used. This can be changed by editing the `db.properties` file:

```
# default configuration is for h2
driver=org.h2.Driver
url=jdbc:h2:file:${GEOSERVER_DATA_DIR}/monitoring/monitoring
```

For example to store request data in an external PostgreSQL database, set `db.properties` to:

```
driver=org.postgresql.Driver
url=jdbc:postgresql://192.168.1.124:5432/monitoring
username=bob
password=foobar
defaultAutoCommit=false
```

In addition to `db.properties` file is the `hibernate.properties` file that contains configuration for Hibernate itself. An important parameter of this file is the hibernate dialect that informs hibernate of the type of database it is talking to.

When changing the type of database both the `databasePlatform` and `database` parameters must be updated. For example to switch to PostgreSQL:

```
# hibernate dialect
databasePlatform=org.hibernate.dialect.PostgreSQLDialect
database=POSTGRESQL

# other hibernate configuration
hibernate.use_sql_comments=true
generateDdl=true
hibernate.format_sql=true
showSql=false
hibernate.generate_statistics=true
hibernate.session_factory_name=SessionFactory
hibernate.hbm2ddl.auto=update
hibernate.bytecode.use_reflection_optimizer=true
hibernate.show_sql=false
```

Hibernate

As mentioned in the previous section the `hibernate.properties` file contains the configuration for Hibernate itself. Aside from the database dialect parameters it is not recommended that you change this file unless you are an experienced Hibernate user.

20.7.7 Audit Logging

The history mode logs all requests into a database. This can put a very significant strain on the database and can lead to insertion issues as the request table begins to host millions of records.

As an alternative to the history mode it's possible to enable the auditing logger, which will log the details of each request in a file, which is periodically rolled. Secondary applications can then process these log files and built ad-hoc summaries off line.

Configuration

The `monitor.properties` file can contain the following items to enable and configure file auditing:

```
audit.enabled=true
audit.path=/path/to/the/logs/directory
audit.roll_limit=20
```

The `audit.enable` is used to turn on the logger (it is off by default). The `audit.path` is the directory where the log files will be created. The `audit.roll_limit` is the number of requests logged into a file before rolling happens. The files are also automatically rolled at the beginning of each day.

In clustered installations with a shared data directory the audit path will need to be different for each node. In this case it's possible to specify the audit path by using a JVM system variable, add the following to the JVM startup options and it will override whatever is specified in `monitor.properties`:

```
-DGEOSERVER_AUDIT_PATH=/path/to/the/logs/directory
```

Log Files

The log directory will contain a number of log files following the `geoserver_audit_yyyymmdd_nn.log` pattern. The `nn` is increased at each roll of the file. The contents of the log directory will look like:

```
geoserver_audit_20110811_2.log
geoserver_audit_20110811_3.log
geoserver_audit_20110811_4.log
geoserver_audit_20110811_5.log
geoserver_audit_20110811_6.log
geoserver_audit_20110811_7.log
geoserver_audit_20110811_8.log
```

By default each log file contents will be a xml document looking like the following:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Requests>
        <Request id="168">
           <Service>WMS</Service>
           <Version>1.1.1</Version>
           <Operation>GetMap</Operation>
           <SubOperation></SubOperation>
           <Resources>GeoSolutions:elba-deparea</Resources>
           <Path>/GeoSolutions/wms</Path>
           <QueryString>LAYERS=GeoSolutions:elba-deparea&amp;STYLES=&amp;FORMAT=image/png&amp;TILED=t
           <HttpMethod>GET</HttpMethod>
           <StartTime>2011-08-11T20:19:28.277Z</StartTime>
           <EndTime>2011-08-11T20:19:28.29Z</EndTime>
           <TotalTime>13</TotalTime>
           <RemoteAddr>192.168.1.5</RemoteAddr>
           <RemoteHost>192.168.1.5</RemoteHost>
           <Host>demo1.geo-solutions.it</Host>
           <RemoteUser>admin</RemoteUser>
           <ResponseStatus>200</ResponseStatus>
           <ResponseLength>1670</ResponseLength>
           <ResponseContentType>image/png</ResponseContentType>
           <Failed>false</Failed>
        </Request>
        ...
</Requests>
```

Customizing Log Contents

The log contents are driven by three FreeMarker templates.

`header.ftl` is used once when a new log file is created to form the first few lines of the file. The default header template is:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Requests>
```

`content.ftl` is used to write out the request details. The default template dumps all the known fields about the request:

```
<#escape x as x?xml>
<Request id="${id!""}">
   <Service>${service!""}</Service>
   <Version>${owsVersion!""}</Version>
   <Operation>${operation!""}</Operation>
   <SubOperation>${subOperation!""}</SubOperation>
   <Resources>${resourcesList!""}</Resources>
   <Path>${path!""}</Path>
   <QueryString>${queryString!""}</QueryString>
   <#if bodyAsString??>
   <Body>
   ${bodyAsString}
   </Body>
   </#if>
   <HttpMethod>${httpMethod!""}</HttpMethod>
   <StartTime>${startTime?datetime?iso_utc_ms}</StartTime>
   <EndTime>${endTime?datetime?iso_utc_ms}</EndTime>
   <TotalTime>${totalTime}</TotalTime>
   <RemoteAddr>${remoteAddr!""}</RemoteAddr>
   <RemoteHost>${remoteHost!""}</RemoteHost>
   <Host>${host}</Host>
   <RemoteUser>${remoteUser!""}</RemoteUser>
   <ResponseStatus>${responseStatus!""}</ResponseStatus>
   <ResponseLength>${responseLength?c}</ResponseLength>
   <ResponseContentType>${responseContentType!""}</ResponseContentType>
   <#if error??>
   <Failed>true</Failed>
   <ErrorMessage>${errorMessage!""}</ErrorMessage>
   <#else>
   <Failed>false</Failed>
   </#if>
</Request>
</#escape>
```

`footer.ftl` is executed just once when the log file is closed to build the last few lines of the file. The default footer template is:

```
</Requests>
```

The administrator is free to provide alternate templates, they can be placed in the same directory as `monitor.properties`, with the same names as above. GeoServer will pick them up automatically.

20.7.8 Monitor Query API

The monitor extension provides a simple HTTP-based API for querying request information. It allows retrieving individual request records or sets of request records, in either HTML or CSV format. Records can be filtered by time range and the result set sorted by any field. Large result sets can be paged over multiple queries.

Examples

The following examples show the syntax for common Monitoring queries.

All requests as HTML

The simplest query is to retrieve an HTML document containing information about all requests:

```
GET http://localhost:8080/geoserver/rest/monitor/requests.html
```

All requests as CSV

Request information can be returned in CSV format, for easier post-processing:

```
GET http://localhost:8080/geoserver/rest/monitor/requests.csv
```

Request bodies containing newlines are handled with quoted text. If your CSV reader doesn't handle quoted newlines, it will not work correctly.

All requests as PKZip

A PKZip archive containing the CSV file above, with all the request bodies and errors as separate files:

```
GET http://localhost:8080/geoserver/rest/monitor/requests.zip
```

All requests as MS Excel

A Microsoft Excel spreadsheet containing the same information as the CSV file:

```
GET http://localhost:8080/geoserver/rest/monitor/requests.xls
```

Requests during a time period

Requests can be filtered by date and time range:

```
GET http://localhost:8080/geoserver/rest/monitor/requests.html?from=2010-06-20&to=2010-07-20
GET http://localhost:8080/geoserver/rest/monitor/requests.html?from=2010-06-20T2:00:00&to=2010-06-20T
```

Request set paging

Large result sets can be paged over multiple queries:

```
GET http://localhost:8080/geoserver/rest/monitor/requests.html?count=100
GET http://localhost:8080/geoserver/rest/monitor/requests.html?count=100&offset=100
GET http://localhost:8080/geoserver/rest/monitor/requests.html?count=100&offset=200
GET http://localhost:8080/geoserver/rest/monitor/requests.html?count=100&offset=300
```

Single request

An individual request can be retrieved by specifying its ID:

```
GET http://localhost:8080/geoserver/rest/monitor/requests/12345.html
```

API Reference

There are two kinds of query: one for single requests, and one for sets of requests.

Single Request Query

A query for a single request record has the structure:

```
GET http://<host>:<port>/geoserver/rest/monitor/requests/<id>.<format>
```

where `id` is the numeric identifier of a single request, and `format` specifies the representation of the returned result as one of:

- `html` - an HTML table.
- `csv` - a Comma Separated Values table.
- `zip` - PKZip archive containing CSV as above, plus plain text of errors and request body.
- `xls` - Microsoft Excel spreadsheet.

Note: An alternative to specifying the returned representation with the `format` extension is to use the http `Accept` header and specify the MIME type as one of:

- `text/html`
- `application/csv`
- `application/zip`
- `application/vnd.ms-excel`

See the HTTP specification for more information about the `Accept` header.

Request Set Query

The structure of a query for a set of requests is:

```
GET http://<host>:<port>/geoserver/rest/monitor/requests.<format>[?parameter{&parameter}]
```

where `format` is as described above, and `parameter` is one or more of the parameters listed below.

The request set query accepts various parameters that control what requests are returned and how they are sorted. The available parameters are:

count Parameter

Specifies how many records should be returned.

Syntax	Example
`count=<integer>`	requests.html?count=100

offset Parameter

Specifies where in the result set records should be returned from.

Syntax	Example
`offset=<integer>`	requests.html?count=100&offset=500

live Parameter

Specifies that only live (currently executing) requests be returned.

Syntax	Example
`live=<yes|no|true|false>`	requests.html?live=yes

This parameter relies on a *Monitor Mode* being used that maintains real time request information (either **live** or **mixed**).

from Parameter

Specifies an inclusive lower bound on the timestamp for the start of a request. The timestamp can be specified to any desired precision.

Syntax	Example
`from=<timestamp>`	requests.html?from=2010-07-23T16:16:44 requests.html?from=2010-07-23

to Parameter

Specifies an inclusive upper bound on the timestamp for the start of a request. The timestamp can be specified to any desired precision.

Syntax	Example
`to=<timestamp>`	requests.html?to=2010-07-24T00:00:00 requests.html?to=2010-07-24

order Parameter

Specifies which request attribute to sort by, and optionally specifies the sort direction.

Syntax	Example
`order=<attribute>[;<ASC|DESC>]`	requests.html?order=path requests.html?order=startTime;DESC requests.html?order=totalTime;ASC

20.7.9 GeoIP

The monitor extension has the capability to integrate with the MaxMind GeoIP database in order to provide geolocation information about the origin of a request. This functionality is not enabled by default.

Note: At this time only the freely available GeoLite City database is supported.

Enabling GeoIP Lookup

In order to enable the GeoIP lookup capabilities

1. Download the GeoLite City database.
2. Uncompress the file and copy `GeoLiteCity.dat` to the `monitoring` directory.
3. Restart GeoServer.

20.8 OGR based WFS Output Format

The ogr2ogr based output format leverages the availability of the ogr2ogr command to allow the generation of more output formats than GeoServer can natively produce. The basics idea is to dump to the file system a file that ogr2ogr can translate, invoke it, zip and return the output of the translation.

20.8.1 Out of the box behaviour

Out of the box the plugin assumes the following:

- ogr2ogr is available in the path
- the GDAL_DATA variable is pointing to the GDAL data directory (which stores the spatial reference information for GDAL)

In the default configuration the following formats are supported:

- MapInfo in TAB format
- MapInfo in MIF format
- Un-styled KML
- CSV (without geometry data dumps)

The list might be shorter if ogr2ogr has not been built with support for the above formats.

Once installed in GeoServer four new GetFeature output formats will be available, in particular, `OGR-TAB`, `OGR-MIF`, `OGR-KML`, `OGR-CSV`.

20.8.2 ogr2ogr conversion abilities

The ogr2ogr utility is usually able to convert more formats than the default setup of this output format allows for, but the exact list depends on how the utility was built from sources. To get a full list of the formats available by your ogr2ogr build just run:

```
ogr2ogr --help
```

and you'll get the full set of options usable by the program, along with the supported formats. For example, the above produces the following output using the FWTools 2.2.8 distribution (which includes ogr2ogr among other useful information and conversion tools):

```
Usage: ogr2ogr [--help-general] [-skipfailures] [-append] [-update] [-gt n]
            [-select field_list] [-where restricted_where]
            [-sql <sql statement>]
            [-spat xmin ymin xmax ymax] [-preserve_fid] [-fid FID]
            [-a_srs srs_def] [-t_srs srs_def] [-s_srs srs_def]
            [-f format_name] [-overwrite] [[-dsco NAME=VALUE] ...]
```

```
            [-segmentize max_dist]
            dst_datasource_name src_datasource_name
            [-lco NAME=VALUE] [-nln name] [-nlt type] [layer [layer ...]]

-f format_name: output file format name, possible values are:
  -f "ESRI Shapefile"
  -f "MapInfo File"
  -f "TIGER"
  -f "S57"
  -f "DGN"
  -f "Memory"
  -f "BNA"
  -f "CSV"
  -f "GML"
  -f "GPX"
  -f "KML"
  -f "GeoJSON"
  -f "Interlis 1"
  -f "Interlis 2"
  -f "GMT"
  -f "SQLite"
  -f "ODBC"
  -f "PostgreSQL"
  -f "MySQL"
  -f "Geoconcept"
-append: Append to existing layer instead of creating new if it exists
-overwrite: delete the output layer and recreate it empty
-update: Open existing output datasource in update mode
-select field_list: Comma-delimited list of fields from input layer to
                   copy to the new layer (defaults to all)
-where restricted_where: Attribute query (like SQL WHERE)
-sql statement: Execute given SQL statement and save result.
-skipfailures: skip features or layers that fail to convert
-gt n: group n features per transaction (default 200)
-spat xmin ymin xmax ymax: spatial query extents
-segmentize max_dist: maximum distance between 2 nodes.
                      Used to create intermediate points
-dsco NAME=VALUE: Dataset creation option (format specific)
-lco  NAME=VALUE: Layer creation option (format specific)
-nln name: Assign an alternate name to the new layer
-nlt type: Force a geometry type for new layer.  One of NONE, GEOMETRY,
     POINT, LINESTRING, POLYGON, GEOMETRYCOLLECTION, MULTIPOINT,
     MULTIPOLYGON, or MULTILINESTRING.  Add "25D" for 3D layers.
     Default is type of source layer.
-a_srs srs_def: Assign an output SRS
-t_srs srs_def: Reproject/transform to this SRS on output
-s_srs srs_def: Override source SRS

Srs_def can be a full WKT definition (hard to escape properly),
or a well known definition (ie. EPSG:4326) or a file with a WKT
definition.
```

The full list of formats that ogr2ogr is able to support is available on the OGR site. Mind that this output format can handle only outputs that are file based and that do support creation. So, for example, you won't be able to use the Postgres output (since it's database based) or the ArcInfo binary coverage (creation not supported).

20.8.3 Customisation

If ogr2ogr is not available in the default path, the GDAL_DATA is not set, or if the output formats needs tweaking, a `ogr2ogr.xml` file can be put in the root of the GeoServer data directory to customize the output format.

The default GeoServer configuration is equivalent to the following xml file:

```
<OgrConfiguration>
  <ogr2ogrLocation>ogr2ogr</ogr2ogrLocation>
  <!-- <gdalData>...</gdalData> -->
  <formats>
    <Format>
      <ogrFormat>MapInfo File</ogrFormat>
      <formatName>OGR-TAB</formatName>
      <fileExtension>.tab</fileExtension>
    </Format>
    <Format>
      <ogrFormat>MapInfo File</ogrFormat>
      <formatName>OGR-MIF</formatName>
      <fileExtension>.mif</fileExtension>
      <option>-dsco</option>
      <option>FORMAT=MIF</option>
    </Format>
    <Format>
      <ogrFormat>CSV</ogrFormat>
      <formatName>OGR-CSV</formatName>
      <fileExtension>.csv</fileExtension>
      <singleFile>true</singleFile>
      <mimeType>text/csv</mimeType>
    </Format>
    <Format>
      <ogrFormat>KML</ogrFormat>
      <formatName>OGR-KML</formatName>
      <fileExtension>.kml</fileExtension>
      <singleFile>true</singleFile>
      <mimeType>application/vnd.google-earth.kml</mimeType>
    </Format>
  </formats>
</OgrConfiguration>
```

The file showcases all possible usage of the configuration elements:

- `ogr2ogrLocation` can be just ogr2ogr if the command is in the path, otherwise it should be the full path to the executable. For example, on a Windows box with FWTools installed it might be:

  ```
  <ogr2ogrLocation>c:\Programmi\FWTools2.2.8\bin\ogr2ogr.exe</ogr2ogrLocation>
  ```

- `gdalData` must point to the GDAL data directory. For example, on a Windows box with FWTools installed it might be:

  ```
  <gdalData>c:\Programmi\FWTools2.2.8\data</gdalData>
  ```

- `Format` defines a single format, which is defined by the following tags:
  - `ogrFormat`: the name of the format to be passed to ogr2ogr with the -f option (it's case sensitive).
  - `formatName`: is the name of the output format as advertised by GeoServer

- `fileExtension`: is the extension of the file generated after the translation, if any (can be omitted)
- `option`: can be used to add one or more options to the ogr2ogr command line. As you can see by the MIF example, each item must be contained in its own tag. You can get a full list of options by running ogr2ogr –help or by visiting the ogr2ogr web page. Also consider that each format supports specific creation options, listed in the description page for each format (for example, here is the MapInfo one).
- `singleFile` (since 2.0.3): if true the output of the conversion is supposed to be a single file that can be streamed directly back without the need to wrap it into a zip file
- `mimeType` (since 2.0.3): the mime type of the file returned when using `singleFile`. If not specified `application/octet-stream` will be used as a default.

20.9 Cross-layer filtering

Cross-layer filtering provides the ability to find features from layer A that have a certain relationship to features in layer B. This can be used, for example, to find all bus stops within a given distance from a specified shop, or to find all coffee shops contained in a specified city district.

The **querylayer** module adds filter functions that implement cross-layer filtering. The functions work by querying a secondary layer within a filter being applied to a primary layer. The name of the secondary layer and an attribute to extract from it are provided as arguments, along with an ECQL filter expression to determine which features are of interest. A common use case is to extract a geometry-valued attribute, and then use the value(s) in a spatial predicate against a geometry attribute in the primary layer.

Filter functions are widely supported in GeoServer, so cross-layer filtering can be used in SLD rules and WMS and WFS requests, in either XML or CQL filters.

20.9.1 Installing the querylayer module

1. Download the **querylayer** extension corresponding to your version of GeoServer.

   Warning: The version of the extension **must** match the version of the GeoServer instance

2. Extract the contents of the extension archive into the `WEB-INF/lib` directory of the GeoServer installation.
3. To check the module is properly installed request the WFS 1.1 capabilities from the GeoServer home page. The `Filter_Capabilities` section should contain a reference to a function named `queryCollection`.

```
...
<ogc:Filter_Capabilities>
    ...
    <ogc:ArithmeticOperators>
      ...
      <ogc:Functions>
        <ogc:FunctionNames>
          ...
          <ogc:FunctionName nArgs="-1">queryCollection</ogc:FunctionName>
          <ogc:FunctionName nArgs="-1">querySingle</ogc:FunctionName>
          ...
        </ogc:FunctionNames>
```

```
        </ogc:Functions>
      </ogc:ArithmeticOperators>
    </ogc:Scalar_Capabilities>
    ...
  </ogc:Filter_Capabilities>
  ...
```

20.9.2 Function reference

The extension provides the following filter functions to support cross-layer filtering.

Name	Arguments	Description
queryS-ingle	`layer` : String, `attribute` : String, `filter` : String	Queries the specified `layer` applying the specified *ECQL* `filter` and returns the value of `attribute` from the first feature in the result set. The layer name must be qualified (e.g. `topp:states`). If no filtering is desired use the filter `INCLUDE`.
queryCol-lection	`layer` : String, `attribute` : String, `filter` : String	Queries the specified `layer` applying the specified *ECQL* `filter` and returns a list containing the value of `attribute` for every feature in the result set. The layer name must be qualified (e.g. `topp:states`). If no filtering is desired use the filter `INCLUDE`. An exception is thrown if too many results are collected (see *Memory Limits*).
collect-Ge-ome-tries	`geometries`: a list of Geometry objects	Converts a list of geometries into a single Geometry object. The output of `queryCollection` must be converted by this function in order to use it in spatial filter expressions (since geometry lists cannot be used directly). An exception is thrown if too many coordinates are collected (see *Memory Limits*).

20.9.3 Optimizing performance

In the GeoServer 2.1.x series, in order to have cross-layer filters execute with optimal performance it is necessary to specify the following system variable when starting the JVM:

```
-Dorg.geotools.filter.function.simplify=true
```

This ensures the functions are evaluated once per query, instead of once per result feature. This flag is not necessary for the GeoServer 2.2.x series. (Hopefully this behavior will become the default in 2.1.x as well.)

20.9.4 Memory limits

The `queryCollection` and `collectGeometries` functions do not perform a true database-style join. Instead they execute a query against the secondary layer every time they are executed, and load the entire result into memory. The functions thus risk using excessive server memory if the query result set is very large, or if the collected geometries are very large. To prevent impacting server stability there are built-in limits to how much data can be processed:

- at most 1000 features are collected by `queryCollection`
- at most 37000 coordinates (1MB worth of Coordinate objects) are collected by `collectGeometries`

These limits can be overridden by setting alternate values for the following parameters (this can be done using JVM system variables, servlet context variables, or enviroment variables):

- `QUERY_LAYER_MAX_FEATURES` controls the maximum number of features collected by `queryCollection`
- `GEOMETRY_COLLECT_MAX_COORDINATES` controls the maximum number of coordinates collected by `collectGeometries`

20.9.5 WMS Examples

The following examples use the `sf:bugsites`, `sf:roads` and `sf:restricted` demo layers available in the standard GeoServer download.

- **Display only the bug sites overlapping the restricted area whose category is 3**:

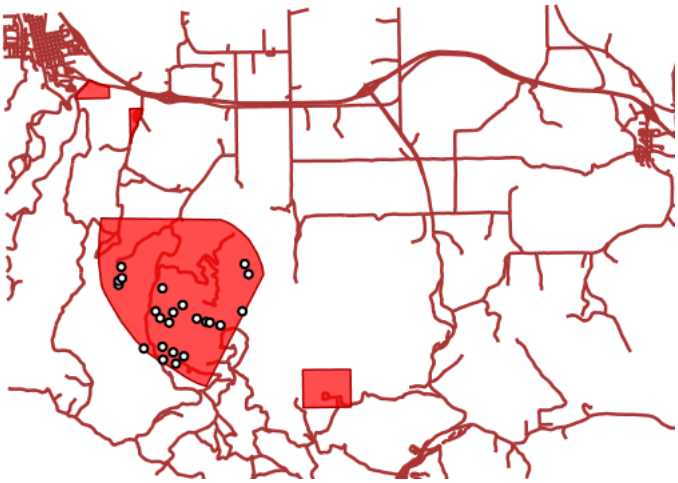
The CQL cross-layer filter on the `bugsites` layer is

```
INTERSECTS(the_geom, querySingle('restricted', 'the_geom','cat = 3')).
```

The WMS request is:

```
http://localhost:8080/geoserver/wms?LAYERS=sf%3Aroads%2Csf%3Arestricted%2Csf%3Abugsites&STYLES=&FORMA
```

The result is:



- **Display all bug sites within 200 meters of any road**:

The CQL cross-layer filter on the `bugsites` layer is

```
DWITHIN(the_geom, collectGeometries(queryCollection('sf:roads','the_geom','INCLUDE')),
200, meters).
```

The WMS request is:

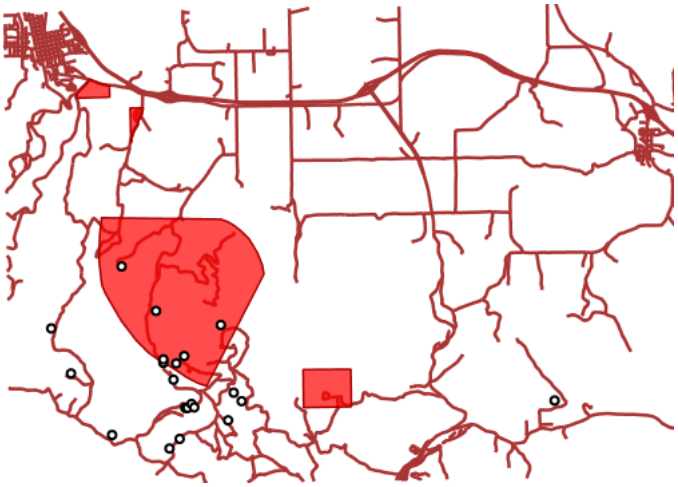
```
http://localhost:8080/geoserver/wms?LAYERS=sf%3Aroads%2Csf%3Arestricted%2Csf%3Abugsites&STYLES=&FORMA
```

The result is:

20.9.6 WFS Examples

The following examples use the `sf:bugsites`, `sf:roads` and `sf:restricted` demo layers available in the standard GeoServer download.

- **Retrieve only the bug sites overlapping the restricted area whose category is 3**:



```
<wfs:GetFeature xmlns:wfs="http://www.opengis.net/wfs"
                xmlns:sf="http://www.openplans.org/spearfish"
                xmlns:ogc="http://www.opengis.net/ogc"
                service="WFS" version="1.0.0">
  <wfs:Query typeName="sf:bugsites">
    <ogc:Filter>
      <ogc:Intersects>
        <ogc:PropertyName>the_geom</ogc:PropertyName>
        <ogc:Function name="querySingle">
           <ogc:Literal>sf:restricted</ogc:Literal>
           <ogc:Literal>the_geom</ogc:Literal>
           <ogc:Literal>cat = 3</ogc:Literal>
        </ogc:Function>
      </ogc:Intersects>
    </ogc:Filter>
  </wfs:Query>
</wfs:GetFeature>
```

- **Retrieve all bugsites within 200 meters of any road**:

```
<wfs:GetFeature xmlns:wfs="http://www.opengis.net/wfs"
  xmlns:sf="http://www.openplans.org/spearfish"
  xmlns:ogc="http://www.opengis.net/ogc"
  service="WFS" version="1.0.0">
  <wfs:Query typeName="sf:bugsites">
    <ogc:Filter>
      <ogc:DWithin>
        <ogc:PropertyName>the_geom</ogc:PropertyName>
        <ogc:Function name="collectGeometries">
          <ogc:Function name="queryCollection">
            <ogc:Literal>sf:roads</ogc:Literal>
            <ogc:Literal>the_geom</ogc:Literal>
            <ogc:Literal>INCLUDE</ogc:Literal>
          </ogc:Function>
        </ogc:Function>
        <ogc:Distance units="meter">100</ogc:Distance>
      </ogc:DWithin>
    </ogc:Filter>
  </wfs:Query>
</wfs:GetFeature>
```

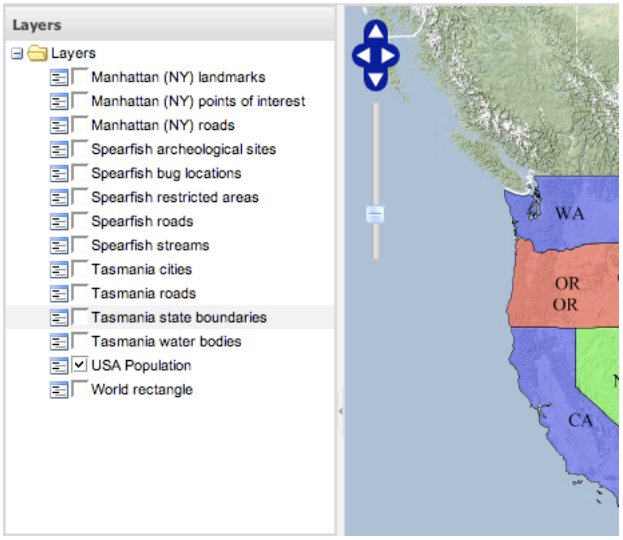
20.10 GeoExt Styler

20.10.1 Installation

1. Download the REST plugin for your version of GeoServer from the download page .
2. Unzip the archive into the WEB-INF/lib directory of the GeoServer installation.
3. Restart GeoServer
4. Download the GeoExt Styler extension from here (it says 1.7.3 but the version number doesn't matter. Soon there will be an updated release)
5. Unzip the archive into the *www/* directory of the GeoServer data directory.

20.10.2 Usage

1. Visit http://localhost:8080/geoserver/www/styler/index.html 2. Use the "Layers" panel to select a layer to style.



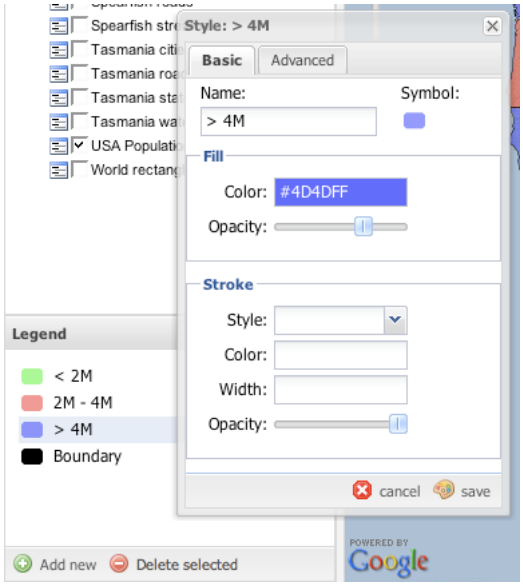
3. In the "Legend" panel select a rule by clicking on it.

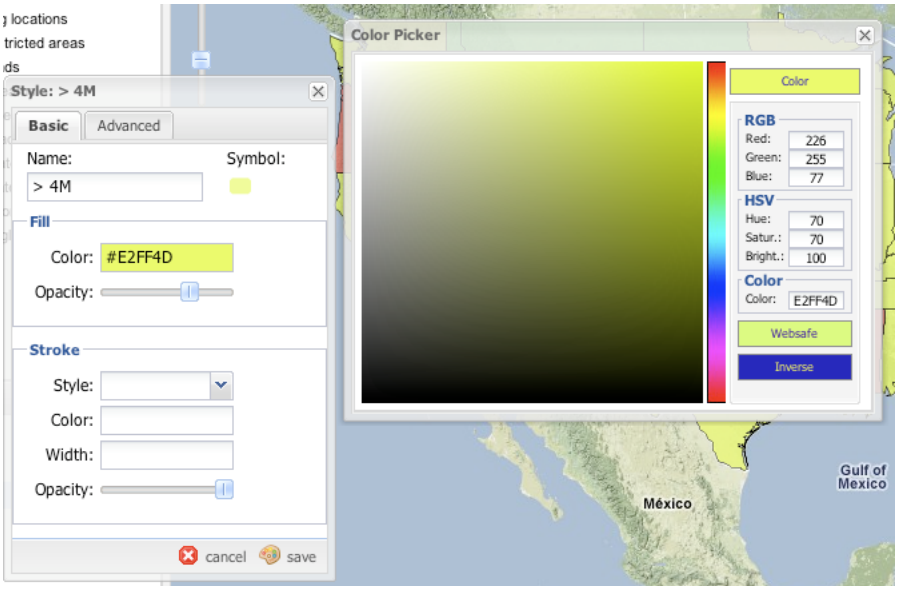
4. Change the color by clicking in the color box.

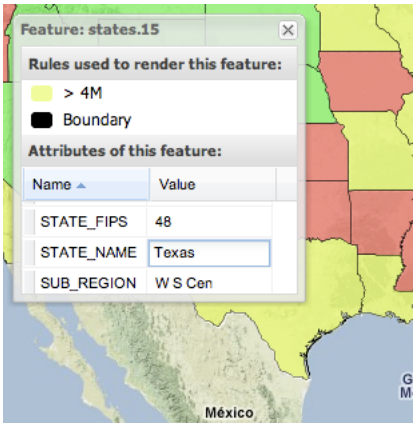
5. Click on a feature to view information about its attributes and which rules applied to it.

20.11 Web Processing Service

Web Processing Service (WPS) is an OGC service for the publishing of geospatial processes, algorithms, and calculations. WPS extends the web mapping server to provide geospatial analysis.


Style: > 4M
Basic
Advanced
Name:
Symbol:
> 4M
Fill
Color: #4D4DFF
Opacity:
Stroke
Style:
Color:
Width:
Opacity:
cancel
save
Legend
< 2M
2M - 4M
> 4M
Boundary
Add new
Delete selected
POWERED BY
Google


locations
tricted areas
ds
Style: > 4M
Basic
Advanced
Name:
Symbol:
> 4M
Fill
Color: #E2FF4D
Opacity:
Stroke
Style:
Color:
Width:
Opacity:
cancel
save
Color Picker
Color
RGB
Red: 226
Green: 255
Blue: 77
HSV
Hue: 70
Satur.: 70
Bright.: 100
Color
Color: E2FF4D
Websafe
Inverse
Gulf of Mexico
México


Feature: states.15
Rules used to render this feature:
> 4M
Boundary
Attributes of this feature:
Name
Value
STATE_FIPS
48
STATE_NAME
Texas
SUB_REGION
W S Cen
México

WPS is not a part of GeoServer by default, but is available as an extension.

The main advantage of GeoServer WPS over a standalone WPS is **direct integration** with other GeoServer services and the data catalog. This means that it is possible to create processes based on data served in GeoServer, as opposed to sending the entire data source in the request. It is also possible for the results of a process to be stored as a new layer in the GeoServer catalog. In this way, WPS acts as a full remote geospatial analysis tool, capable of reading and writing data from and to GeoServer.

For the official WPS specification, see http://www.opengeospatial.org/standards/wps.

20.11.1 Installing the WPS extension

The WPS module is not a part of GeoServer core, but instead must be installed as an extension. To install WPS:

1. Navigate to the GeoServer download page
2. Find the page that matches the version of the running GeoServer.

   Warning: Be sure to match the version of the extension with that of GeoServer, otherwise errors will occur.

3. Download the WPS extension. The download link for *WPS* will be in the *Extensions* section under *Other*.
4. Extract the files in this archive to the `WEB-INF/lib` directory of your GeoServer installation.
5. Restart GeoServer.

After restarting, load the *Web Administration Interface*. If the extension loaded properly, you should see an extra entry for WPS in the *Service Capabilities* column. If you don't see this entry, check the logs for errors.

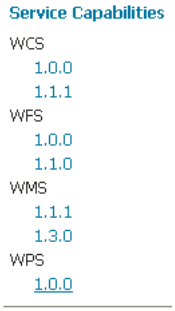


Figure 20.59: *A link for the WPS capabilities document will display if installed properly*

Configuring WPS

WPS processes are subject to the same feature limit as the WFS service. The limit applies to process **input**, so even processes which summarize data and return few results will be affected if applied to very large datasets. The limit is set on the *WFS* Admin page.

Warning: If the limit is encountered during process execution, no error is given. Any results computed by the process may be incomplete

20.11.2 WPS Operations

WPS defines three operations for the discovery and execution of geospatial processes. The operations are:

- GetCapabilities
- DescribeProcess
- Execute

GetCapabilities

The **GetCapabilities** operation requests details of the service offering, including service metadata and metadata describing the available processes. The response is an XML document called the **capabilities document**.

The required parameters, as in all OGC GetCapabilities requests, are `service=WPS`, `version=1.0.0` and `request=GetCapabilities`.

An example of a GetCapabilities request is:

```
http://localhost:8080/geoserver/ows?
  service=WPS&
  version=1.0.0&
  request=GetCapabilities
```

DescribeProcess

The **DescribeProcess** operation requests a description of a WPS process available through the service.

The parameter `identifier` specifies the process to describe. Multiple processes can be requested, separated by commas (for example, `identifier=JTS:buffer,gs:Clip`). At least one process must be specified.

Note: As with all OGC parameters, the keys (`request`, `version`, etc) are case-insensitive, and the values (`GetCapabilities`, `JTS:buffer`, etc.) are case-sensitive. GeoServer is generally more relaxed about case, but it is best to follow the specification.

The response is an XML document containing metadata about each requested process, including the following:

- Process name, title and abstract
- For each input and output parameter: identifier, title, abstract, multiplicity, and supported datatype and format

An example request for the process `JTS:buffer` is:

```
http://localhost:8080/geoserver/ows?
  service=WPS&
  version=1.0.0&
  request=DescribeProcess&
  identifier=JTS:buffer
```

The response XML document contains the following information:

Title	"Buffers a geometry using a certain distance"
Inputs	**geom**: "The geometry to be buffered" *(geometry, mandatory)* **distance**: "The distance (same unit of measure as the geometry)" *(double, mandatory)* **quadrant segments**: "Number of quadrant segments. Use > 0 for round joins, 0 for flat joins, < 0 for mitred joins" *(integer, optional)* **capstyle**: "The buffer cap style, round, flat, square" *(literal value, optional)*
Output formats	One of GML 3.1.1, GML 2.1.2, or WKT

Execute

The **Execute** operation is a request to perform the process with specified input values and required output data items. The request may be made as either a GET URL, or a POST with an XML request document. Because the request has a complex structure, the POST form is more typically used.

The inputs and outputs required for the request depend on the process being executed. GeoServer provides a wide variety of processes to process geometry, features, and coverage data. For more information see the section *WPS Processes*.

Below is an example of a `Execute` POST request. The example process (`JTS:buffer`) takes as input a geometry `geom` (in this case the point `POINT(0 0)`), a `distance` (with the value `10`), a quantization factor `quadrantSegments` (here set to be 1), and a `capStyle` (specified as `flat`). The `<ResponseForm>` element specifies the format for the single output `result` to be GML 3.1.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:Execute version="1.0.0" service="WPS" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xml
  <ows:Identifier>JTS:buffer</ows:Identifier>
  <wps:DataInputs>
    <wps:Input>
      <ows:Identifier>geom</ows:Identifier>
      <wps:Data>
        <wps:ComplexData mimeType="application/wkt"><![CDATA[POINT(0 0)]]></wps:ComplexData>
      </wps:Data>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>distance</ows:Identifier>
      <wps:Data>
        <wps:LiteralData>10</wps:LiteralData>
      </wps:Data>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>quadrantSegments</ows:Identifier>
      <wps:Data>
        <wps:LiteralData>1</wps:LiteralData>
      </wps:Data>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>capStyle</ows:Identifier>
      <wps:Data>
        <wps:LiteralData>flat</wps:LiteralData>
      </wps:Data>
    </wps:Input>
  </wps:DataInputs>
  <wps:ResponseForm>
    <wps:RawDataOutput mimeType="application/gml-3.1.1">
      <ows:Identifier>result</ows:Identifier>
    </wps:RawDataOutput>
```

```
  </wps:ResponseForm>
</wps:Execute>
```

The process performs a buffer operation using the supplied inputs, and returns the outputs as specified. The response from the request is (with numbers rounded for clarity):

```
<?xml version="1.0" encoding="utf-8"?>
<gml:Polygon xmlns:sch="http://www.ascc.net/xml/schematron"
 xmlns:gml="http://www.opengis.net/gml"
 xmlns:xlink="http://www.w3.org/1999/xlink">
  <gml:exterior>
    <gml:LinearRing>
      <gml:posList>
        10.0 0.0
        0.0 -10.0
        -10.0 0.0
        0.0 10.0
        10.0 0.0
      </gml:posList>
    </gml:LinearRing>
  </gml:exterior>
</gml:Polygon>
```

For help in generating WPS requests you can use the built-in interactive *WPS Request Builder*.

20.11.3 WPS Processes

The Web Processing Service describes a method for publishing geospatial processes, but does not specify what those processes should be. Servers that implement WPS therefore have complete leeway in what types of processes to implement, as well as how those processes are implemented. This means that a process request designed for one type of WPS is not expected to work on a different type of WPS.

GeoServer implements processes from two different categories:

- JTS Topology Suite processes
- GeoServer-specific processes

JTS Topology Suite processes

JTS Topology Suite is a Java library of functions for processing geometries in two dimensions. JTS conforms to the Simple Features Specification for SQL published by the Open Geospatial Consortium (OGC), similar to PostGIS. JTS includes common spatial functions such as area, buffer, intersection, and simplify.

GeoServer WPS implements some of these functions as processes. The names and definitions of these processes are subject to change, so they have not been included here. For a full list of JTS processes, please see the GeoServer *WPS capabilities document*.

GeoServer processes

GeoServer WPS includes a few processes created especially for use with GeoServer. These are usually GeoServer-specific functions, such as bounds and reprojection. They use an internal connection to the GeoServer WFS/WCS, not part of the WPS specification, for reading and writing data.

As with JTS, the names and definitions of these processes are subject to change, so they have not been included here. For a full list of GeoServer-specific processes, please see the GeoServer *WPS capabilities document*.

Process chaining

One of the benefits of WPS is its native ability to chain processes. Much like how functions can call other functions, a WPS process can use as its input the output of another process. Many complex functions can thus be combined in to a single powerful request.

For example, let's take some of the sample data that is shipped with GeoServer and use the WPS engine to chain a few of the built in processes, which will allow users to perform geospatial analysis on the fly.

The question we want to answer in this example is the following: How many miles of roads are crossing a protected area?

The data that will be used for this example is included with a standard installation of GeoServer:

- *sf:roads*: the layer that contains road information
- *sf:restricted*: the layer representing restricted areas

The restricted areas partially overlap the roads. We would like to know the total length of roads inside the restricted areas, as shown in the next screenshot. The road network is represented in white against a false color DEM (Digital Elevation Model). The restricted areas are represented with a dashed line in dark brown. The portion of the road network that is inside the restricted areas is drawn in red.

In order to calculate the total length, we will need the following built in WPS processes:

- `gs:IntersectionFeatureCollection`: returns the intersection between two feature collections adding the attributes from both of them
- `gs:CollectGeometries`: collects all the default geometries in a feature collection and returns them as a single geometry collection
- `JTS:length`: calculates the length of a geometry in the same unit of measure as the geometry

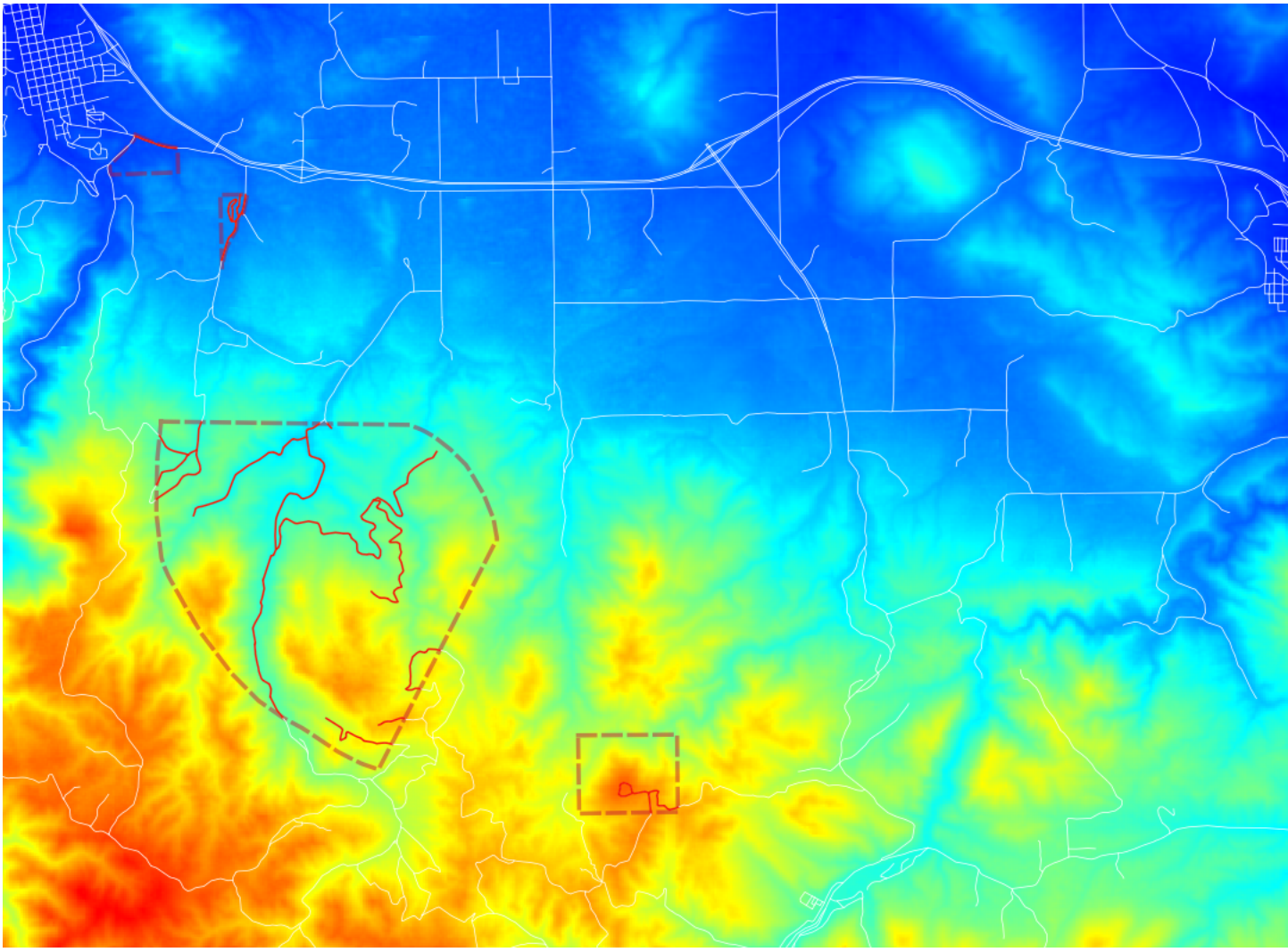
The sequence in which these processes are executed is important. The first thing we want to do is interesect the road network with the restricted areas. This gives us the feature collection with all the roads that we are interested in. Then we collect those geometries into a single GeometryCollection so that the length can be calculated with the built in JTS algorithm.

gs:IntersectionFeatureCollection –> gs:CollectGeometries –> JTS:length

The sequence of processes determines how the WPS request is built, by embedding the first process into the second, the second into the third, etc. A process produces some output which will become the input of the next process, resulting in a processing pipeline that can solve complex spatial analysis with a single HTTP request. The advantage of using GeoServer's layers is that data is not being shipped back and forth between processes, resulting in very good performance.

Here is the complete WPS request in XML format:

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:Execute version="1.0.0" service="WPS" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns

  <ows:Identifier>JTS:length</ows:Identifier>
  <wps:DataInputs>
    <wps:Input>
      <ows:Identifier>geom</ows:Identifier>
      <wps:Reference mimeType="text/xml; subtype=gml/3.1.1"
                     xlink:href="http://geoserver/wps" method="POST">
```



```
<wps:Body>
  <wps:Execute version="1.0.0" service="WPS">
    <ows:Identifier>gs:CollectGeometries</ows:Identifier>
    <wps:DataInputs>
      <wps:Input>
        <ows:Identifier>features</ows:Identifier>
        <wps:Reference mimeType="text/xml; subtype=wfs-collection/1.0" xlink:href="http://geose
          <wps:Body>
            <wps:Execute version="1.0.0" service="WPS">
              <ows:Identifier>gs:IntersectionFeatureCollection</ows:Identifier>
              <wps:DataInputs>
                <wps:Input>
                  <ows:Identifier>first feature collection</ows:Identifier>
                  <wps:Reference mimeType="text/xml; subtype=wfs-collection/1.0" xlink:href="ht
                    <wps:Body>
                      <wfs:GetFeature service="WFS" version="1.0.0" outputFormat="GML2">
                        <wfs:Query typeName="sf:roads"/>
                      </wfs:GetFeature>
                    </wps:Body>
                  </wps:Reference>
                </wps:Input>
                <wps:Input>
                  <ows:Identifier>second feature collection</ows:Identifier>
                  <wps:Reference mimeType="text/xml; subtype=wfs-collection/1.0" xlink:href="ht
                    <wps:Body>
                      <wfs:GetFeature service="WFS" version="1.0.0" outputFormat="GML2">
                        <wfs:Query typeName="sf:restricted"/>
                      </wfs:GetFeature>
                    </wps:Body>
                  </wps:Reference>
                </wps:Input>
                <wps:Input>
                  <ows:Identifier>first attributes to retain</ows:Identifier>
                  <wps:Data>
                    <wps:LiteralData>the_geom cat</wps:LiteralData>
                  </wps:Data>
                </wps:Input>
                <wps:Input>
                  <ows:Identifier>second attributes to retain</ows:Identifier>
                  <wps:Data>
                    <wps:LiteralData>cat</wps:LiteralData>
                  </wps:Data>
                </wps:Input>
              </wps:DataInputs>
              <wps:ResponseForm>
                <wps:RawDataOutput mimeType="text/xml;
                                   subtype=wfs-collection/1.0">
                  <ows:Identifier>result</ows:Identifier>
                </wps:RawDataOutput>
              </wps:ResponseForm>
            </wps:Execute>
          </wps:Body>
        </wps:Reference>
      </wps:Input>
    </wps:DataInputs>
    <wps:ResponseForm>
      <wps:RawDataOutput mimeType="text/xml; subtype=gml/3.1.1">
        <ows:Identifier>result</ows:Identifier>
```

```
            </wps:RawDataOutput>
          </wps:ResponseForm>
        </wps:Execute>
      </wps:Body>
    </wps:Reference>
    </wps:Input>
  </wps:DataInputs>
  <wps:ResponseForm>
    <wps:RawDataOutput>
      <ows:Identifier>result</ows:Identifier>
    </wps:RawDataOutput>
  </wps:ResponseForm>
</wps:Execute>
```

You can save this XML request in a file called wps-chaining.xml and execute the request using cURL like this:

curl -u admin:geoserver -H 'Content-type: xml' -XPOST -d@'wps-chaining.xml' http://localhost:8080/geoserver/wps

The response is just a number, the total length of the roads that intersect the restricted areas, and should be around *25076.285* meters (the length process returns map units)

To see WPS requests in action, you can use the built-in *WPS Request Builder*.

20.11.4 WPS Request Builder

The GeoServer WPS extension includes a request builder for testing out various WPS processes through the *Web Administration Interface*.

Accessing the request builder

To access the WPS Request Builder:

1. Navigate to the main *Web Administration Interface*.
2. Click on the *Demos* link on the left side.
3. Select *WPS Request Builder* from the list of demos.

Figure 20.60: *WPS request builder in the list of demos*

Using the request builder

The WPS Request Builder primarily consists of a selection box listing all of the available processes, and two buttons, one to submit the WPS request, and another to display what the POST request looks like.

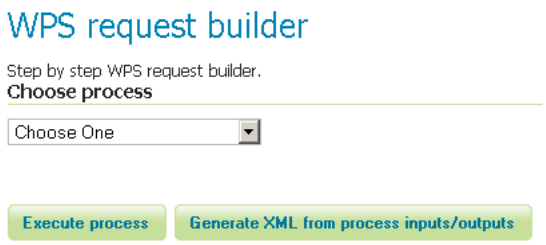


Figure 20.61: *Blank WPS request builder form*

The display changes depending on the process and input selected. JTS processes have available as inputs any of a GML/WKT-based feature collection, URL reference, or subprocess. GeoServer-specific processes have all these as options and also includes the ability to choose a GeoServer layer as input.

For each process, a form will display based on the required and optional parameters associated with that process, if any.

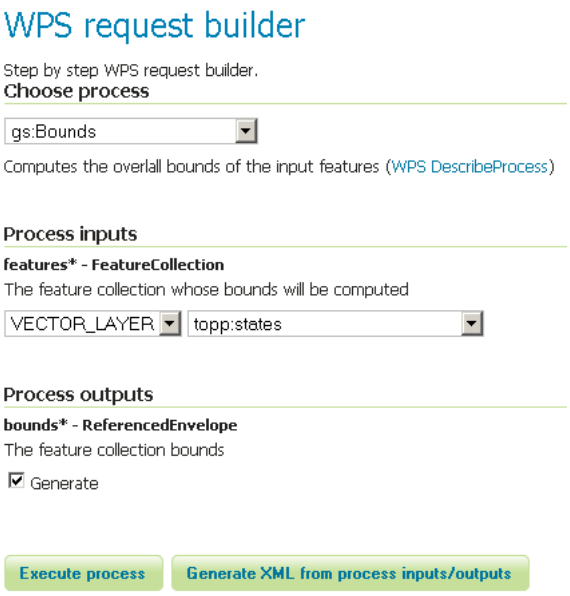


Figure 20.62: *WPS request builder form to determine the bounds of topp:states*

To see the process as a POST request, click the *Generate XML from process inputs/outputs* button.

To execute the process, click the *Execute Process* button. The response will be displayed in a window or

20.12 XSLT WFS output format module

The `xslt` module for GeoServer is a WFS output format generator which brings togheter a base output, such as GML, and a XSLT 1.0 style sheet to generate a new textual output format of user choosing.

The configuration for this output format can be either performed directly on disk, or via a REST API.

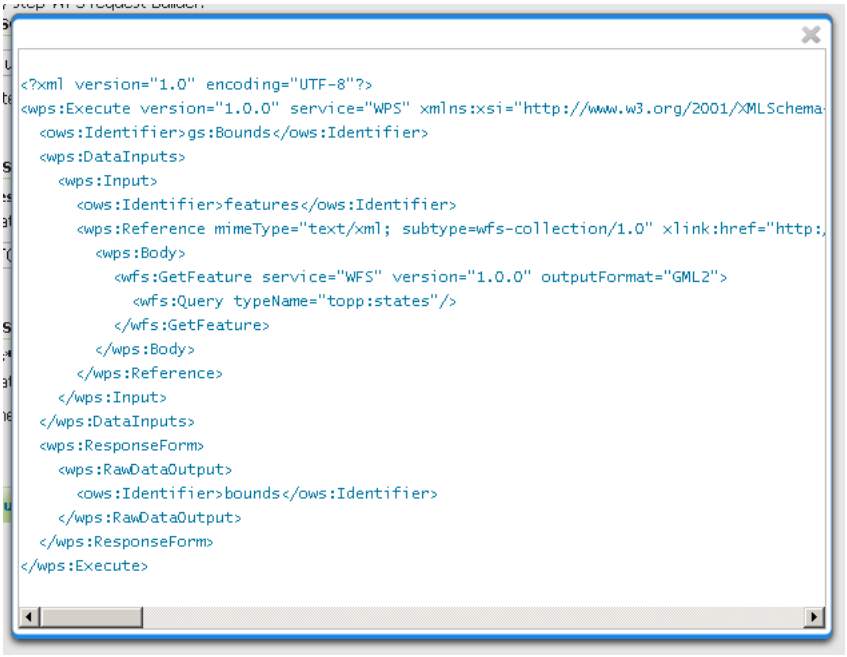


Figure 20.63: *Raw WPS POST request for the above process*

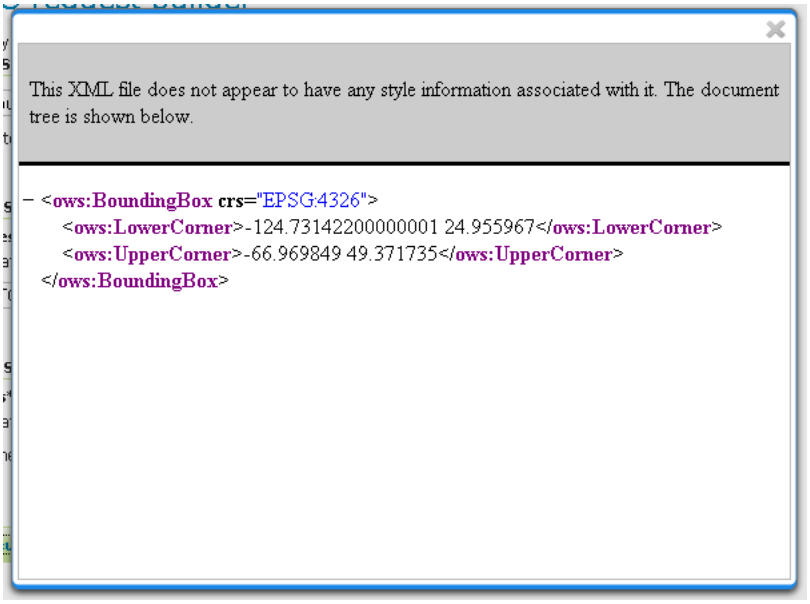


Figure 20.64: *WPS server response*

20.12.1 Manual configuration

All the configuration for the output resides in the `$GEOSERVER_DATA_DIR/wfs/transform` folder, which is going to be created on startup when missing if the XSLT output format has been installed in GeoServer.

Each XSLT transformation must be configured with its own xml file in the `$GEOSERVER_DATA_DIR/wfs/transform` folder, which in turn points to a xslt file for the transformation. While the names can be freeform, it is suggested to follow a simple naming convention:

- <mytransformation>.xml for the xml config file
- <mytransformation>.xslt for the xslt tile

Transformations can be either global, and thus applicable to any feature type, or type specific, in which case the transformation knows about the specific attributes of the transformed feature type.

20.12.2 Global transformation example

Here is an example of a global transformation setup. The `$GEOSERVER_DATA_DIR/wfs/transform/global.xml` file will look like:

```
<transform>
  <sourceFormat>text/xml; subtype=gml/2.1.2</sourceFormat>
  <outputFormat>HTML</outputFormat>
  <outputMimeType>text/html</outputMimeType>
  <fileExtension>html</fileExtension>
  <xslt>global.xslt</xslt>
</transform>
```

Here is an explanation of each element:

- `sourceFormat` (mandatory): the output format used as the source of the XSLT transformation
- `outputFormat` (mandatory): the output format generated by the transformation
- `outputMimeType` (optional): the mime type for the generated output. In case it's missing, the `outputFormat` is assumed to be a mime type and used for the purpose.
- `fileExtension` (optional): the file extension for the generated output. In case it's missing `txt` will be used.
- `xslt` (mandatory): the name of XSLT 1.0 style sheet used for the transformation

The associated XSLT file will be `$GEOSERVER_DATA_DIR/wfs/transform/global.xslt` folder, and it will be able to transform any GML2 input into a corresponding HTML file. Here is an example:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:wfs="http://www.opengis.net/wfs"
  xmlns:tiger="http://www.census.gov" xmlns:gml="http://www.opengis.net/gml"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <body>
      <xsl:for-each select="wfs:FeatureCollection/gml:featureMember/*">
        <h2><xsl:value-of select="@fid"/></h2>
        <table border="1">
          <tr>
            <th>Attribute</th>
            <th>Value</th>
```

```
          </tr>
            <!-- [not(*)] strips away all nodes having
                 children, in particular, geometries -->
            <xsl:for-each select="./*[not(*)]">
            <tr>
              <td>
                <xsl:value-of select="name()" />
              </td>
              <td>
                <xsl:value-of select="." />
              </td>
            </tr>
            </xsl:for-each>
        </table>
      </xsl:for-each>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

20.12.3 Type specific transformations

Type specific transformations can refer to a specific type and leverage its attributes directly. While not required, it is good practice to setup a global transformation that can handle any feature type (since the output format is declared in the capabilities document as being general, not type specific) and then override it for specific feature types in order to create special transformations for them.

Here is an example of a transformation declaration that is type specific, that will be located at `$GEOSERVER_DATA_DIR/wfs/transform/html_bridges.xml`

```
<transform>
  <sourceFormat>text/xml; subtype=gml/2.1.2</sourceFormat>
  <outputFormat>HTML</outputFormat>
  <outputMimeType>text/html</outputMimeType>
  <fileExtension>html</fileExtension>
  <xslt>html_bridges.xslt</xslt>
  <featureType>
    <id>cite:Bridges</id>
  </featureType>
</transform>
```

The extra `featureType` element associates the transformation to the specific feature type

The associated xslt file will be located at `$GEOSERVER_DATA_DIR/wfs/transform/html_bridges.xslt` and will leveraging knowlegde about the input attributes:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:wfs="http://www.opengis.net/wfs"
  xmlns:cite="http://www.opengis.net/cite" xmlns:gml="http://www.opengis.net/gml"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <body>
        <h2>Bridges</h2>
        <xsl:for-each
            select="wfs:FeatureCollection/gml:featureMember/cite:Bridges">
        <ul>
```

```
          <li>ID: <xsl:value-of select="@fid" /></li>
          <li>FID: <xsl:value-of select="cite:FID" /></li>
          <li>Name: <xsl:value-of select="cite:NAME" /></li>
        </ul>
        <p/>
        </xsl:for-each>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Note: While writing the XSLT always remember to declare all prefixes used in the sheet in the `stylesheet` element, otherwise you might encounter hard to understand error messages

A specific feature type can be associated to multiple output formats. While uncommon, the same xslt file can also be associated to multiple feature types by creating multiple xml configuration files, and associating a different feature type in each.

20.12.4 Rest configuration

Transformations can be created, updated and deleted via the REST api (normally, this requires administrator privileges). Each transformation is represented with the same XML format used on disk, but with two variants:

- a new `name` attribute appears, which matches the XML file name
- the `featureType` element contains also a link to the resource representing the feature type in the REST config tree

For example:

```
<transform>
  <name>test</name>
  <sourceFormat>text/xml; subtype=gml/2.1.2</sourceFormat>
  <outputFormat>text/html</outputFormat>
  <fileExtension>html</fileExtension>
  <xslt>test-tx.xslt</xslt>
  <featureType>
    <name>tiger:poi</name>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" href="http://localhost:8080/g
  </featureType>
</transform>
```

Here is a list of resources and the HTTP methods that can be used on them.

`/rest/services/wfs/transforms[.<format>]`

Method	Action	Return Code	Formats	Default Format	Parameters
GET	List all available transforms	200	HTML, XML, JSON	HTML	
POST	Add a new transformation	201, with `Location` header	XML, JSON		name, sourceFormat, outputFormat, outputMimeType
PUT	Update global settings	200	XML, JSON		
DELETE		405			

The `POST` method can be used to create a transformation in two ways:

- if the content type used is `application/xml` the server will assume a `<transform>` definition is being posted, and the XSLT will have to be uploaded separately using a `PUT` request with content type `application/xslt+xml` against the transformation resource
- if the content type used is `application/xslt+xml` the server will assume the XSLT itself is being posted, and the `name`, `sourceFormat`, `outputFormat`, `outputMimeType` query parameters will be used to fill in the transform configuration instead

`/rest/services/wfs/transforms/<transform>[.<format>]`

Method	Action	Return Code	Formats	Default Format
GET	Returns the transformation	200	HTML, XML, XSLT	HTML
POST		405		
PUT	Updates either the transformation configuration, or its XSLT, depending on the mime type used	200	XML, XSLT	
DELETE	Deletes the transformation	200		

The PUT operation behaves differently depending on the content type used in the request:

- if the content type used is `application/xml` the server will assume a `<transform>` definition is being sent and will update it
- if the content type used is `application/xslt+xml` the server will assume the XSLT itself is being posted, as such the configuration won't be modified, but the XSLT associated to it will be overwritten instead

20.13 Catalog Services for the Web (CSW)

This section discusses the Catalog Services for Web (CSW) community module for GeoServer. With this module, GeoServer supports retrieving and displaying items from the GeoServer catalog using the CSW service.

For more information on CSW, please refer to OGC OpenGIS Implementation Specification 07-006r1 and the OGC tutorial on CSW.

20.13.1 Installing Catalog Services for Web (CSW)

To install the CSW module:

1. Download the module. The file name is called `geoserver-*-csw-plugin.zip`, where `*` is the version/snapshot name.
2. Extract this file and place the JARs in `WEB-INF/lib`.
3. Perform any configuration required by your servlet container, and then restart.
4. Verify that the module was installed correctly by going to the Welcome page of the *Web Administration Interface* and seeing that *CSW* is listed in the *Service Capabilities* list.

20.13.2 Catalog Services for the Web (CSW) features

Supported operations

The following standard CSW operations are currently supported:

- GetCapabilities
- GetRecords
- GetRecordById
- GetDomain
- DescribeRecord

The Internal Catalog Store supports filtering on both full x-paths as well as the "Queryables" specified in GetCapabilities.

Catalog stores

The default catalog store is the Internal Catalog Store, which retrieves information from the GeoServer's internal catalog. The Simple Catalog Store (`simple-store` module) adds an alternative simple store which reads the catalog data directly from files (mainly used for testing).

If there are multiple catalog stores present (for example, when the Simple Catalog Store module is loaded), set the Java system property `DefaultCatalogStore` to make sure that the correct catalog store will be used. To use the Internal Catalog Store, this property must be set to:

```
DefaultCatalogStore=org.geoserver.csw.store.internal.GeoServerInternalCatalogStore
```

To use the Simple Catalog Store:

```
DefaultCatalogStore=org.geoserver.csw.store.simple.GeoServerSimpleCatalogStore
```

Supported schemes

The Internal Catalog Store supports two metadata schemes:

- Dublin Core
- ISO Metadata Profile

Mapping Files

Mapping files are located in the `csw` directory inside the *GeoServer Data Directory*. Each mapping file must have the exact name of the record type name combined with the `.properties` extension. For example:

- Dublin Core mapping can be found in the file `csw/Record.properties` inside the data directory.
- ISO Metadata mapping can be found in the file `csw/MD_Metadata.properties` inside the data directory.

The mapping files take the syntax from Java properties files. The left side of the equals sign specifies the target field name or path in the metadata record, paths being separated with dots. The right side of the equals sign specifies any CQL expression that denotes the value of the target property. The CQL expression is applied to each ResourceInfo object in the catalog and can retrieve all properties from this object. These expressions can make use of literals, properties present in the ResourceInfo object, and all normal CQL operators and functions. There is also support for complex datastructures such as Maps using the dot notation and Lists using the bracket notation (Example mapping files are given below).

The properties in the ResourceInfo object that can be used are:

```
name
qualifiedName
nativeName
qualifiedNativeName
alias
title
abstract
description
metadata.?
namespace
namespace.prefix
namespace.name
namespace.uri
namespace.metadata.?
keywords
keywords[?]
keywords[?].value
keywords[?].language
keywords[?].vocabulary
keywordValues
keywordValues[?]
metadataLinks
metadataLinks[?]
metadataLinks[?].id
metadataLinks[?].about
metadataLinks[?].metadataType
metadataLinks[?].type
metadataLinks[?].content
latLonBoundingBox
latLonBoundingBox.dimension
latLonBoundingBox.lowerCorner
latLonBoundingBox.upperCorner
nativeBoundingBox
nativeBoundingBox.dimension
nativeBoundingBox.lowerCorner
nativeBoundingBox.upperCorner
srs
nativeCrs
projectionPolicy
```

```
enabled
advertised
catalog.defaultNamespace
catalog.defaultWorkspace
store.name
store.description
store.type
store.metadata.?
store.enabled
store.workspace
store.workspace.name
store.metadata.?
store.connectionParameters.?
store.error
```

Depending on whether the resource is a FeatureTypeInfo or a CoverageInfo, additional properties may be taken from their respective object structure. You may use *REST configuration* to view an xml model of feature types and datastores in which the xml tags represent the available properties in the objects.

Some fields in the metadata schemes can have multiple occurences. They may be mapped to properties in the Catalog model that are also multi-valued, such as for example `keywords`. It is also possible to use a filter function called `list` to map multiple single-valued or multi-valued catalog properties to a MetaData field with multiple occurences (see in ISO MetaData Profile example, mapping for the `identificationInfo.AbstractMD_Identification.citation.CI_Citation.alternateTitle` field).

Placing the `@` symbol in front of the field will set that to use as identifier for each metadata record. This may be useful for ID filters. Use a `$` sign in front of fields that are required to make sure the mapping is aware of the requirement (specifically for the purpose of property selection).

Dublin Core

Below is an example of a Dublin Core mapping file:

```
@identifier.value=id
title.value=title
creator.value='GeoServer Catalog'
subject.value=keywords
subject.scheme='http://www.digest.org/2.1'
abstract.value=abstract
description.value=strConcat('description about ' , title)
date.value="metadata.date"
type.value='http://purl.org/dc/dcmitype/Dataset'
publisher.value='Niels Charlier'
#format.value=
#language.value=
#coverage.value=
#source.value=
#relation.value=
#rights.value=
#contributor.value=
```

All fields have the form of `<fieldname>.value` for the actual value in the field. Additionally `<fieldname>.scheme` can be specified for the `@scheme` attribute of this field.

Examples of attributes extracted from the ResourceInfo are `id`, `title`, and `keywords`, etc. The attribute `metadata.date` uses the metadata (`java.util.`)Map from the Resource object. In this map, it searches for the keyword "date".

Note that double quotes are necessary in order to preserve this meaning of the dots.

ISO Metadata Profile

Below is an example of an ISO Metadata Profile Mapping File:

```
@fileIdentifier.CharacterString=id
identificationInfo.AbstractMD_Identification.citation.CI_Citation.title.CharacterString=title
identificationInfo.AbstractMD_Identification.citation.CI_Citation.alternateTitle.CharacterString=list
identificationInfo.AbstractMD_Identification.descriptiveKeywords.MD_Keywords.keyword.CharacterString=
identificationInfo.AbstractMD_Identification.abstract.CharacterString=abstract
$dateStamp.Date= if_then_else ( isNull("metadata.date") , 'Unknown', "metadata.date")
hierarchyLevel.MD_ScopeCode.@codeListValue='http://purl.org/dc/dcmitype/Dataset'
$contact.CI_ResponsibleParty.individualName.CharacterString=
```

The full path of each field must be specified (separated with dots). XML attributes are specified with the `@` symbol, similar to the usual XML X-path notation.

To keep the result XSD compliant, the parameters `dateStamp.Date` and `contact.CI_ResponsibleParty.individualName.CharacterString` must be preceded by a `$` sign to make sure that they are always included even when using property selection.

For more information on the ISO Metadata standard, please see the OGC Implementation Specification 07-045.

20.13.3 Catalog Services for the Web (CSW) tutorial

This tutorial will show how to use the CSW module. It assumes a fresh installation of GeoServer with the *CSW module installed*.

Configuration

In the `<data_dir>/csw` directory, create a new file named `MD_Metadata` (ISO Metadata Profile mapping file) with the following contents:

```
@fileIdentifier.CharacterString=prefixedName
identificationInfo.AbstractMD_Identification.citation.CI_Citation.title.CharacterString=title
identificationInfo.AbstractMD_Identification.descriptiveKeywords.MD_Keywords.keyword.CharacterString=
identificationInfo.AbstractMD_Identification.abstract.CharacterString=abstract
$dateStamp.Date= if_then_else ( isNull("metadata.date") , 'Unknown', "metadata.date")
hierarchyLevel.MD_ScopeCode.@codeListValue='http://purl.org/dc/dcmitype/Dataset'
$contact.CI_ResponsibleParty.individualName.CharacterString='John Smith'
```

Services

With GeoServer running (and responding on `http://localhost:8080`), test GeoServer CSW in a web browser by querying the CSW capabilities as follows:

```
http://localhost:8080/geoserver/csw?service=csw&version=2.0.2&request=GetCapabilities
```

We can request a description of our Metadata record:

```
http://localhost:8080/geoserver/csw?service=CSW&version=2.0.2&request=DescribeRecord&typeName=gmd:MD_
```

This yields the following result:

```
<?xml version="1.0" encoding="UTF-8"?>
<csw:DescribeRecordResponse xmlns:csw="http://www.opengis.net/cat/csw/2.0.2" xmlns:xsi="http://www.w3
<csw:SchemaComponent targetNamespace="http://www.opengis.net/cat/csw/2.0.2" schemaLanguage="http://ww
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xlink="http://www.w3.org/1999/xlink" xml
      <!-- ================================= Annotation ================================= -->
      <xs:annotation>
              <xs:documentation>Geographic MetaData (GMD) extensible markup language is a component o
      </xs:annotation>
      ...
```

Query all layers as follows:

```
http://localhost:8080/geoserver/csw?service=CSW&version=2.0.2&request=GetRecords&typeNames=gmd:MD_Met
```

Request a particular layer by ID...:

```
http://localhost:8080/geoserver/csw?service=CSW&version=2.0.2&request=GetRecordById&elementsetname=su
```

...or use a filter to retrieve it by Title:

```
http://localhost:8080/geoserver/csw?service=CSW&version=2.0.2&request=GetRecords&typeNames=gmd:MD_Met
```

Either case should return:

```
<?xml version="1.0" encoding="UTF-8"?>
<csw:GetRecordsResponse xmlns:xml="http://www.w3.org/XML/1998/namespace" xmlns="http://www.opengis.ne
  <csw:SearchStatus timestamp="2013-06-28T13:41:43.090Z"/>
  <csw:SearchResults numberOfRecordsMatched="1" numberOfRecordsReturned="1" nextRecord="0" recordSche
    <gmd:MD_Metadata>
      <gmd:fileIdentifier>
        <gco:CharacterString>CoverageInfoImpl--4a9eec43:132d48aac79:-8000</gco:CharacterString>
      </gmd:fileIdentifier>
      <gmd:dateStamp>
        <gco:Date>Unknown</gco:Date>
      </gmd:dateStamp>
      <gmd:identificationInfo>
        <gmd:MD_DataIdentification>
          <gmd:extent>
            <gmd:EX_Extent>
              <gmd:geographicElement>
                <gmd:EX_GeographicBoundingBox crs="urn:x-ogc:def:crs:EPSG:6.11:4326">
                  <gmd:westBoundLongitude>36.492</gmd:westBoundLongitude>
                  <gmd:southBoundLatitude>6.346</gmd:southBoundLatitude>
                  <gmd:eastBoundLongitude>46.591</gmd:eastBoundLongitude>
                  <gmd:northBoundLatitude>20.83</gmd:northBoundLatitude>
                </gmd:EX_GeographicBoundingBox>
              </gmd:geographicElement>
            </gmd:EX_Extent>
          </gmd:extent>
        </gmd:MD_DataIdentification>
        <gmd:AbstractMD_Identification>
          <gmd:citation>
            <gmd:CI_Citation>
              <gmd:title>
                <gco:CharacterString>mosaic</gco:CharacterString>
              </gmd:title>
            </gmd:CI_Citation>
          </gmd:citation>
```

```
          <gmd:descriptiveKeywords>
            <gmd:MD_Keywords>
              <gmd:keyword>
                <gco:CharacterString>WCS</gco:CharacterString>
              </gmd:keyword>
              <gmd:keyword>
                <gco:CharacterString>ImageMosaic</gco:CharacterString>
              </gmd:keyword>
              <gmd:keyword>
                <gco:CharacterString>mosaic</gco:CharacterString>
              </gmd:keyword>
            </gmd:MD_Keywords>
          </gmd:descriptiveKeywords>
        </gmd:AbstractMD_Identification>
      </gmd:identificationInfo>
      <gmd:contact>
        <gmd:CI_ResponsibleParty>
          <gmd:individualName>
            <gco:CharacterString>John Smith</gco:CharacterString>
          </gmd:individualName>
        </gmd:CI_ResponsibleParty>
      </gmd:contact>
      <gmd:hierarchyLevel>
        <gmd:MD_ScopeCode codeListValue="http://purl.org/dc/dcmitype/Dataset"/>
      </gmd:hierarchyLevel>
    </gmd:MD_Metadata>
  </csw:SearchResults>
</csw:GetRecordsResponse>
```

We can request the domain of a property. For example, all values of "Title":

```
http://localhost:8080/geoserver/csw?service=csw&version=2.0.2&request=GetDomain&propertyName=Title
```

This should yield the following result:

```
<?xml version="1.0" encoding="UTF-8"?>
<csw:GetDomainResponse xmlns:csw="http://www.opengis.net/cat/csw/2.0.2" xmlns:dc="http://purl.org/dc/
  <csw:DomainValues type="csw:Record">
      <csw:PropertyName>Title</csw:PropertyName>
      <csw:ListOfValues>
        <csw:Value>A sample ArcGrid file</csw:Value>
        <csw:Value>Manhattan (NY) landmarks</csw:Value>
        <csw:Value>Manhattan (NY) points of interest</csw:Value>
        <csw:Value>Manhattan (NY) roads</csw:Value>
        <csw:Value>North America sample imagery</csw:Value>
        <csw:Value>Pk50095 is a A raster file accompanied by a spatial data file</csw:Value>
        <csw:Value>Spearfish archeological sites</csw:Value>
        <csw:Value>Spearfish bug locations</csw:Value>
        <csw:Value>Spearfish restricted areas</csw:Value>
        <csw:Value>Spearfish roads</csw:Value>
        <csw:Value>Spearfish streams</csw:Value>
        <csw:Value>Tasmania cities</csw:Value>
        <csw:Value>Tasmania roads</csw:Value>
        <csw:Value>Tasmania state boundaries</csw:Value>
        <csw:Value>Tasmania water bodies</csw:Value>
        <csw:Value>USA Population</csw:Value>
        <csw:Value>World rectangle</csw:Value>
        <csw:Value>mosaic</csw:Value>
        <csw:Value>sfdem is a Tagged Image File Format with Geographic information</csw:Value>
```

```
      </csw:ListOfValues>
  </csw:DomainValues>
</csw:GetDomainResponse>
```

CHAPTER 21

Tutorials

21.1 Freemarker Templates

21.1.1 Introduction

This tutorial will introduce you to a more in depth view of what FreeMarker templates are and how you can use the data provided to templates by GeoServer.

Freemarker is a simple yet powerful template engine that GeoServer uses whenever developer allowed user customization of outputs. In particular, at the time of writing it's used to allow customization of GetFeatureInfo, GeoRSS and KML outputs.

Freemarker allows for simple variable expansions, as in `${myVarName}`, expansion of nested properties, such as in `${feature.myAtt.value}`, up to little programs using loops, ifs and variables. Most of the relevant information about how to approach template writing is included in the Freemarker's Designer guide and won't be repeated here: the guide, along with the *KML Placemark Templates* and *GetFeatureInfo Templates* tutorials should be good enough to give you a good grip on how a template is built.

Template Lookup

Geoserver looks up templates in three different places, allowing for various level of customization. For example given the `content.ftl` template used to generate WMS GetFeatureInfo content:

- Look into `GEOSERVER_DATA_DIR/workspaces/<workspace>/<datastore>/<featuretype>/content.ftl` to see if there is a feature type specific template
- Look into `GEOSERVER_DATA_DIR/workspaces/<workspace>/<datastore>/content.ftl` to see if there is a store specific template
- Look into `GEOSERVER_DATA_DIR/workspaces/<workspace>/content.ftl` to see if there is a workspace specific template
- Look into `GEOSERVER_DATA_DIR/workspaces/content.ftl` looking for a global override
- Look into `GEOSERVER_DATA_DIR/templates/content.ftl` looking for a global override
- Look into the GeoServer classpath and load the default template

Each templated output format tutorial should provide you with the template names, and state whether the templates can be type specific, or not. Missing the source for the default template, look up for the service jar in the geoserver distribution (for example, wms-x.y.z.jar), unpack it, and you'll find the actual xxx.ftl files GeoServer is using as the default templates.

Common Data Models

Freemarker calls "data model" the set of data provided to the template. Each output format used by Geoserver will inject a different data model according to the informations it's managing, yet there are three very common elements that appear in almost each template, Feature, FeatureType and FeatureCollection. Here we provide a data model of each.

The data model is a sort of a tree, where each element has a name and a type. Besides basic types, we'll use:

- list: a flat list of items that you can scan thru using the FreeMarker `<#list>` directive;
- map: a key/value map, that you usually access using the dot notation, as in `${myMap.myKey}`, and can be nested;
- listMap: a special construct that is, at the same time, a Map, and a list of the values.

Here are the three data models (as you can see there are redundancies, in particular in attributes, we chose this approach to make template building easier):

FeatureType (map)

- name (string): the type name
- attributes (listMap): the type attributes
  - name (string): attribute name
  - namespace (string): attribute namespace URI
  - prefix (string): attribute namespace prefix
  - type (string): attribute type, the fully qualified Java class name
  - isGeometry (boolean): true if the attribute is geometric, false otherwise

Feature (map)

- fid (string): the feature ID (WFS feature id)
- typeName (string): the type name
- attributes (listMap): the list of attributes (both data and metadata)
  - name (string): attribute name
  - namespace (string): attribute namespace URI
  - prefix (string): attribute namespace prefix
  - isGeometry (boolean): true if the attribute is geometric, false otherwise
  - value: a string representation of the the attribute value
  - isComplex (boolean): true if the attribute is a feature (see *Complex Features*), false otherwise
  - type (string or FeatureType): attribute type: if isComplex is false, the fully qualified Java class name; if isComplex is true, a FeatureType
  - rawValue: the actual attribute value (is isComplex is true rawValue is a Feature)
- type (map)
  - name (string): the type name (same as typeName)
  - namespace (string): attribute namespace URI
  - prefix (string): attribute namespace prefix
  - title (string): The title configured in the admin console

- abstract (string): The abstract for the type
- description (string): The description for the type
- keywords (list): The keywords for the type
- metadataLinks (list): The metadata URLs for the type
- SRS (string): The layer's SRS
- nativeCRS (string): The layer's coordinate reference system as WKT

FeatureCollection (map)

- features (list of Feature, see above)
- type (FeatureType, see above)

21.2 GeoRSS

GeoServer supports GeoRSS as an output format allowing you to serve features as an RSS feed.

21.2.1 Quick Start

If you are using a web browser which can render rss feeds simply visit the url http://localhost:8080/geoserver/wms/reflect?layers=states&format=rss in your browser. This is assuming a local GeoServer instance is running with an out of the box configuration. You should see a result that looks more or less like this:

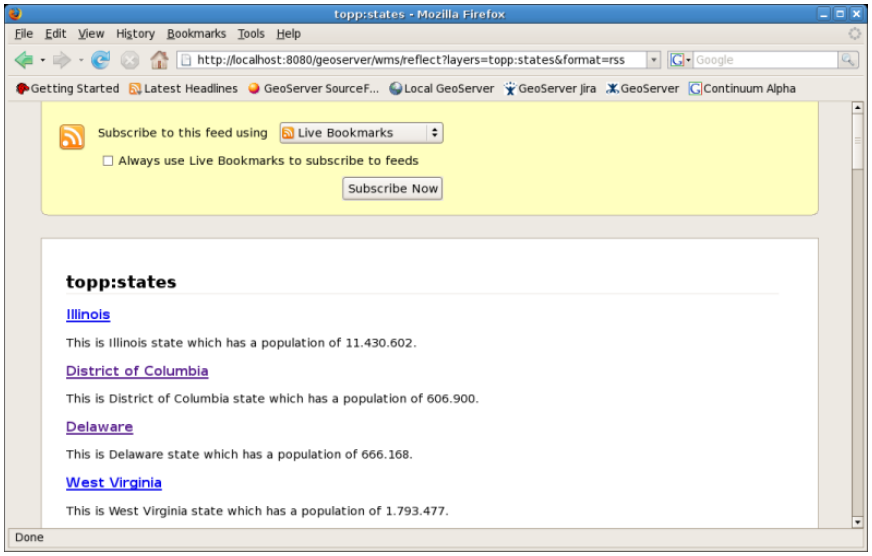


Figure 21.1: *topp:states rss feed*

21.2.2 Ajax Map Mashups

Note: For Ajax map mashups to work, the GeoServer instance must be visible to the Internet (i.e. using the address `localhost` will not work).

21.2.3 Google Maps

How to create a Google Maps mashup with a GeoRSS overlay produced by GeoServer.

1. Obtain a Google Maps API Key from Google.
2. Create an html file called `gmaps.html`:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org    R/xhtml1/DTD/xhtm
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8"/>
    <title>Google Maps JavaScript API Example<    itle>
    <script src="http://maps.google.com/maps?file=api&amp;v=2.x&amp;key=<INSERT MAPS API KEY HER

    <script type="text/javascript">
    //<![CDATA[
        function load() {
            if (GBrowserIsCompatible()) {
                var map = new GMap2(document.getElementById("map"));
                map.addControl(new GLargeMapControl());
                map.setCenter(new GLatLng(40,-98), 4);
                var geoXml = new GGeoXml("<INSERT GEOSERVER URL HERE>/geoserver/wms/reflect?laye
                map.addOverlay(geoXml);
            }
        }
    //]]>
    </script>

    </head>
    <body onload="load()" onunload="GUnload()">
        <div id="map" style="width: 800px; height: 600px"></div>
    </body>
</html>
```

3. Visit `gmaps.html` in your web browser.

Note: The version of the google maps api must be **2.x**, and not just **2** You must insert your specific maps api key, and geoserver base url

21.2.4 Yahoo Maps

How to create a Yahoo! Maps mashup with a GeoRSS overlay produced by GeoServer.

1. Obtain a <Yahoo Maps Application ID <http://search.yahooapis.com/webservices/register_application>'_ from Yahoo.
2. Create an html file called `ymaps.html`:

```
<html>
    <head>
    <title>Yahoo! Maps GeoRSS Overlay Example<    itle>
    <script src="http://api.maps.yahoo.com/ajaxymap?v=3.0&appid=<INSERT APPLICATION ID HERE>" ty
    <script type="text/javascript" language="JavaScript">
```

```
        function StartYMap() {
            var map = new YMap(document.getElementById('ymap'));
            map.addPanControl();
            map.addZoomShort();

            function doStart(eventObj) {
                var defaultEventObject = eventObj;
                //eventObj.ThisMap [map object]
                //eventObj.URL [argument]
                //eventObj.Data [processed input]
            }

            function doEnd(eventObj) {
                var defaultEventObject = eventObj;
                //eventObj.ThisMap [map object]
                //eventObj.URL [argument]
                //eventObj.Data [processed input]
                map.smoothMoveByXY(new YCoordPoint(10,50));
            }

            YEvent.Capture(map,EventsList.onStartGeoRSS, function(eventObj) { doStart(eventObj);
            YEvent.Capture(map,EventsList.onEndGeoRSS, function(eventObj) { doEnd(eventObj); });

            map.addOverlay(new YGeoRSS('http://<INSERT GEOSERVER URL HERE>/geoserver/wms/reflect
        }

    window.onload = StartYMap;
     </script>
      </head>
      <body>
           <div id="ymap" style="width: 800px; height: 600px; left:2px; top:2px"></div>
     </body>
</html>
```

3. Visit `ymaps.html` in your web browser.

Note: The version of the yahoo maps api must be **3.0** You must insert your specific application id, and geoserver base url

21.2.5 Microsoft Virtual Earth

Note: Non Internet Explorer Users*: GeoRSS overlays are only supported in Internet Explorer, versions greater then 5.5.

How to create a Microsoft Virtual Earth mashup with a GeoRSS overlay produced by GeoServer.

Note: To access a GeoRSS feed from Microsoft Virtual Earth the file (ve.html) must be accessed from a Web Server, IE. It will not work if run from local disk.

1. Create an html file called `ve.html`. **Note**: You must insert your specific maps api key, and geoserver base url:

```
<html>
  <head>
    <script src="http://dev.virtualearth.net/mapcontrol/v4/mapcontrol.js"></script>
```

```
    <script>
     var map;

     function OnPageLoad()
     {
        map = new VEMap('map');
        map.LoadMap();

        var veLayerSpec = new VELayerSpecification();
        veLayerSpec.Type = VELayerType.GeoRSS;
        veLayerSpec.ID = 'Hazards';
    veLayerSpec.LayerSource = 'http://<INSERT GEOSERVER URL HERE>/geoserver/wms/reflect?layers=s
    veLayerSpec.Method = 'get';
    map.AddLayer(veLayerSpec);
    }
   </script>
 </head>
 <body onload="OnPageLoad();">
    <div id="map" style="position:relative;width:800px;height:600px;"></div>
  </body>

</html>
```

2. Visit `ve.html` in your web browser. You should see the following:

21.3 GetFeatureInfo Templates

This tutorial describes how to use the GeoServer template system to create custom HTML GetFeatureInfo responses.

21.3.1 Introduction

GetFeatureInfo is a WMS standard call that allows one to retrieve information about features and coverages displayed in a map. The map can be composed of various layers, and GetFeatureInfo can be instructed to return multiple feature descriptions, which may be of different types. GetFeatureInfo can generate output in various formats: GML2, plain text and HTML. Templating is concerned with the HTML one.

The default HTML output is a sequence of titled tables, each one for a different layer. The following example shows the default output for the tiger-ny basemap (included in the above cited releases, and onwards).

21.3.2 Standard Templates

The following assumes you're already up to speed with Freemarker templates. If you're not, read the *Freemarker Templates* tutorial, and the *KML Placemark Templates* page, which has simple examples.

The default output is generated by the standard templates, which are three:

- `header.ftl`
- `content.ftl`
- `footer.ftl`

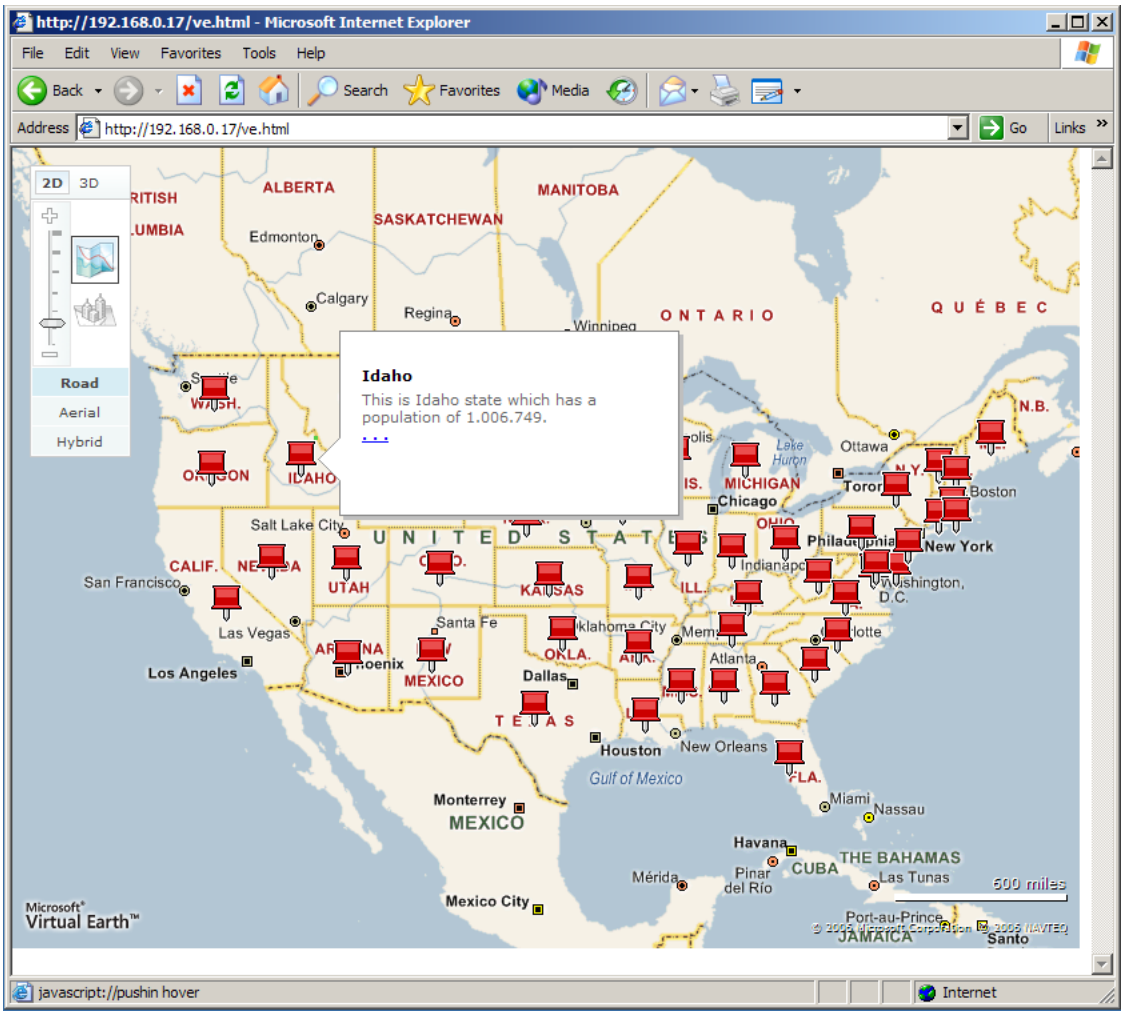
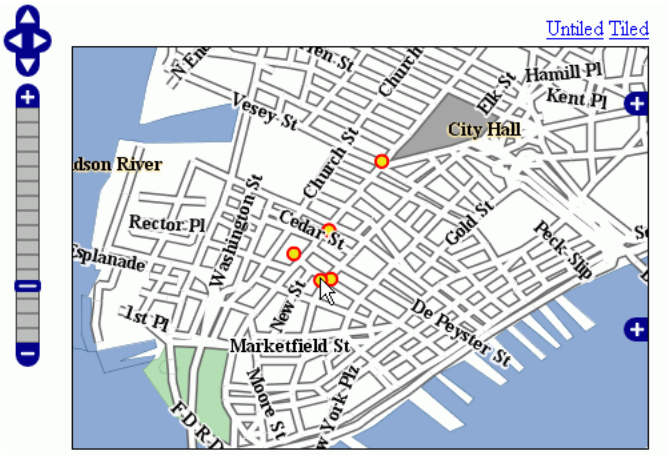


Figure 21.2: *Virtual Earth*



TIGER_ROADS

CFCC	NAME
A41	Wall St

POI

NAME	THUMBNAIL	MAINPAGE
stock	pics/22037829-T.jpg	pics/22037829-L.jpg

Figure 21.3: *Default Output*

The *header template* is invoked just once, and usually contains the start of the HTML page, along with some CSS. The default header template looks like this (as you can see, it's completely static, and it's in fact not provided with any variable you could expand):

```
<#--
Header section of the GetFeatureInfo HTML output. Should have the <head> section, and
a starter of the <body>. It is advised that eventual css uses a special class for featureInfo,
since the generated HTML may blend with another page changing its aspect when usign generic classes
like td, tr, and so on.
-->
<html>
  <head>
    <title>Geoserver GetFeatureInfo output</title>
  </head>
  <style type="text/css">
        table.featureInfo, table.featureInfo td, table.featureInfo th {
                border:1px solid #ddd;
                border-collapse:collapse;
                margin:0;
                padding:0;
                font-size: 90%;
                padding:.2em .1em;
        }
        table.featureInfo th{
            padding:.2em .2em;
                text-transform:uppercase;
                font-weight:bold;
                background:#eee;
        }
        table.featureInfo td{
                background:#fff;
```

```
        }
        table.featureInfo tr.odd td{
                background:#eee;
        }
        table.featureInfo caption{
                text-align:left;
                font-size:100%;
                font-weight:bold;
                text-transform:uppercase;
                padding:.2em .2em;
        }
  </style>
  <body>
```

The *footer template* is similar, a static template used to close the HTML document properly:

```
<#--
Footer section of the GetFeatureInfo HTML output. Should close the body and the html tag.
-->
  </body>
</html>
```

The *content template* is the one that turns feature objects into actual HTML tables. The template is called multiple times: each time it's fed with a different feature collection, whose features all have the same type. In the above example, the template has been called once for the roads, and once for the points of interest (POI). Here is the template source:

```
<#--
Body section of the GetFeatureInfo template, it's provided with one feature collection, and
will be called multiple times if there are various feature collections
-->
<table class="featureInfo">
  <caption class="featureInfo">${type.name}</caption>
  <tr>
<#list type.attributes as attribute>
  <#if !attribute.isGeometry>
    <th >${attribute.name}</th>
  </#if>
</#list>
  </tr>

<#assign odd = false>
<#list features as feature>
  <#if odd>
    <tr class="odd">
  <#else>
    <tr>
  </#if>
  <#assign odd = !odd>

  <#list feature.attributes as attribute>
    <#if !attribute.isGeometry>
      <td>${attribute.value}</td>
    </#if>
  </#list>
  </tr>
</#list>
</table>
<br/>
```

As you can see there is a first loop scanning type and outputting its attributes into the table header, then a second loop going over each feature in the collection (features). From each feature, the attribute collections are accessed to dump the attribute value. In both cases, geometries are skipped, since there is not much point in including them in the tabular report. In the table building code you can also see how odd rows are given the "odd" class, so that their background colors improve readability.

21.3.3 Custom Templates

So, what do you have to do if you want to override the custom templates? Well, it depends on which template you want to override.

`header.ftl` and `footer.ftl` are type independent, so if you want to override them you have to place a file named `header.ftl` or `footer.ftl` in the `templates` directory, located in your GeoServer *GeoServer Data Directory*. On the contrary, `content.ftl` may be generic, or specific to a feature type.

For example, let's say you would prefer a bulleted list appearance for your feature info output, and you want this to be applied to all GetFeatureInfo HTML output. In that case you would drop the following `content.ftl` in the templates directory:

```
<ul>
<#list features as feature>
  <li><b>Type: ${type.name}</b> (id: <em>${feature.fid}</em>):
  <ul>
  <#list feature.attributes as attribute>
    <#if !attribute.isGeometry>
      <li>${attribute.name}: ${attribute.value}</li>
    </#if>
  </#list>
  </ul>
  </li>
</#list>
</ul>
```

With this template in place, the output would be:

Looking at the output we notice that point of interest features refer to image files, which we know are stored inside the default GeoServer distribution in the `demo_app/pics` path. So, we could provide a POI specific override that actually loads the images.

This is easy: just put the following template in the feature type folder, which in this case is `workspaces/topp/DS_poi/poi` (you should refer to your Internet visible server address instead of localhost, or its IP if you have fixed IPs):

```
<ul>
<#list features as feature>
  <li><b>Point of interest, "${feature.NAME.value}"</b>: <br/>
  <img src="http://localhost:8080/geoserver/popup_map/${feature.THUMBNAIL.value}"/>
  </li>
</#list>
</ul>
```

With this additional template, the output is:

As you can see, roads are still using the generic template, whilst POI is using its own custom template.

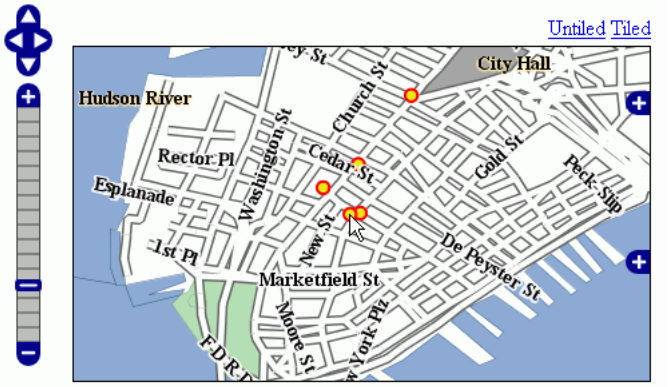


Figure 21.4: *Bulleted List Output*

Figure 21.5: *Output with Thumbnail Image*

21.3.4 Advanced Formating

The `value` property of Feature attribute values are given by geoserver in `String` form, using a sensible default depending on the actual type of the attribute value. If you need to access the raw attribute value in order to apply a custom format (for example, to output `"Enabled"` or `"Disabled"` for a given boolean property, instead of the default `true/false`, you can just use the `rawValue` property instead of `value`. For example: `${attribute.rawValue?string("Enabled", "Disabled")}` instead of just `${attribute.value}`.

21.4 Paletted Images

Geoserver has the ability to output high quality 256 color images. This tutorial introduces you to the palette concepts, the various image generation options, and offers a quality/resource comparison of them in different situations.

21.4.1 What are Paletted Images?

Some image formats, such as GIF or PNG, can use a palette, which is a table of (usually) 256 colors to allow for better compression. Basically, instead of representing each pixel with its full color triplet, which takes 24bits (plus eventual 8 more for transparency), they use a 8 bit index that represent the position inside the palette, and thus the color.

This allows for images that are 3-4 times smaller than the standard images, with the limitation that only 256 different colors can appear on the image itself. Depending of the actual map, this may be a very stringent limitation, visibly degrading the image quality, or it may be that the output cannot be told from a full color image. But for many maps one can easily find 256 representative colors.

In the latter case, the smaller footprint of paletted images is usually a big gain in both performance and costs, because more data can be served with the same internet connection, and the clients will obtain responses faster.

21.4.2 Formats and Antialiasing

Internet standards offer a variety of image formats, all having different strong and weak points. The three most common formats are:

- **JPEG**: a lossy format with tunable compression. JPEG is best suited for imagery layers, where the pixel color varies continuously from one pixel to the next one, and allows for the best compressed outputs. On the contrary, it's not suited to most vector layers, because even slight compression generates visible artifacts on uniform color areas.
- **PNG**: a non lossy format allowing for both full color and paletted. In full color images each pixel is encoded as a 24bits integer with full transparency information (so PNG images can be translucent), in paletted mode each pixel is an 8 bit index into a 256 color table (the palette). This format is best suited to vector layers, especially in the paletted version. The full color version is sometimes referred as PNG24, the paletted version as PNG8.
- **GIF**: a non lossy format with a 256 color palette, best suited for vector layers. Does not support translucency, but allows for fully transparent pixels.

So, as it turns out, paletted images can be used with profit on vector data sets, either using the PNG8 or GIF formats.

Antialiasing plays a role too. Let's take a road layer, where each road is depicted by a solid gray line, 2 pixels thick. One may think this layer needs only 2 colors: the background one (eventually transparent) and gray. In fact, this is true only if no antialiasing is enabled. Antialiasing will smooth the borders of the line giving a softer, better looking shape, and it will do so by adding pixels with an intermediate color, thus increasing the number of colors that are needed to fully display the image.

The following zoom of an image shows antialiasing in action:

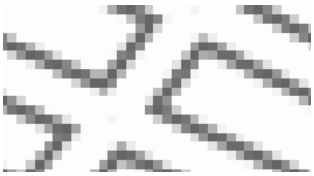


Figure 21.6: *Antialiasing*

These output formats, if no other parameters are provided, do compute the optimal palette on the fly. As you'll see, this is an expensive process (CPU bound), but as you'll see, depending on the speed of the network connecting the server and the client, the extra cost can be ignored (especially if the bottleneck can be found in the network instead of the server CPU).

Optimal palette computation is anyways a repetitive work that can be done up front: a user can compute the optimal palette once, and tell GeoServer to use it. There are three ways to do so:

1. Use the internet safe palette, a standard palette built in into GeoServer, by appending `palette=safe` to the `GetMap` request.
2. Provide a palette by example. In this case, the user will generate an 256 color images using an external program (such as Photoshop), and then will save it into the `$GEOSERVER_DATA_DIR/palettes` directory. The sample file can be either in GIF or PNG format. If the file is named mypalette.gif or mypalette.png, the user will be able to refer it appending `palette=mypalette` to the GetMap request. GeoServer will load the palette from the file and use it.
3. Provide a palette file. The palette file must be in JASC-PAL format, and have a `.pal` extension. This file type can be generated by applications such as Paint Shop Pro and IrfanView, but also can be generated manually in a text editor. The process is just as before, but this time only the palette file will be stored into `$GEOSERVER_DATA_DIR/palettes`.

Note: GeoServer does not support palette files in Microsoft Palette format, despite having the same `.pal` file extension.

21.4.3 An Example with Vector Data

Enough theory, let's have a look at how to deal with paletted images in practice. We'll use the `tiger-ny` basemap to gather some numbers, and in particular the following map request:

And we'll change various parameters in order to play with formats and palettes. Here goes the sampler:

Parameters:`FORMAT=image/png` | Size: 257 KB | Map generation time: 0.3s

Parameters:`FORMAT=image/png8` | Size: 60 KB | Map generation time: 0.6s

Parameters:`FORMAT=image/png` | Size: 257 KB | Map generation time: 0.3s

Parameters:`FORMAT=image/png` & `palette=nyp` | Size: 56KB | Map generation time: 0.3s

Figure 21.7: *The standard PNG full color output*

Figure 21.8: *The PNG8 output*

Figure 21.9: *PNG + internet safe palette*

Figure 21.10: *PNG + 'custom palette <http://geoserver.org/download/attachments/1278244/nyp.pal?version=1>'_*

The attachments include also the GIF outputs, whose size, appearance and generation time does not differ significantly from the PNG outputs.

As we can see, depending on the choice we have a variation on the image quality, size and generation time (which has been recorded using the FasterFox Firefox extension timer, with the browser sitting on the same box as the server). Using `palette=xxx` provides the best match in speed and size, thought using the built in internet safe palette altered the colors. Then again, the real gain can be seen only by assuming a certain connection speed between the server and the client, and adding the time required to move the image to the client. The following table provides some results:

Configuration	GT(s)	File size (kb)	TT 256kbit/s	TT 1MBit/s	TT 4MBit/s	TT 20MBit/s
tiger-ny-png	0,36	257	8,39	2,42	0,87	0,46
tyger-ny-png8	0,6	60	2,48	1,08	0,72	0,62
tiger-ny-png + safe palette	0,3	56	22,05	0,75	0,41	0,32
tiger-ny-png + custom palette	0,3	59	2,14	0,77	0,42	0,32

Legend:

- `GT`: map generation time on the same box
- `TT <speed>`: total time needed for a client to show the image, assuming an internet connection of the given speed. This time is a sum of of the image generation time and the transfer time, that is, `GT + sizeInKbytes * 8/ speedInKbits`.

As the table shows, the full color PNG image takes usually a lot more time than other formats, unless it's being served over a fast network (and even in this case, one should consider network congestion as well). The png8 output format proves to be a good choice if the connection is slow, whilst the extra work done in looking up an optimal palette always pays back in faster map delivery.

21.4.4 Generating the custom palette

The nyp.pal file has been generated using IrfanView, on Windows. The steps are simple:

- open the png 24 bit version of the image
- use Image/Decrease Color Depth and set 256 colors
- use Image/Palette/Export to save the palette

21.4.5 An example with raster data

To give you an example when paletted images may not fit the bill, let's consider the sf:dem coverage from the sample data, and repeat the same operation as before.

Parameters:`FORMAT=image/png` Size: 117 KB | Map generation time: 0.2s

Parameters:`FORMAT=image/jpeg` Size: 23KB | Map generation time: 0.12s

Parameters:`FORMAT=image/png8` Size: 60 KB | Map generation time: 0.5s

Parameters:`FORMAT=image/png` & `palette=dem-png8` Size: 48KB | Map generation time: 0.15s

Parameters:`FORMAT=image/png``& ``palette=safe` Size: 17KB | Map generation time: 0.15s

As the sample shows, the JPEG output has the same quality as the full color image, is generated faster and uses only 1/5 of its size. On the other hand, the version using the internet safe palette is fast and small, but the output is totally ruined. Everything considered, JPEG is the clear winner, sporting good quality, fast image generation and a size that's half of the best png output we can get.

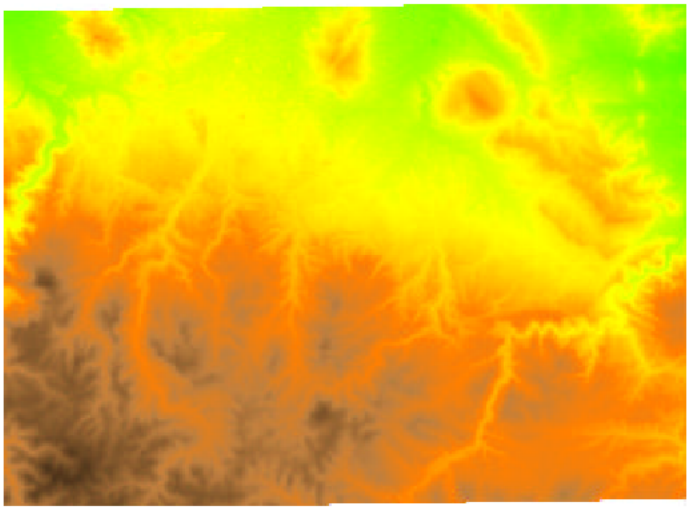


Figure 21.11: *The standard PNG full color output.*

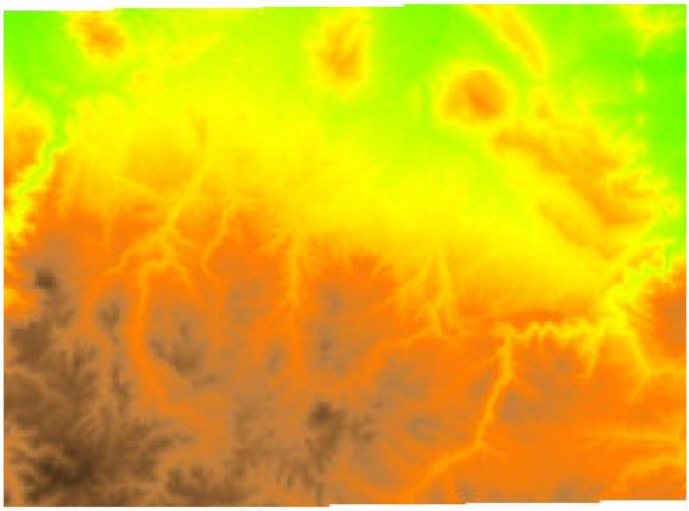


Figure 21.12: *JPEG output*

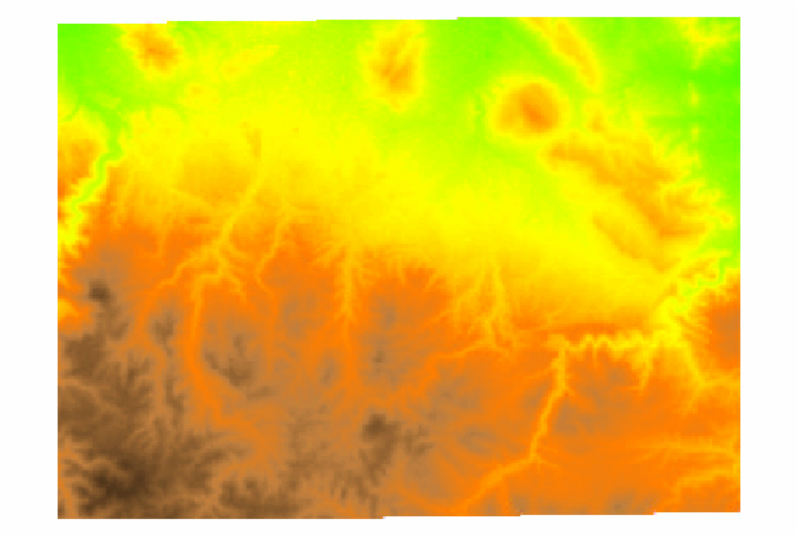


Figure 21.13: *The PNG8 output.*

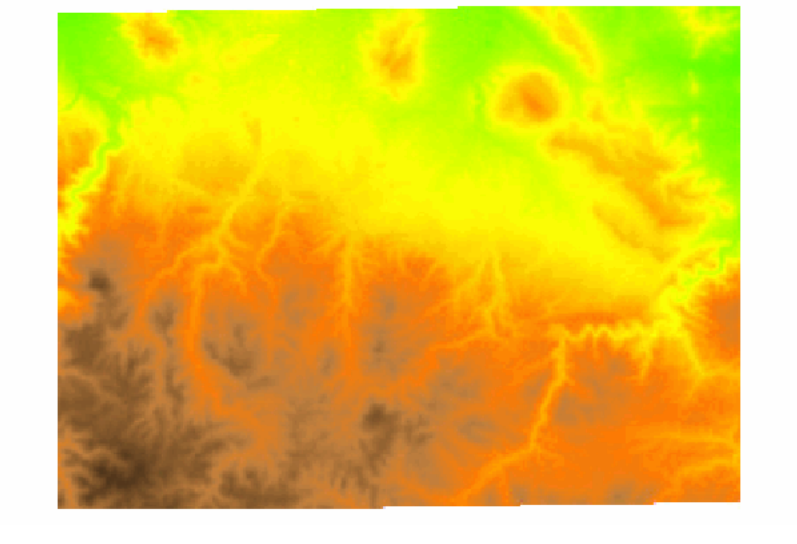


Figure 21.14: *PNG + custom palette (using the png8 output as the palette).*

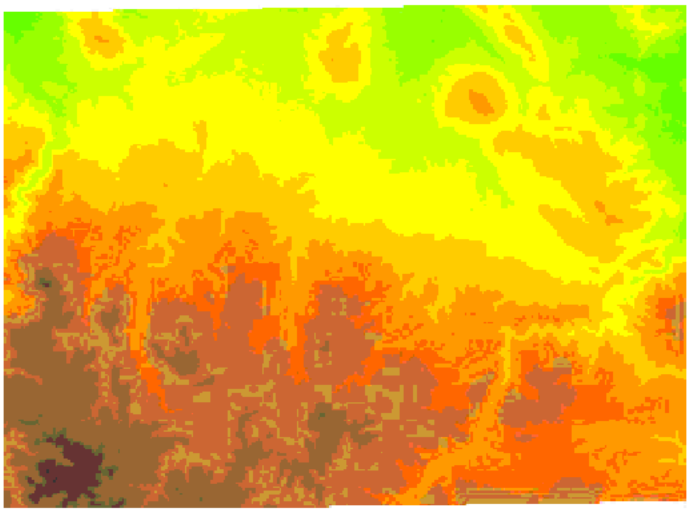


Figure 21.15: *PNG + internet safe palette.*

21.5 Serving Static Files

You can place static files in the `www` subdirectory of the GeoServer *data directory*, and they will be served at `http:/myhost:8080/geoserver/www`. This means you can deploy HTML, images, or JavaScript, and have GeoServer serve them directly on the web.

This approach has some limitations:

- GeoServer can only serve files whose MIME type is recognized. If you get an HTTP 415 error, this is because GeoServer cannot determine a file's MIME type.
- This approach does not make use of accelerators such as the Tomcat APR library. If you have many static files to be served at high speed, you may wish to create your own web app to be deployed along with GeoServer or use a separate web server to serve the content.

21.6 WMS Reflector

21.6.1 Overview

Standard WMS requests can be quite long and verbose. For instance the following, which returns an OpenLayers application with an 800x600 image set to display the feature `topp:states`, with bounds set to the northwestern hemisphere by providing the appropriate bounding box.

```
http://localhost:8080/geoserver/wms?service=WMS&request=GetMap&version=1.1.1&format=application/openl
```

Typing into a browser, or HTML editor, can be quite cumbersome and error prone. The WMS Reflector solves this problem nicely by using good default values for the options that you do not specify. Using the reflector one can shorten the above request to:

```
http://localhost:8080/geoserver/wms/reflect?format=application/openlayers&layers=topp:states&width=80
```

This request only specifies that you want the reflector (wms/reflect) to return an OpenLayers application (format=application/openlayers), that you want it to display the feature "topp:states" (layers=topp:states) and that the width should be 800 pixels (width=800). However, this will not return the exact same value

as above. Instead, the reflector will zoom to the bounds of the feature and return a map that is 800 pixels wide, but with the height adjusted to the aspect ratio of the feature.

21.6.2 Using the WMS Reflector

To use the WMS reflector all one must do is specify `wms/reflect?` as opposed to `wms?` in a request. The only mandatory parameter to a WMS reflector call is the *layers* parameter. As stated above the reflector fills in sensible defaults for the rest of the parameters. The following table lists all the defaults used:

request	getmap
service	wms
version	1.1.1
format	image/png
width	512
height	512 if width is not specified
srs	EPSG:4326
bbox	bounds of layer(s)

Any of these defaults can be overridden when specifying the request. The *styles* parameter is derived by using the default style as configured by GeoServer for each *layer* specified in the layers parameter.

Any parameter you send with a WMS request is also legitimate when requesting data from the reflector. Its strength is what it does with the parameters you do not specify, which is explored in the next section.

layers: This is the only mandatory parameter. It is a comma separated list of the layers you wish to include in your image or OpenLayers application.

format: The default output format is image/png. Alternatives include image/jpeg (good for raster backgrounds), image/png8 (8 bit colors, smaller files) and image/gif

width: Describes the width of the image, alternatively the size of the map in an OpenLayers. It defaults to 512 pixels and can be calculated based on the height and the aspect ratio of the bounding box.

height: Describes the height of the image, alternatively the map in an OpenLayers. It can be calculated based on the width and the aspect ratio of the bounding box.

bbox: The bounding box is automatically determined by taking the union of the bounds of the specified layers. In essence, it determines the extent of the map. By default, if you do not specify bbox, it will show you everything. If you have one layer of Los Angeles, and another of New York, it show you most of the United States. The bounding box, automatically set or specified, also determines the aspect ratio of the map. If you only specify one of width or height, the other will be determined based on the aspect ratio of the bounding box.

> **Warning:** If you specify height, width and bounding box there are zero degrees of freedom, and if the aspect ratios do not match your image will be warped.

styles: You can override the default styles by providing a comma separated list with the names of styles which must be known by the server.

srs: The spatial reference system (SRS) parameter is somewhat difficult. If not specified the WMS Reflector will use EPSG:4326 / WGS84. It will support the native SRS of the layers as well, provided all layers share the same one.

Example 1

Request the layer topp:states , it will come back with the default style (demographic), width (512 pixels) and height (adjusted to aspect ratio).

```
http://localhost:8080/geoserver/wms/reflect?layers=topp:states
```

Example 2

Request the layers topp:states and sf:restricted, it will come back with the default styles, and the specified width (640 pixels) and the height automatically adjusted to the aspect ratio.

```
http://localhost:8080/geoserver/wms/reflect?layers=topp:states,sf:restricted&width=640
```

Example 3

In the example above the sf:restricted layer is very difficult to see, because it is so small compared to the United States. To give the user a chance to get a better view, if they choose, we can return an OpenLayers application instead. Zoom in on South Dakota (SD) to see the restricted areas.

```
http://localhost:8080/geoserver/wms/reflect?format=application/openlayers&layers=topp:states,sf:restr
```

Example 4

Now, if you mainly want to show the restricted layer, but also provide the context, you can set the bounding box for the the request. The easiest way to obtain the coordinates is to use the application in example three and the coordinates at the bottom right of the map. The coordinates displayed in OpenLayers are x , y , the reflector service expects to be given bbox=minx,miny,maxx,maxy . Make sure it contains no whitespaces and users a period (".") as the decimal separator. In our case, it will be bbox=-103.929,44.375,-103.633,44.500

```
http://localhost:8080/geoserver/wms/reflect?format=application/openlayers&layers=topp:states,sf:restr
```

21.6.3 Outputting to a Webpage

Say you have a webpage and you wish to include a picture that is 400 pixels wide and that shows the layer `topp:states`, on this page.

```
<img src="http://localhost:8080/geoserver/wms/reflect?layers=topp:states&width=400" />
```

If you want the page to render in the browser before Geoserver is done, you should specify the height and width of the picture. You could just pick any approximate value, but it may be a good idea to look at the generated image first and then use those values. In the case of the layer above, the height becomes 169 pixels, so we can specify that as an attribute in the <img> tag:

```
<img src="http://localhost:8080/geoserver/wms/reflect?layers=topp:states&width=400" height="169" widt
```

If you are worried that the bounds of the layer may change, so that the height changes relative to the width, you may also want to specify the height in the URL to the reflector. This ensures the layer will always be centered and fit on the 400x169 canvas.

The reflector can also create a simple instance of OpenLayers that shows the layers you specify in your request. One possible application is to turn the image above into a link that refers to the OpenLayers instance for the same feature, which is especially handy if you think a minority of your users will want to

take closer look. To link to this JavaScript application, you need to specify the output format of the reflector: `format=application/OpenLayers`

```
http://localhost:8080/geoserver/wms/reflect?format=application/openlayers&width=400
```

The image above then becomes

```
<a href="http://localhost:8080/geoserver/wms/reflect?format=application/openlayers&layers=topp:states
<img src="http://localhost:8080/geoserver/wms/reflect?layers=topp:states&width=400" height="169" widt
</a>
```

(The a-tags are on separate lines for clarity, they will in fact result in a space in front and after the image).

21.6.4 OpenLayers in an iframe

Many people do not like iframes, and for good reasons, but they may be appropriate in this case. The following example will run OpenLayers in an iframe.

```
<iframe src ="http://localhost:8080/geoserver/wms/reflect?format=application/openlayers&layers=topp:s
</iframe>
```

Alternatively, you can open OpenLayers in a separate webpage and choose "View Source code" in your browser. By copying the HTML you can insert the OpenLayers client in your own page without using an iframe.

21.7 WMS Animator

21.7.1 Overview

Standard WMS can generate static maps only. There is a number of use cases in which generating an animation is of interest. An obvious case is time-based animation. Other uses include elevation-based animation, varying the values of SQL View or SLD substitution parameters, or the changing the extent of the generated map to produce the appearance of a moving viewport.

This capability is provided by the **WMS Animator**. The WMS Animator works in a similar way to the WMS Reflector. It uses a provided partial WMS request as a template, and the **animator parameters** are used to generate and execute a sequence of complete requests. The rendered map images are combined into a single output image (in a format that supports multi-frame images).

The Animator is invoked by using the `wms/animate` request path. Any WMS paramaters can be animated, including nested ones such as *SLD environment variables*. To define the appearance of the animation additional parameters are provided:

- **aparam** specifies the name of the parameter that will be changed in the request for each frame. This can be any WMS parameter such as `layers`, `cql_filter`, `bbox`, `style` and so on. Nested parameters (such as required by the `format_options`, `env` and `view_params` parameters), are supported using the syntax of `param:name` (for example, `view_params:year`).
- **avalues** is a comma-separated list of the values the animation parameter has for each frame. If a value contain commas these must be escaped using a backslash. (For instance, this occurs when providing BBOX values.)

The Animator parses the input values and uses string replacement to generate the sequence of WMS requests to be executed. Each generated request is executed to produce one frame. It is up to the caller to ensure the provided animation parameters result in valid WMS requests.

For example, to generate an animation of a layer with the viewport scrolling towards the east, the WMS BBOX parameter is given the series of values `-90,40,-60,70`, `-80,40,-60,70` and `-70,40,-50,70` (note the escaping of the commas in the BBOX values):

```
http://localhost:8080/geoserver/wms/animate
?layers=topp:states
&aparam=bbox
&avalues=-90\,40\,-60\,70,-80\,40\,-60\,70,-70\,40\,-50\,70
```

For an example of nested parameters, assume the existence of a style named `selection` using an SLD variable `color`. The following request creates an animated map where the selection color changes between red, green and blue

```
http://localhost:8080/geoserver/wms/animate
?layers=topp:states,topp:states
&styles=polygon,selection
&aparam=env:color
&avalues=FF0000,00FF00,0000FF
```

21.7.2 Using the WMS Animator

To invoke the WMS Animator specify the path `wms/animate` instead of `wms` in a GetMap request.

Every Animator request must specify the `layers`, `aparam` and `avalues` parameters. Any other valid WMS parameters may be used in the request as well. If any necessary parameters are omitted, the Animator provides sensible default values for them. The following defaults are used:

Parameter	Default Value
`request`	`getmap`
`service`	`wms`
`version`	`1.1.1`
`format`	`image/png`
`width`	`512`
`height`	`512` if width is not specified
`srs`	`EPSG:4326`, or SRS common to all layers
`bbox`	bounds of specified layer(s)
`styles`	default styles configured for specified layer(s)

Further details of these parameters are:

layers: This is the only mandatory standard parameter. It is a comma-separated list of the layers to be included in the output map.

format: The default output format is `image/png`. Supported values are `image/jpeg` (suitable for raster backgrounds), `image/png8` (8-bit colors, smaller files) and `image/gif`

> **Warning:** In order to produce an actual animated image the format must support animation. At this time the only one provide in GeoServer is **image/gif;subtype=animated**

width: Describes the width of the image. It defaults to 512 pixels, and can be calculated based on the specified height and the aspect ratio of the bounding box.

height: Describes the height of the image. It can be calculated based on the specified width and the aspect ratio of the bounding box.

bbox: Specifies the extent of the map frame. The default bounding box is determined by taking the union of the bounds of the specified layers. (For example, if one layer shows Los Angeles and another shows New

York, the default map shows most of the United States. The bounding box also determines the aspect ratio of the map. If only one of `width` or `height` is specified, the other is determined based on the aspect ratio of the bounding box.

styles: The default value is the default styles configured in GeoServer for the layers specified in the `layers` parameter. This can be overridden by providing a comma-separated list of style names (which must be known to the server).

srs: If all layers share the same SRS, this is used as the default value. Otherwise, the default value is EPSG:4326 (WGS84).

Animation Options

The Animator provides options to control looping and frame speed. These are specified using the `format_options` *WMS parameter*. The available options are:

Option	Description
`gif_loop_continuosly`	If `true` the animation will loop continuously. The default is `false`.
`gif_frames_delay`	Specifies the frame delay in milliseconds. The default is 1000 ms.

Example 1

Requests the layer `topp:states`, using the default style (`demographic`), width (512 pixels) and height (adjusted to aspect ratio). The `aparam=bbox` parameter specifies that the output animation has two frames, one using a whole-world extent and the other with the extent of the USA. This gives the effect of zooming in.

```
http://localhost:8080/geoserver/wms/animate
?layers=topp:states
&format=image/gif;subtype=animated
&aparam=bbox
&avalues=-180\,-90\,180\,90,-125\,25\,-67\,50
```

Example 2

Requests the layers `topp:states` and `sf:restricted`, using `format_options=gif_loop_continuosly:true` to request an infinite loop animation. The output map uses the default styles, the specified width (640 pixels), and the height automatically adjusted to the aspect ratio.

```
http://localhost:8080/geoserver/wms/animate
?layers=topp:states,sf:restricted
&format=image/gif;subtype=animated
&aparam=bbox
&avalues=-180\,-90\,180\,90,-125\,25\,-67\,50
&format_options=gif_loop_continuosly:true
&width=640
```

Example 3

The following request uses the `format_options` of `gif_loop_continuosly:true` and `gif_frames_delay:10` to rotate the map image fast and continously.

```
http://localhost:8080/geoserver/wms/animate
?layers=topp:states,sf:restricted
&format=image/gif;subtype=animated
&aparam=angle
&avalues=0,45,90,135,180,225,270,365
&format_options=gif_loop_continuosly:true;gif_frames_delay:10
&width=640
```

21.7.3 Displaying frame parameters as decorations

It is possible to decorate each frame image with the `avalue` parameter value that generated it using the *WMS Decorations* `text` decoration. The current animation parameter value can be accessed via the `avalue` environment variable. (This environment variable can also be used in *Variable substitution in SLD*.)

Here is an example that uses a decoration showing the frame parameter value:

```
http://localhost:8080/geoserver/wms/animate
?layers=topp%3Aworld
&aparam=time
&avalues=2004-01-01T00:00:00.000Z,2004-02-01T00:00:00.000Z
&format=image/gif;subtype=animated
&format_options=layout:message
```

It uses the following decoration layout, located in `layouts/message.xml`:

```
<layout>
  <decoration type="text" affinity="bottom,right" offset="6,6">
    <option name="message" value="${avalue}"/>
    <option name="font-size" value="12"/>
    <option name="font-family" value="Arial"/>
    <option name="halo-radius" value="2"/>
  </decoration>
</layout>
```

21.7.4 Specifying WMS Animator default behaviour

The GeoServer Adinistrator GUI allows specifying some limits and default options for the WMS Animator. The settings are made on the *Services* > *WMS* config screen as shown below:

The first three options set server limits on the animation output. It is possible to set the **maximum number of frames** an animation can contain, the **maximum rendering time** to produce an animation and the **maximum size** of the whole animation.

The default animation **frame delay** (expressed in ms) and **looping behaviour** can be set as well. These values can be overridden by using the `format_options` parameter as described above.

21.8 CQL and ECQL

CQL (Common Query Language) is a query language created by the OGC for the Catalogue Web Services specification. Unlike the XML-based Filter Encoding language, CQL is written using a familiar text-based syntax. It is thus more readable and better-suited for manual authoring.

However, CQL has some limitations. For example it cannot encode id filters, and it requires an attribute to be on the left side of any comparison operator. For this reason, GeoServer provides an extended version of

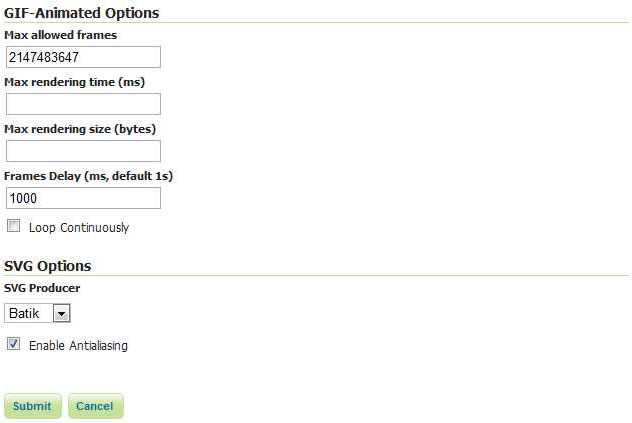


Figure 21.16: *WMS Animator default settings*

CQL called ECQL. ECQL removes the limitations of CQL, providing a more flexible language with stronger similarities with SQL.

GeoServer supports the use of both CQL and ECQL in WMS and WFS requests, as well as in GeoServer's SLD *dynamic symbolizers*. Whenever the documentation refers to CQL, ECQL syntax can be used as well (and if not, please report that as a bug!).

This tutorial introduces the CQL/ECQL language by example. For a full reference, refer to the *ECQL Reference*.

21.8.1 Getting started

The following examples use the `topp:states` sample layer shipped with GeoServer. They demonstrate how CQL filters work by using the WMS *CQL_FILTER vendor parameter* to alter the data displayed by WMS requests. The easiest way to follow the tutorial is to open the GeoServer Map Preview for the `topp:states` layer. Click on the *Options* button at the top of the map preview to open the advanced options toolbar. The example filters can be entered in the *Filter: CQL* box.

The attributes used in the filter examples are those included in the layer. For example, the following are the attribute names and values for the Colorado feature:

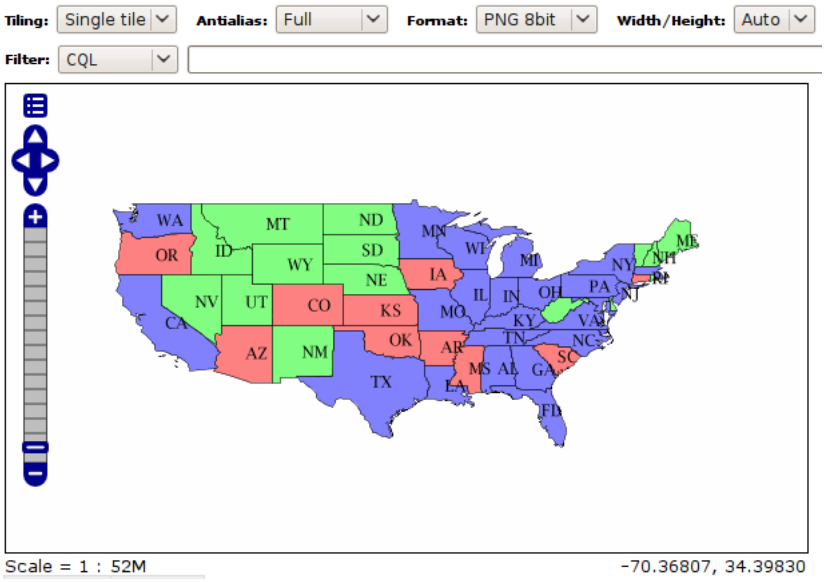


Figure 21.17: *topp:states preview with advanced toolbar open.*

Attribute	states.6
STATE_NAME	Colorado
STATE_FIPS	08
SUB_REGION	Mtn
STATE_ABBR	CO
LAND_KM	268659.501
WATER_KM	960.364
PERSONS	3294394.0
FAMILIES	854214.0
HOUSHOLD	1282489.0
MALE	1631295.0
FEMALE	1663099.0
WORKERS	1233023.0
DRVALONE	1216639.0
CARPOOL	210274.0
PUBTRANS	46983.0
EMPLOYED	1633281.0
UNEMPLOY	99438.0
SERVICE	421079.0
MANUAL	181760.0
P_MALE	0.495
P_FEMALE	0.505
SAMP_POP	512677.0

21.8.2 Simple comparisons

Let's get started with a simple example. In CQL arithmetic and comparisons are expressed using plain text. The filter `PERSONS > 15000000` will select states that have more than 15 million inhabitants:

The full list of comparison operators is: =, <>, >, >=, <, <=.

To select a range of values the BETWEEN operator can be used: `PERSONS BETWEEN 1000000 AND`

Figure 21.18: *PERSONS > 15000000*

`3000000`:

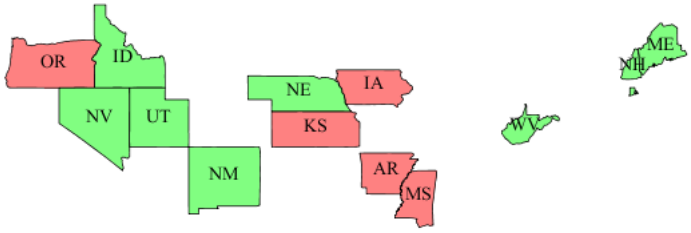


Figure 21.19: *PERSONS BETWEEN 1000000 AND 3000000*

Comparison operators also support text values. For instance, to select only the state of California, the filter is `STATE_NAME = 'California'`. More general text comparisons can be made using the `LIKE` operator. `STATE_NAME LIKE 'N%'` will extract all states starting with an "N":

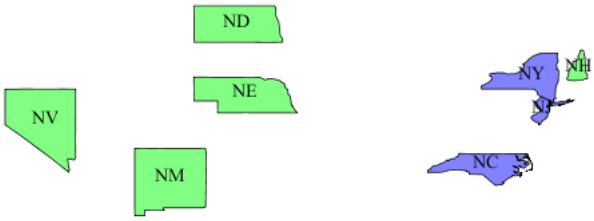


Figure 21.20: *STATE_NAME LIKE 'N%'*

It is also possible to compare two attributes with each other. `MALE > FEMALE` selects the states in which the male population surpasses the female one (a rare occurrence):

Arithmetic expressions can be computed using the `+`, `-`, `*`, `/` operators. The filter `UNEMPLOY / (EMPLOYED + UNEMPLOY) > 0.07` selects all states whose unemployment ratio is above 7% (remember the sample data is very old, so don't draw any conclusion from the results!)

Figure 21.21: *MALE > FEMALE*

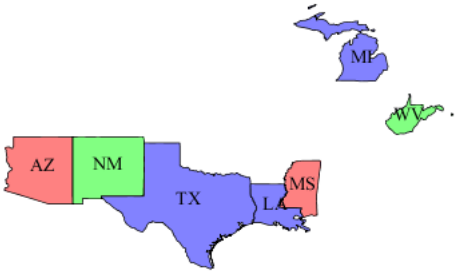


Figure 21.22: *UNEMPLOY / (EMPLOYED + UNEMPLOY) > 0.07*

21.8.3 Id and list comparisons

If we want to extract only the states with specific feature ids we can use the `IN` operator without specifying any attribute, as in `IN ('states.1', 'states.12')`:

Figure 21.23: *IN ('states.1', 'states.12')*

If instead we want to extract the states whose name is in a given list we can use the `IN` operator specifying an attribute name, as in `STATE_NAME IN ('New York', 'California', 'Montana', 'Texas')`:

21.8.4 Filter functions

CQL/ECQL can use any of the *filter functions* available in GeoServer. This greatly increases the power of CQL expressions.

For example, suppose we want to find all states whose name contains an "m", regardless of letter case. We can use the `strToLowerCase` to turn all the state names to lowercase and then use a like comparison:

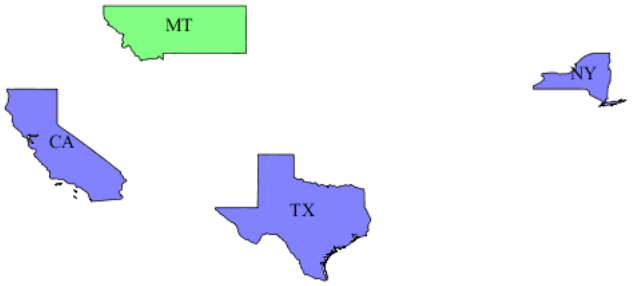


Figure 21.24: *STATE_NAME IN ('New York', 'California', 'Montana', 'Texas')*

`strToLowerCase(STATE_NAME) like '%m%'`:

Figure 21.25: *strToLowerCase(STATE_NAME) like '%m%'*

21.8.5 Geometric filters

CQL provides a full set of geometric filter capabilities. Say, for example, you want to display only the states that intersect the (-90,40,-60,45) bounding box. The filter will be `BBOX(the_geom, -90, 40, -60, 45)`

Figure 21.26: *BBOX(the_geom, -90, 40, -60, 45)*

Conversely, you can select the states that do *not* intersect the bounding box with the filter: `DISJOINT(the_geom, POLYGON((-90 40, -90 45, -60 45, -60 40, -90 40)))`:

The full list of geometric predicates is: `EQUALS`, `DISJOINT`, `INTERSECTS`, `TOUCHES`, `CROSSES`, `WITHIN`, `CONTAINS`, `OVERLAPS`, `RELATE`, `DWITHIN`, `BEYOND`.

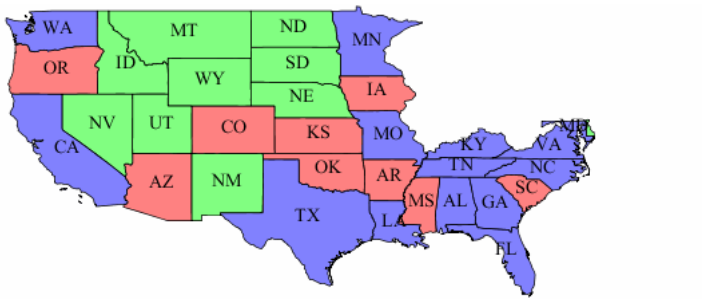


Figure 21.27: *DISJOINT(the_geom, POLYGON((-90 40, -90 45, -60 45, -60 40, -90 40)))*

21.9 Using the ImageMosaic plugin

21.9.1 Introduction

This tutorial describes the process of creating a new coverage using the new ImageMosaic plugin. The ImageMosaic plugin is provided by GeoTools, and allows the creation of a mosaic from a number of georeferenced rasters. The plugin can be used with Geotiffs, as well as rasters accompanied by a world file (.pgw for png files, .jgw for jpg files, etc.). In addition, if imageio-ext GDAL extensions are properly installed we can also serve all the formats supported by it like MrSID, ECW, JPEG2000, etc... See *GDAL Image Formats* for more information on how to install them.

The JAI documentation gives a good description about what a Mosaic does:

The "Mosaic" operation creates a mosaic of two or more source images. This operation could be used for example to assemble a set of overlapping geospatially rectified images into a contiguous image. It could also be used to create a montage of photographs such as a panorama.

Briefly the ImageMosaic plugin is responsible for composing together a set of similar raster data, which, from now on I will call *granules*. The plugin has, of course, some limitations:

1. All the granules must share the same Coordinate Reference System, no reprojection is performed. This will always be a constraint.
2. All the granules must share the same ColorModel and SampleModel. This is a limitation/assumption of the underlying JAI Mosaic operator: it basically means that the granules must share the same pixel layout and photometric interpretation. It would be quite difficult to overcome this limitation, but to some extent it could be done. Notice that, in case of colormapped granules, if the various granules share the same colormap the code will do its best to retain it and try not to expand them in memory. This can also be controlled via a parameter in the configuration file (se next sections)
3. All the granules must share the same spatial resolution and set of overviews (if this is not true, overviews will not be used).

21.9.2 Granule Index

In order to configure a new CoverageStore and a new Coverage with this plugin, an index file needs to be generated first in order to associate each granule to its bounding box. Currently we support only a Shapefile as a proper index, although it would be possible to extend this and use other means to persist the index.

More specifically, the following files make up the mosaic configuration:

1. A shapefile that contains enclosing polygons for each raster file. This shapefile needs to have a field whose values are the paths for the mosaic granules. The path can be either relative to the shape-

file itself or absolute, moreover, while the default name for the shapefile attribute that contains the granules' paths is "location", such a name can be configured to be different (we'll describe this later on).

2. A projection file (.prj) for the above-mentioned shapefile.
3. A configuration file (.properties). This file contains properties such as cell size in x and y direction, the number of rasters for the ImageMosaic coverage, etc.. We will describe this file in the next section.

Normally GeoServer will create automatically those files when pointed a directory containing images, but it's importat to understand them anyways in order to get better control on how the mosaic works, and troubleshoot eventual issues.

21.9.3 Configuration File

The mosaic configuration file is used to store some configuration parameters to control the ImageMosaic plugin. It is created as part of the mosac creation and usually do not require manual editing. The table below describes the various elements in this configuration file.

Parameter	**Mandatory**	**Description**
Envelope2D	Y	Contains the envelope for this mosaic formatted as LLCx,LLXy URCx,URCy (notice the space between the coordinates of the Lower Left Corner and the coordinates of the Upper Right Corner). An example is *Envelope2D=432500.25,81999.75 439250.25,84999.75*
LevelsNum	Y	Represents the number of reduced resolution layers that we currently have for the granules of this mosaic.
Levels	Y	Represents the resolutions for the various levels of the granules of this mosaic. Please remember that we are currently assuming that the number of levels and the resolutions for such levels are the same across alll the granules.
Name	Y	Represents the name for this mosaic.
ExpandToRGB	N	Applies to colormapped granules. Asks the internal mosaic engine to expand the colormapped granules to RGB prior to mosaicing them. This is needed whenever the the granulesdo not share the same color map hence a straight composition that would retain such a color map cannot be performed.
AbsolutePath	Y	It controls whether or not the path stored inside the "location" attribute represents an absolute path or a path relative to the location of the shapefile index. Notice that a relative index ensure much more portability of the mosaic itself. Default value for this parameter is False, which means relative paths.
LocationAttribute	N	The name of the attribute path in the shapefile index. Default value is *location*.

21.9.4 Creating Granules Index and Configuration File

The refactored version of the ImageMosaic plugin can be used to create the shapefile index as well as the mosaic configuration file on the fly without having to rely on gdal or some other similar utility.

If you have a tree of directories containing the granules you want to be able to serve as a mosaic (and providing that you are respecting the conditions written above) all you need to do is to point the GeoServer to such a directory and it will create the proper ancillary files by inspecting all the files present in the the tree of directories starting from the provided input one.

21.9.5 Configuring a Coverage in Geoserver

This is a process very similar to creating a FeatureType. More specifically, one has to perform the steps higlighted in the sections here below.

Create a new CoverageStore:

1. Go to "Data Panel | Stores" via the web interface and click 'Add new Store'. Finally click "ImageMosaic - Image mosaicking plugin" from "Raster Data Source":

Figure 21.28: *ImageMosaic in the list of raster data stores*

2. In order to create a new mosaic is necessary:

- To chose the Workspace in the 'Basic Store Info' section.
- To give a name in the 'Basic Store Info' section.
- To fill the field URL in the 'Connection Parameters' section. You have three alternatives:
  - Inserting the absolute path of the shapefile.
  - Inserting the absolute path of the directory in which the mosaic shapefile index resides, the GeoServer will look for it and make use of it.
  - Inserting the absolute path of a directory where the files you want to mosaic together reside. In this case GeoServer automatically creates the needed mosaic files (.dbf, .prj, .properties, .shp and .shx) by inspecting the data of present in the given directory (GeoServer will also find the data in the subdirectories).

Finally click the "Save" button:

Create a new Coverage using the new ImageMosaic CoverageStore:

1. Go to "Data Panel | Layers" via the web interface and click 'Add a new resource'. Finally choose the name of the Store you just created:

Layer Chooser

2. Click on the layer you wish to configure and you will be presented with the Coverage Editor:

Coverage Editor

3. Make sure there is a value for "Native SRS", then click the Submit button. If the "Native CRS" is 'UNKNOWN', you must to declare the SRS specifying him in the "Declared SRS" field. Hopefully there are no errors.
4. Click on the Save button.

Once you complete the preceding operations it is possible to access the OpenLayers map preview of the created mosaic.

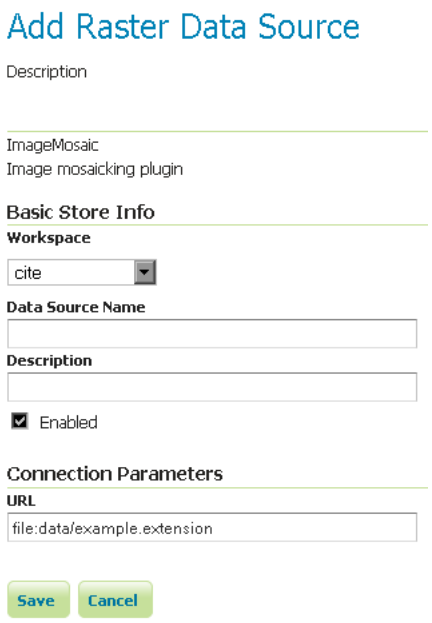
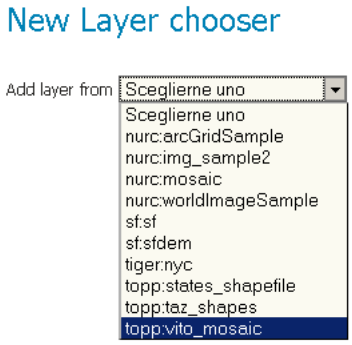
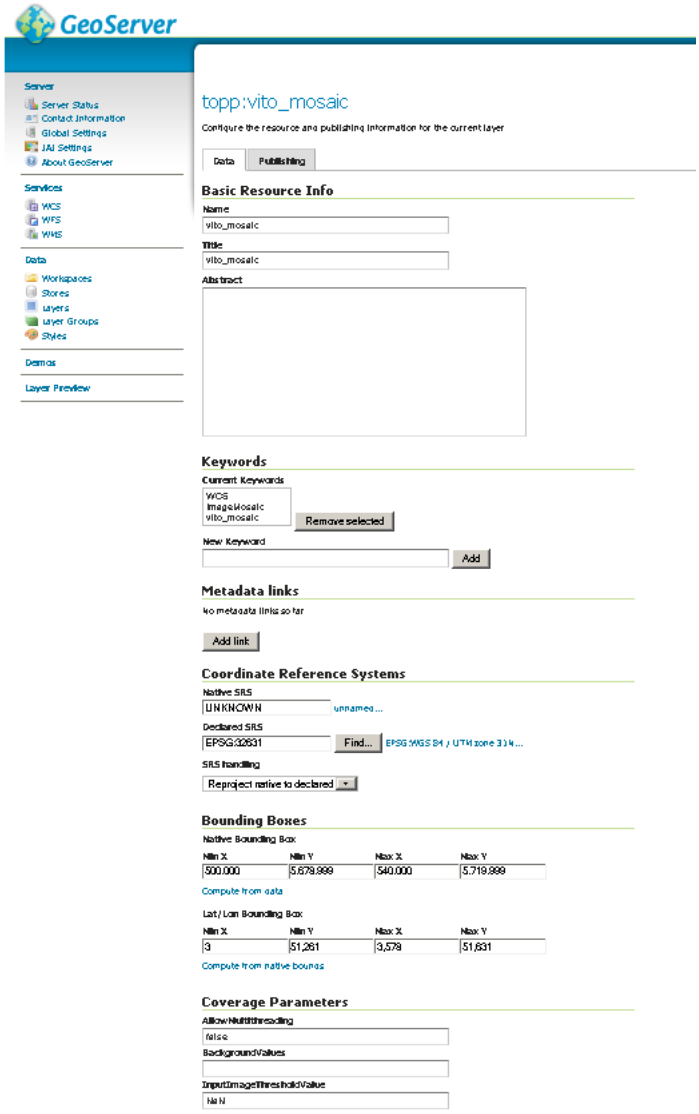


Figure 21.29: *Configuring an ImageMosaic data store*




GeoServer
Server
Server Status
Contact Information
Global Settings
JAI Settings
About GeoServer
Services
WCS
WFS
WMS
Data
Workspaces
Stores
Layers
Layer Groups
Styles
Demos
Layer Preview
topp:vito_mosaic
Configure the resource and publishing information for the current layer
Data
Publishing
Basic Resource Info
Name
vito_mosaic
Title
vito_mosaic
Abstract
Keywords
Current Keywords
WCS
ImageMosaic
vito_mosaic
Remove selected
New Keyword
Add
Metadata links
No metadata links so far
Add link
Coordinate Reference Systems
Native SRS
UNKNOWN
unnamed ...
Declared SRS
EPSG:32631
Find...
EPSG:WGS 84 / UTM zone 31N ...
SRS handling
Reproject native to declared
Bounding Boxes
Native Bounding Box
Min X
Min Y
Max X
Max Y
500,000
5,679,999
540,000
5,719,999
Compute from data
Lat/Lon Bounding Box
Min X
Min Y
Max X
Max Y
3
51,261
3,579
51,631
Compute from native bounds
Coverage Parameters
AllowMultithreading
false
BackgroundValues
InputImageThresholdValue
NaN

Warning: In case the created layer appears to be all black it might be that GeoServer has not found no acceptable granules in the provided ImageMosaic index. It is possible that the shapefile index empty (not granules where found in in the provided directory) or it might be that the granules' paths in the shapefile index are not correct as it might happen in case we have moved an existing index using absolute paths to another place. If the shapefile index paths are not correct the dbf file can be opened and fixed with, as an instance OpenOffice. As an alternative on could simple delete the index and let GeoServer recreate it from the root directory.

Tweaking an ImageMosaic CoverageStore:

The Coverage Editor gives users the possibility to set a few control parameters to further tweak and/or control the mosaic creation process. Such parameters are as follows:

Parameter	Description
MaxAllowedTiles	Set the maximum number of the tiles that can be load simulatenously for a request. In case of a large mosaic this parameter should be opportunely set to not saturating the server with too many granules loaded at the same time.
BackgroundValues	Set the value of the mosaic background. Depending on the nature of the mosaic it is wise to set a value for the 'no data' area (usually -9999). This value is repeated on all the mosaic bands.
Filter	Set the default mosaic filter. It should be a valid ECQL query which will be used as default if no 'cql_filter' are specified (instead of Filter.INCLUDE). If the cql_filter is specified in the request it will be overriden.

Note: Do not use this filter to change time or elevation dimensions defaults. It will be added as AND condition with CURRENT for 'time' and LOWER for 'elevation'.

- – *OutputTransparentColor*
  - – Set the transparent color for the created mosaic. See below for an example:

OutputTransparentColor parameter configured with 'no color'

OutputTransparentColor parameter configured with 'no data' color

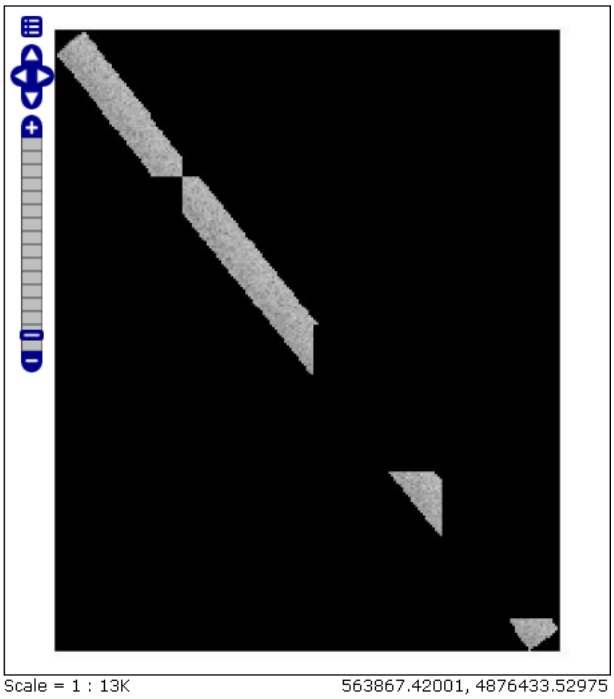
InputTransparentColor	Set the transparent color for the granules prior to mosaicing them in order to control the superimposition process between them. When GeoServer composes the granules to satisfy the user request, some of them can overlap some others, therefore, setting this parameter with the opportune color avoids the overlap of 'no data' areas between granules. See below for an example:

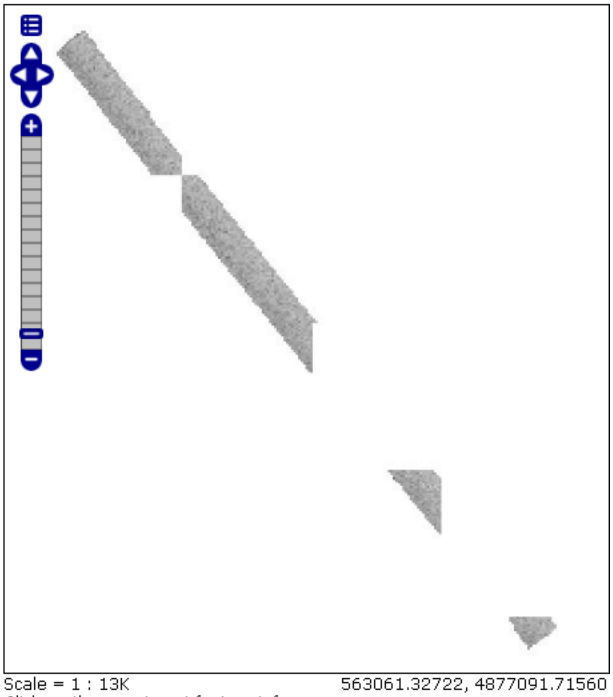
InputTransparentColor parameter not configured

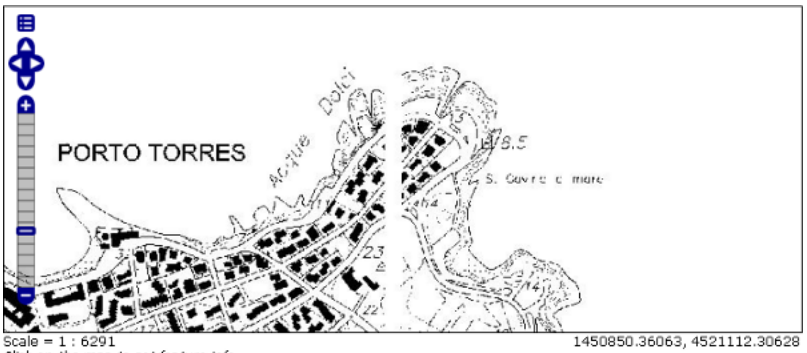
InputTransparentColor parameter configured

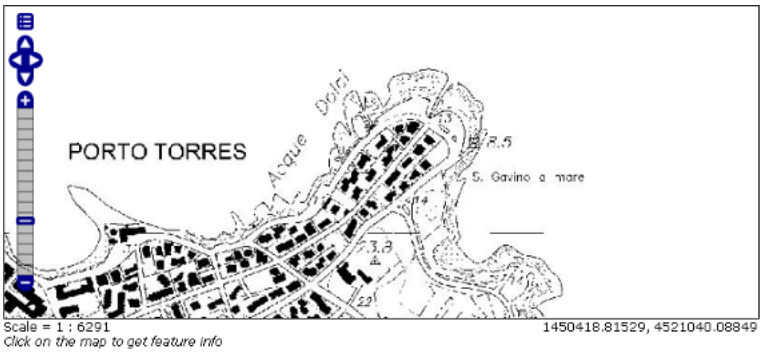
AllowMultithreading	If true enable tiles multithreading loading. This allows to perform parallelized loading of the granules that compose the mosaic.
USE_JAI_IMAGEREAD	Controls the low level mechanism to read the granules. If 'true' GeoServer will make use of JAI ImageRead operation and its deferred loading mechanism, if 'false' GeoServer will perform direct ImageIO read calls which will result in immediate loading.
SUGGESTED_TILE_SIZE	Controls the tile size of the input granules as well as the tile size of the output mosaic. It consists of two positive integersseparated by a comma,like 512,512.

Note: Deferred loading consumes less memory since it uses a streaming approach to load in memory only


Scale = 1 : 13K
563867.42001, 4876433.52975


Scale = 1 : 13K
563061.32722, 4877091.71560
Click on the map to get feature info





the data that is needed for the processing at each time, but, on the other side, may cause problems under heavy load since it keeps granules' files open for a long time to support deferred loading.

Note: Immediate loading consumes more memory since it loads in memory the whole requested mosaic at once, but, on the other side, it usually performs faster and does not leave room for "too many files open" error conditions as it happens for deferred loading.

21.9.6 Configuration examples

Now we are going to provide a few examples of mosaic configurations to demonstrate how we can make use of the ImageMosaic parameters.

DEM/Bathymetric mosaic configuration (raw data)

Such a mosaic can be use to serve large amount of data which represents altitude or depth and therefore does not specify colors directly while it reather needs an SLD to generate pictures. In our case we have a DEM dataset which consists of a set of raw geotiff files.

The first operation is to create the CoverageStore following the three steps showed in 'Create a new CoverageStore' specifying, for example, the path of the shapefile in the 'URL' field. Inside the Coverage Editor, Publishing tab - Default Title section, you can specify the 'dem' default style (Default Style combo box) in order to represent the visualization style of the mosaic. The following is an example style:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StyledLayerDescriptor version="1.0.0"
  xmlns="http://www.opengis.net/sld" xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
  xsi:schemaLocation="http://www.opengis.net/sld      http://schemas.opengis.net/sld/1.0.0/StyledLaye
  <NamedLayer>
    <Name>gtopo</Name>
    <UserStyle>
      <Name>dem</Name>
      <Title>Simple DEM style</Title>
      <Abstract>Classic elevation color progression</Abstract>
      <FeatureTypeStyle>
        <Rule>
          <RasterSymbolizer>
            <Opacity>1.0</Opacity>
            <ColorMap>
              <ColorMapEntry color="#000000" quantity="-9999" label="nodata" opacity="1.0" />
              <ColorMapEntry color="#AAFFAA" quantity="0" label="values" />
              <ColorMapEntry color="#00FF00" quantity="1000" label="values" />
              <ColorMapEntry color="#FFFF00" quantity="1200" label="values" />
              <ColorMapEntry color="#FF7F00" quantity="1400" label="values" />
              <ColorMapEntry color="#BF7F3F" quantity="1600" label="values" />
              <ColorMapEntry color="#000000" quantity="2000" label="values" />
            </ColorMap>
          </RasterSymbolizer>
        </Rule>
      </FeatureTypeStyle>
    </UserStyle>
  </NamedLayer>
</StyledLayerDescriptor>
```

In this way you have a clear distinction between the different intervals of the dataset that compose the mosaic, like the background and the 'no data' area.

Note: The 'no data' on the sample mosaic is -9999, on the other side the default background value is for mosaics is '0.0'.

The result is the following.

By setting in opportune ways the other configuration parameters, it is possible to improve at the same time both the appearance of the mosaic as well as the its performances. As an instance we could:

1. Make the 'no data' areas transparent and coherent with the real data. To achieve this we need to change the opacity of the 'no data' ColorMapEntry in the 'dem' style to '0.0' and set 'BackgroundValues' parameter at '-9999' so that empty areas will be filled with this value. The result is as follows:
2. Allow multithreaded granules loading. By setting the 'AllowMultiThreading' parameter to tru GeoServer will load the granules in parallell sing multiple threads with a consequent increase of the performances on some architectures..

The configuration parameters are the followings:

1. MaxAllowedTiles: 2147483647
2. BackgroundValues: -9999.
3. OutputTransparentColor: 'no color'.
4. InputImageThresholdValue: NaN.
5. InputTransparentColor: 'no color'.
6. AllowMultiThreading: true.
7. USE_JAI_IMAGEREAD: true.

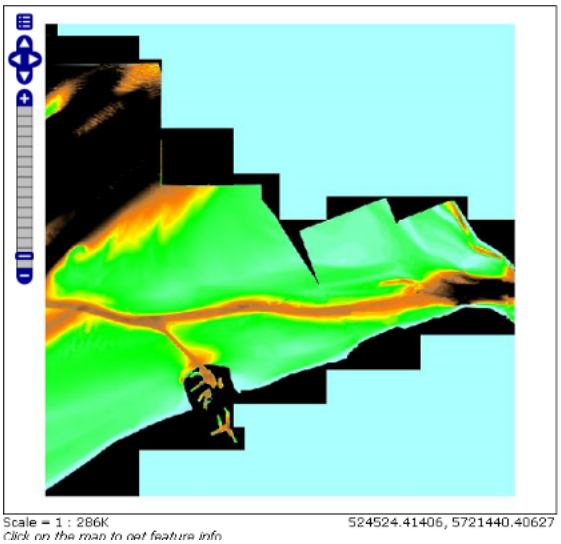


Figure 21.30: *Basic configuration*

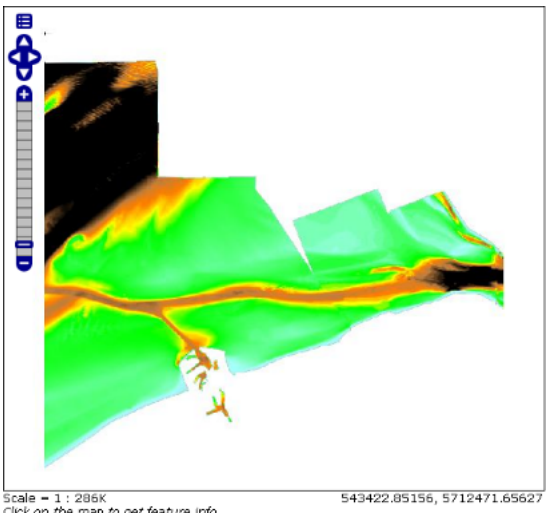


Figure 21.31: *Advanced configuration*

8. SUGGESTED_TILE_SIZE: 512,512.

Aerial Imagery mosaic configuration

In this example we are going to create a mosaic that will serve aerial imagery, RGB geotiffs in this case. Noticed that since we are talking about visual data, in the Coverage Editor you can use the basic 'raster' style, as reported here below, which is just a stub SLD to instruct the GeoServer raster renderer to not do anything particular in terms of color management:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StyledLayerDescriptor version="1.0.0"
  xmlns="http://www.opengis.net/sld" xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.opengis.net/sld      http://schemas.opengis.net/sld/1.0.0/StyledLaye
  <NamedLayer>
    <Name>raster</Name>
    <UserStyle>
      <Name>raster</Name>
      <Title>Raster</Title>
      <Abstract>A sample style for rasters, good for displaying imagery       </Abstract>
      <FeatureTypeStyle>
        <FeatureTypeName>Feature</FeatureTypeName>
        <Rule>
          <RasterSymbolizer>
            <Opacity>1.0</Opacity>
          </RasterSymbolizer>
        </Rule>
      </FeatureTypeStyle>
    </UserStyle>
  </NamedLayer>
</StyledLayerDescriptor>
```

The result is the following.

Note: Those ugly black areas, are the resulting of applying the eafalt mosaic parameters to a mosaic that does not entirey cover its bounding box. The areas within the BBOX that are not covered with data will de-

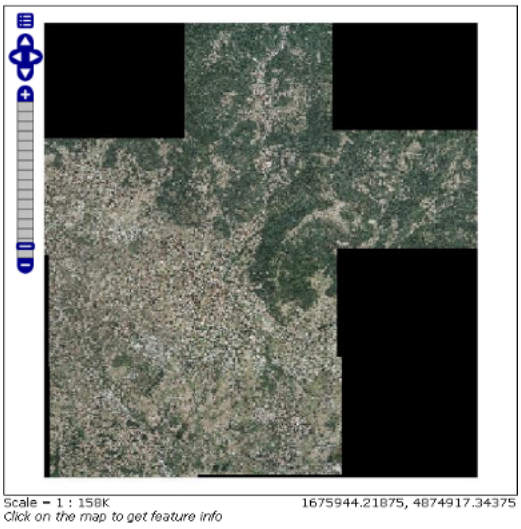


Figure 21.32: *Basic configuration*

fault to a value of 0 on each band. Since this mosaic is RGB wecan simply set the OutputTransparentCOlor to 0,0,0 in order to get back transparent fills for the BBOX.

The various parameters can be set as follows:

1. MaxAllowedTiles: 2147483647
2. BackgroundValues: default value.
3. OutputTransparentColor: #000000 (to make transparent the background).
4. InputImageThresholdValue: NaN.
5. InputTransparentColor: 'no color'.
6. AllowMultiThreading: true (in this way GeoServer manages the loading of the tiles in parallel mode with a consequent increase of the performances).
7. USE_JAI_IMAGEREAD: true.
8. SUGGESTED_TILE_SIZE: 512,512.

The results is the following:

Scanned Maps mosaic configuration

In this case we want to show how to serve scanned maps (mostly B&W images) via a GeoServer mosaic.

In the Coverage Editor you can use the basic 'raster' style as shown above since there is not need to use any of the advanced RasterSymbolizer capabilities.

The result is the following.

This mosaic, formed by two single granules, shows a typical case where the 'no data' collar areas of the granules overlap, as it is shown in the picture above. In this case we can use the 'InputTrasparentColor' parameter at to make the collar areas disappear during the superimposition process, as instance, in this case, by using the '#FFFFFF' 'InputTrasparentColor'.

This is the result:

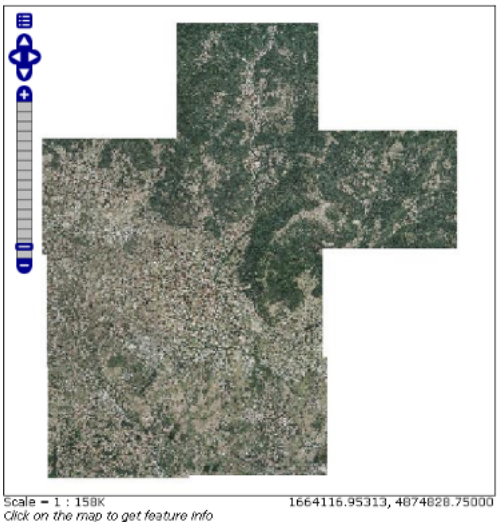


Figure 21.33: *Advanced configuration*

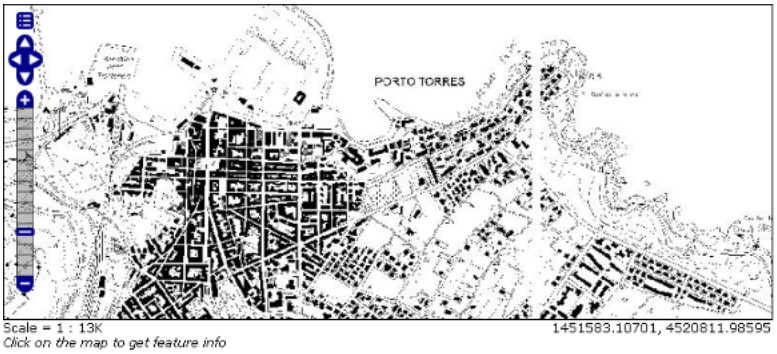


Figure 21.34: *Basic configuration*

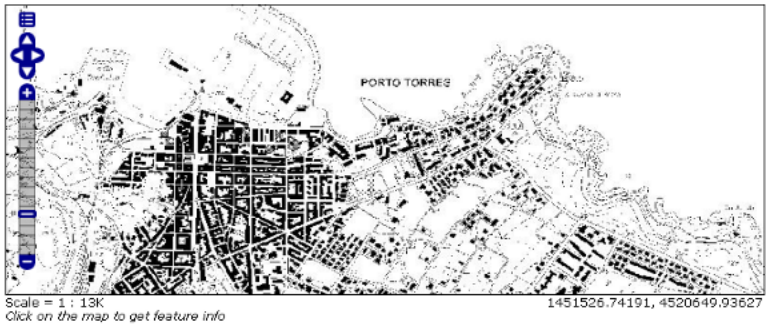


Figure 21.35: *Advanced configuration*

The final configuration parameters are the followings:

1. MaxAllowedTiles: 2147483647
2. BackgroundValues: default value.
3. OutputTransparentColor: 'no color'.
4. InputImageThresholdValue: NaN.
5. InputTransparentColor: #FFFFFF.
6. AllowMultiThreading: true (in this way GeoServer manages the loading of the tiles in parallel mode with a consequent increase of the performances).
7. USE_JAI_IMAGEREAD: true.
8. SUGGESTED_TILE_SIZE: 512,512.

21.10 Using the ImageMosaic plugin for raster time-series data

21.10.1 Introduction

This step-by-step tutorial describes all the steps for building a time-series coverage using the new ImageMosaic plugin. The ImageMosaic plugin allows the creation of a time-series layer of a raster dataset. The single images are hold in a queryable structure in order to access to a specific dataset with a temporal filter.

The concepts explained in *Using the ImageMosaic plugin* are required in order to properly understand the steps shown here.

This tutorial is organized in 4 chapter:

- The first chapter, **Configuration**, describes the environment configurations needed before load an Imagemosaic store from geoserver
- The second chapter, **Configuration examples**, describes the details, providing examples, of the configurations files needed.
- The last 2 chapters, **Coverage based on filestore** and **Coverage based on database** describe, once the previous configurations steps are done, how to create and configure an Imagemosaic store using the geoserver GUI.

The dataset used in the tutorial can be downloaded `Here`. It contains 3 image files and a .sld file representing a style needed for correctly render the images.

21.10.2 Configuration

In order to load a new CoverageStore from the GeoServer GUI two steps are needed:

1. Create a new directory in which you store all your tif files (the mosaic granules) and three configuration files. This directory represents the **MOSAIC_DIR**.
2. Install and setup a DBMS instance, this DB is that one where the mosaic indexes will be stored.
3. Another important thing is that the web container where geoserver is deployed must have the **timezone properly configured**.

In order to set the time in Coordinated Universal Time (UTC) add this switch when launching the java process:

```
-Duser.timezone=GMT
```

If a shapefile is used (see next chapter) also this switch is needed in order to manage properly the timezone:

```
-Dorg.geotools.shapefile.datetime=true
```

Note: The above properties enables support for timestamp (date and time) data in Shapefile stores. Support for timestamp is not part of the DBF standard, which only supports Date instead, and only few applications understand it. As long as shapefiles are only used for GeoServer input that is not a problem, but the above setting will cause issues if you have WFS enabled and users also download shapefiles as GetFeature output: if the feature type extracted has timestamps the generated shapefile will have as well, making it difficult to use the generated shapefile in desktop applications. As a rule of thumb, if you also need WFS support it is advisable to use an external store (PostGIS, Oracle) instead. Of course, if all that's needed is a date, using shapefile as an index without the above property is fine as well.

MOSAIC_DIR and the Configuration Files

The user can name the and place the **MOSAIC_DIR** as and where he wants.

The **MOSAIC_DIR** contains all mosaic granules files and the 3 needed configuration files. The files are in `.properties` format.

Note: Every tif file must follow the same naming convention. In this tutorial will be used {coverage-name}_{timestamp}.tif

---

In a properties file you specify your properties in a key-value manner: e.g. *myproperty=myvalue*

The configuration files needed are:

1. **datastore.properties**: contains all relevant information responsible for connecting to the database in which the spatial indexes of the mosaic will be stored
2. **indexer.properties**: specifies the name of the time-variant attribute, the elevation attribute and the type of the attributes
3. **timeregex.properties**: specifies the regular expression used to extract the time information from the filename.

All the configuration files must be placed in the root of the **MOSAIC_DIR**. The granule images could be placed also in **MOSAIC_DIR** subfolders.

Please note that **datastore.properties** isn't mandatory. The plugin provides two possibilities to access to time series data:

- **Using a shapefile** in order to store the granules indexes. That's the default behavior without providing the *datastore.properties* file.
- **Using a DBMS**, which maps the timestamp to the corresponding raster source. The former uses the **time** attribute for access to the granule from the mapping table.

For production installation is strong recommended the usage of a DBMS instead of shapefile in order to improve performances.

Otherwise the usage of a shapefile could be useful in development and test environments due to less configurations are needed.

datastore.properties

Here is shown an example of datastore.properties suitable for Postgis.

Parameter	Mandatory	Description
SPI	**Y**	The factory class used for the datastore e.g. org.geotools.data.postgis.PostgisNGDataStoreFactory
host	**Y**	The host name of the database.
port	**Y**	The port of the database
database	**Y**	The name/instance of the database.
schema	**Y**	The name of the database schema.
user	**Y**	The database user.
passwd	**Y**	The database password.
Loose bbox	**N** default 'false'	Boolean value to specify if loosing the bounding box.
Estimated extend	**N** default 'true'	Boolean values to specify if the extent of the data should be estimated.
validate connections	**N** default 'true'	Boolean value to specify if the connection should be validated.
Connection timeout	**N** default '20'	Specifies the timeout in minutes.
preparedStatements	**N** default 'false'	Boolean flag that specifies if for the database queries prepared statements should be used. This improves performance, because the database query parser has to parse the query only once

Note: The first 8 parameters are valid for each DBMS used, the last 4 may vary from different DBMS. for more information see GeoTools JDBC documentation

indexer.properties

Parameter	Mandatory	Description
TimeAttribute	N	Specifies the name of the time-variant attribute
ElevationAttribute	N	Specifies the name of the elevation attribute.
Schema	Y	A coma separated sequence that describes the mapping between attribute and the data type.
PropertyCollectors	Y	Specifies the extractor classes.

Warning: **TimeAttribute** is not a mandatory param but for the purpose of this tutorial is needed.

timeregex.properties

Parameter	Mandatory	Description
regex	Y	Specifies the pattern used for extracting the information from the file

After this you can create a new imagemosaic datastore.

Install and setup a DBMS instance

First of all note that the usage of a DBMS to store the mosaic indexes **isn't mandatory**. If the user don't place in the MOSAIC_DIR the datastore.properties file the plugin uses a **shapefile**. The shapefile will be created into the MOSAIC_DIR.

Anyway, especially for large dataset, **the usage of a DBMS is strong recommended**. The ImageMosaic plugin supports all the most used DBMS.

The configuration needed are the basics: create a new empty DB with geospatial extensions, a new schema and configure the user with W/R grants.

In this tutorial will be used PostgreSQL 9.1 together with PostGIS 2.0 .

21.10.3 Configuration examples

As example is used a set of data that represents hydrological data in a certain area in South Tyrol, a region in northern Italy. The origin data were converted from asc format to tiff using the gdal utility **gdal translate**.

For this running example we will create a layer named snow.

As mentioned before the files could located in any part of the file system.

In this tutorial the chosen MOSAIC_DIR directory is called `hydroalp` and is placed under the root of the GEOSERVER_DATA_DIR.

Configure the MOSAIC_DIR:

In this part is shown an entire MOSAIC_DIR configuration.

datastore.properties:

```
SPI=org.geotools.data.postgis.PostgisNGDataStoreFactory
host=localhost
port=5432
database=db
schema=public
user=dbuser
passwd=dbpasswd
Loose\ bbox=true
Estimated\ extends=false
validate\ connections=true
Connection\ timeout=10
preparedStatements=true
```

Note: In case of a missing datastore.properties file a shape file is created for the use of the indexes.

Granules Naming Convention

Here an example of the granules naming that satisfy the rule shown before:

```
$ls hydroalp/snow/*.tif

snow/snow_20091001.tif
snow/snow_20091101.tif
snow/snow_20091201.tif
snow/snow_20100101.tif
snow/snow_20100201.tif
snow/snow_20100301.tif
snow/snow_20100401.tif
snow/snow_20100501.tif
snow/snow_20100601.tif
snow/snow_20100701.tif
snow/snow_20100801.tif
snow/snow_20100901.tif
```

timeregex.properties:

In the timeregex property file you specify the pattern how the date(time) in the file looks like. In this example it consists simply of 8 digits as specified below.

```
regex=[0-9]{8}
```

indexer.properties:

Here the user can specify the information that needs Geoserver for creating the table in the database. In this table the time values are stored in the column ingestion.

```
TimeAttribute=ingestion
ElevationAttribute=elevation
Schema=*the_geom:Polygon,location:String,ingestion:java.util.Date,elevation:Integer
PropertyCollectors=TimestampFileNameExtractorSPI[timeregex](ingestion)
```

21.10.4 Create and Publish an Imagemosaic store:

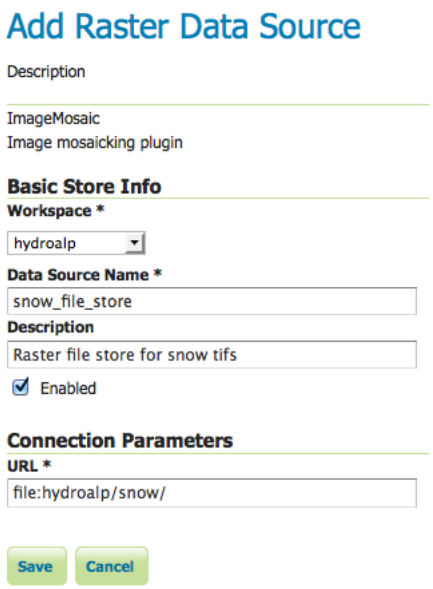
Step 1: create new imagemosaic data store

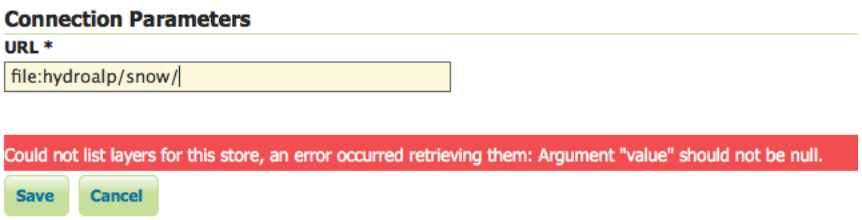
We create a new data store of type raster data and choose ImageMosaic.

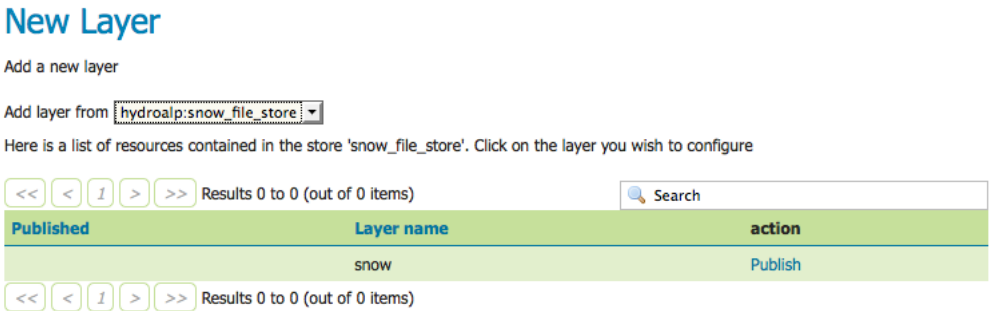
Note: Be aware that Geoserver creates a table which is identical with the name of your layer. If the table already exists, it will not be dropped from the DB and the following error message appear. The same message appear, if the generated property file already exists in the directory or there are wrong connection parameters in datastore.properties file.

Step 2: Specify layer

We specify the directory in which the property and tif files are located (path must end with a slash) and add the layer.

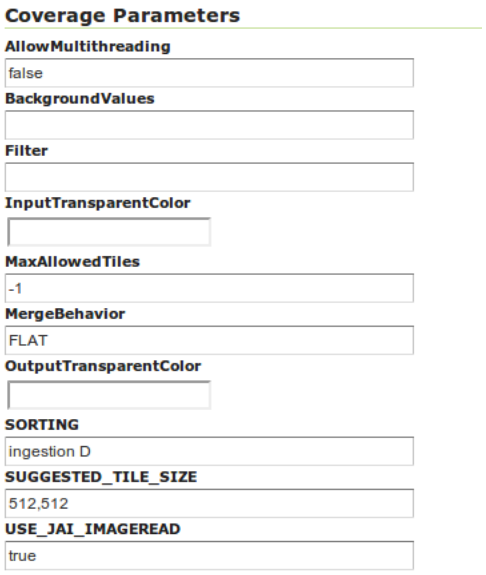

Add Raster Data Source
Description
ImageMosaic
Image mosaicking plugin
Basic Store Info
Workspace *
hydroalp
Data Source Name *
snow_file_store
Description
Raster file store for snow tifs
Enabled
Connection Parameters
URL *
file:hydroalp/snow/
Save
Cancel


Connection Parameters
URL *
file:hydroalp/snow/
Could not list layers for this store, an error occurred retrieving them: Argument "value" should not be null.
Save
Cancel


New Layer
Add a new layer
Add layer from hydroalp:snow_file_store
Here is a list of resources contained in the store 'snow_file_store'. Click on the layer you wish to configure
<< < 1 > >> Results 0 to 0 (out of 0 items)
Search
Published
Layer name
action
snow
Publish
<< < 1 > >> Results 0 to 0 (out of 0 items)

Step 3: set Coverage Parameter

The relevant parameters are AllowMultithreading and USE_JAI_IMAGEREAD. Do not forget to specify the background value according to your the value in your tif file. If you want to control which granule is displayed when a number of images match the time period specified then set the SORTING parameter to the variable name you want to use for sorting followed by a space and either D or A for descending or ascending. Descending values will mean that the latest image in a series that occurs in the time period requested is shown.



Remember that for display correctly the images contained in the provided dataset a custom style is needed.

Set as default style the *snow_style.sld* contained in the dataset archive.

More information about raster styling can be found at chapter *Rasters*

Step 4: set temporal properties

In the tab Dimensions you can specify how the time attributes are represented.

By enabling the Time or Elevation checkbox you can specify the way of presentation. In this example query is performed just only over the time attribute.

Below is shown a snippet of the Capabilities document for each presentation case:

Setting the presentation to **List**, all mosaic times are listed:

```
<Dimension name="time" default="current" units="ISO8601">
        2009-10-01T00:00:00.000Z,2009-11-01T00:00:00.000Z,2009-12-01T00:00:00.000Z,2010-01-01T00:00:0
</Dimension>
```

Setting the presentation to **Continuous interval** only the start, end and interval extent times are listed:

```
<Dimension name="time" default="current" units="ISO8601">
        2009-10-01T00:00:00.000Z/2011-09-01T00:00:00.000Z/P1Y11MT10H
</Dimension>
```

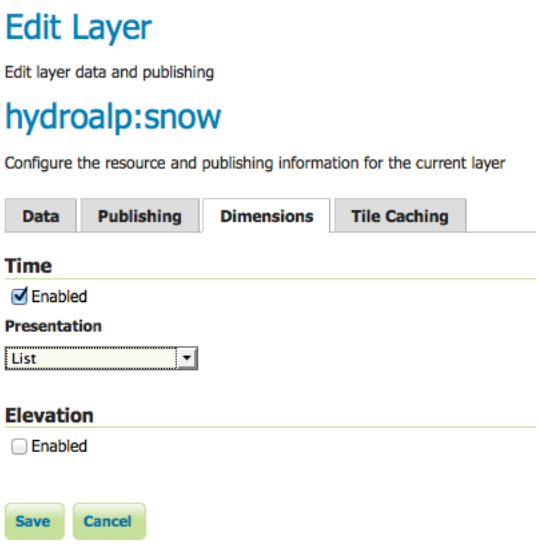
Setting the presentation to **Interval and resolutions** gives to user the possibility to specify the resolutions of the interval:

```
<Dimension name="time" default="current" units="ISO8601">
        2009-10-01T00:00:00.000Z/2011-09-01T00:00:00.000Z/P1DT12H
</Dimension>
```

In this case the resolution is set to one day and half

Note: For visualize the getCapabilities document go to geoserver homepage, under the right tab called **Service Capabilities** click on the WMS 1.3.0 link.

For this tutorial the Presentation attribute is set to **List**



After this steps the new layer is available in Geoserver. Additionally Geoserver has created in the source directory a property file and on the database he has created a table named with the name of the layer.

Generated property file:

```
#-Automagically created from GeoTools-
#Sat Oct 13 10:47:08 CEST 2012
Levels=100.0,100.0
Heterogeneous=false
ElevationAttribute=elevation
TimeAttribute=ingestion
AbsolutePath=false
Name=snow
Caching=false
ExpandToRGB=false
LocationAttribute=location
SuggestedSPI=it.geosolutions.imageioimpl.plugins.tiff.TIFFImageReaderSpi
LevelsNum=1
```

Note: The parameter **Caching=false** is important because in this way GeoServer doesn't cache any data. So the user is able to update manually the mosaic adding and removing granules to MOSAIC_DIR and update the relative entry on DB.

Generated table:

	fid [PK] serial	the_geom geometry	location character varying(255)	ingestion timestamp wi	elevation integer
1	1	01030000	snow_20091001.tif	2009-10-01	
2	2	01030000	snow_20091101.tif	2009-11-01	
3	3	01030000	snow_20091201.tif	2009-12-01	
4	4	01030000	snow_20100101.tif	2010-01-01	
5	5	01030000	snow_20100201.tif	2010-02-01	
6	6	01030000	snow_20100301.tif	2010-03-01	
7	7	01030000	snow_20100401.tif	2010-04-01	
8	8	01030000	snow_20100501.tif	2010-05-01	
9	9	01030000	snow_20100601.tif	2010-06-01	
10	10	01030000	snow_20100701.tif	2010-07-01	
11	11	01030000	snow_20100801.tif	2010-08-01	
12	12	01030000	snow_20100901.tif	2010-09-01	

Note: The user must create manually the index on the table in order to speed up the search by attribute.

Step 5: query layer on timestamp:

In order to display a snapshot of the map at a specific time instant you have to pass in the request an additional time parameter with a specific notation **&time=** < **pattern** > where you pass a value that corresponds to them in the filestore. The only thing is the pattern of the time value is slightly different.

For example if an user wants to obtain the snow coverage images from the months **Oct,Nov,Dec 2009** I pass in each request **&time=2009-10-01**, **&time=2009-11-01** and **&time=2009-12-01**. You can recognize in the three images how the snow coverage changes. Here the color blue means a lot of snow.

21.10.5 Create and publish a Layer from mosaic indexes:

After the previous steps it is also be possible to create a layer that represents the spatial indexes of the mosaic. This is an useful features when is required to handle large dataset of Mosaics with High Resolutions granules, the user can easily get the footprints of the Images. In this case will be rendered only the geometries stored on the indexes tables.

Step 1: add a postgis datastore:

and specify the connection parameters

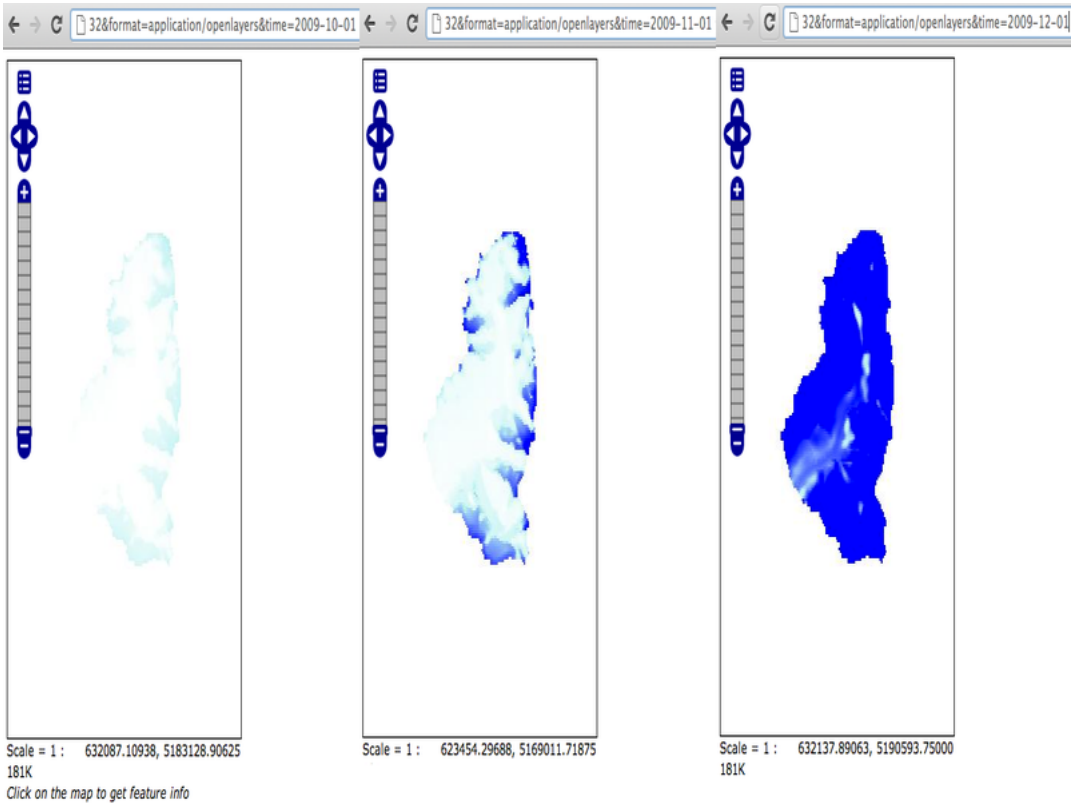
Step 2: add database layer:

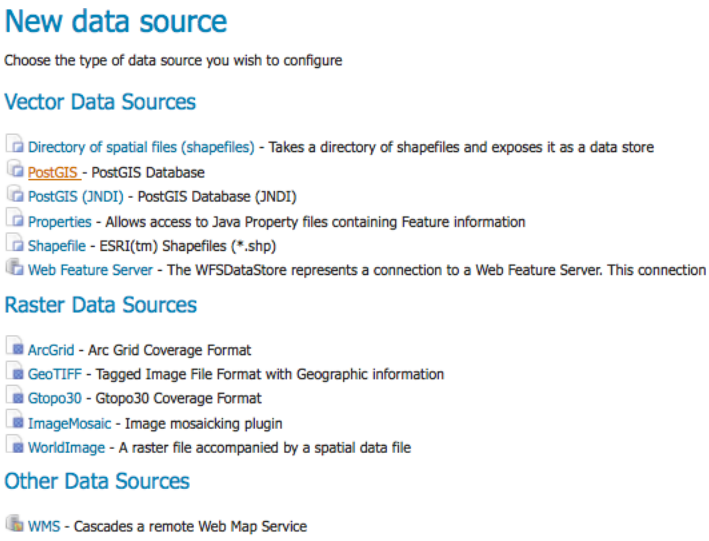
Choose from the created datastore the table that you want to publish as a layer.

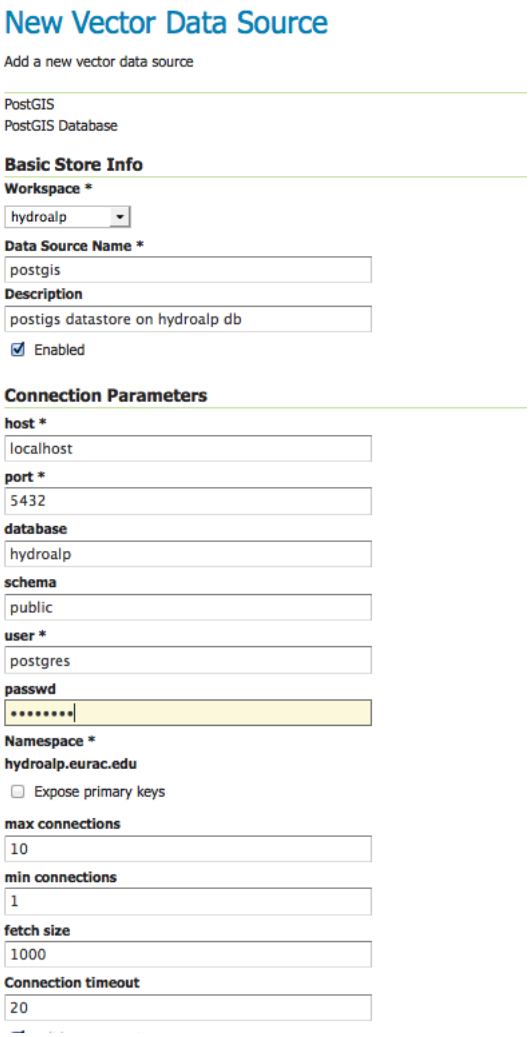
Step 3: specify dimension:

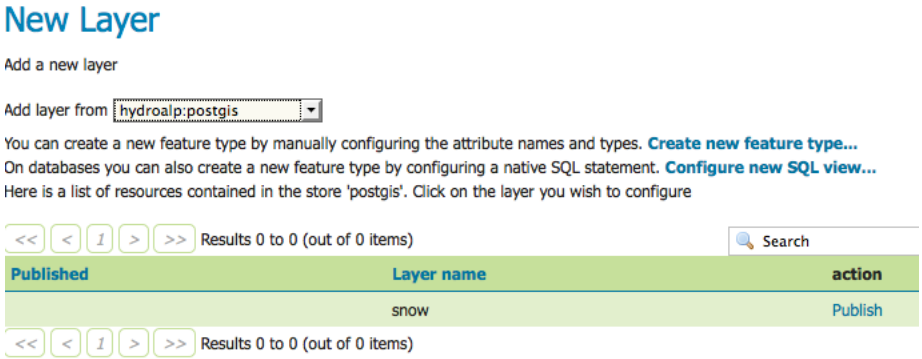
In the tab Dimension specify the time-variant attribute and the form of presentation.

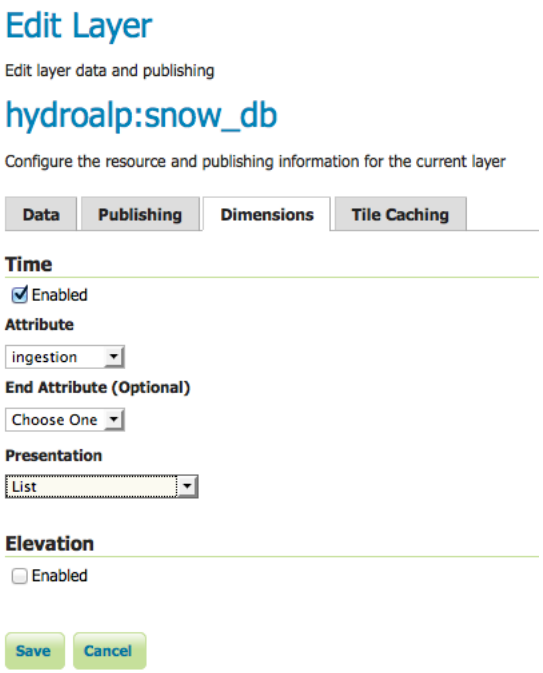
That's it. Now is possible query this layer too.


32&format=application/openlayers&time=2009-10-01
32&format=application/openlayers&time=2009-11-01
32&format=application/openlayers&time=2009-12-01
Scale = 1 : 632087.10938, 5183128.90625
181K
Click on the map to get feature info
Scale = 1 : 623454.29688, 5169011.71875
Scale = 1 : 632137.89063, 5190593.75000
181K


New data source
Choose the type of data source you wish to configure
Vector Data Sources
Directory of spatial files (shapefiles) - Takes a directory of shapefiles and exposes it as a data store
PostGIS - PostGIS Database
PostGIS (JNDI) - PostGIS Database (JNDI)
Properties - Allows access to Java Property files containing Feature information
Shapefile - ESRI(tm) Shapefiles (*.shp)
Web Feature Server - The WFSDataStore represents a connection to a Web Feature Server. This connection
Raster Data Sources
ArcGrid - Arc Grid Coverage Format
GeoTIFF - Tagged Image File Format with Geographic information
Gtopo30 - Gtopo30 Coverage Format
ImageMosaic - Image mosaicking plugin
WorldImage - A raster file accompanied by a spatial data file
Other Data Sources
WMS - Cascades a remote Web Map Service


New Vector Data Source
Add a new vector data source
PostGIS
PostGIS Database
Basic Store Info
Workspace *
hydroalp
Data Source Name *
postgis
Description
postigs datastore on hydroalp db
Enabled
Connection Parameters
host *
localhost
port *
5432
database
hydroalp
schema
public
user *
postgres
passwd
Namespace *
hydroalp.eurac.edu
Expose primary keys
max connections
10
min connections
1
fetch size
1000
Connection timeout
20


New Layer
Add a new layer
Add layer from hydroalp:postgis
You can create a new feature type by manually configuring the attribute names and types. Create new feature type...
On databases you can also create a new feature type by configuring a native SQL statement. Configure new SQL view...
Here is a list of resources contained in the store 'postgis'. Click on the layer you wish to configure
<< < 1 > >> Results 0 to 0 (out of 0 items)
Search
Published
Layer name
action
snow
Publish
<< < 1 > >> Results 0 to 0 (out of 0 items)



21.11 Using the ImageMosaic plugin for raster with time and elevation data

21.11.1 Introduction

This tutorial is the following of *Using the ImageMosaic plugin for raster time-series data* and explains how manage an Imagemosaic using both **Time** and **Elevation** attributes.

The dataset used is a set of raster images used in weather forecast, representing the temperature in a certain zone at different times and elevations.

All the steps explained in chapter *Configurations* of *Using the ImageMosaic plugin for raster time-series data* are still the same.

This tutorial explains just how to configure the **elevationregex.properties** that is an additional configuration file needed, and how to modify the **indexer.properties**.

The dataset used is different so also a fix to the **timeregex.properties** used in the previous tutorial is needed.

Will be shown also how query geoserver asking for layers specifying both time and elevation dimensions.

The dataset used in the tutorial can be downloaded `Here`

21.11.2 Configuration examples

The additional configurations needed in order to handle also the elevation attributes are:

- Improve the previous version of the *indexer.properties* file
- Add the *elevationregex.properties file* in order to extract the elevation dimension from the filename

indexer.properties:

Here the user can specify the information that needs Geoserver for creating the table in the database.

In this case the time values are stored in the column ingestion as shown in the previous tutorial but now is mandatory specify the elevation column too.

```
Caching=false
TimeAttribute=ingestion
ElevationAttribute=elevation
Schema=*the_geom:Polygon,location:String,ingestion:java.util.Date,elevation:Double
PropertyCollectors=TimestampFileNameExtractorSPI[timeregex](ingestion),DoubleFileNameExtractorSPI[ele
```

elevationregex.properties:

Remember that every tif file must follow this naming convention:

```
{coveragename}_{timestamp}_[{elevation}].tif
```

As in the timeregex property file the user must specify the pattern that the elevation in the file name looks like. In this example it consists of 4 digits, a dot '.' and other 3 digits.

an example of filename, that is used in this tutorial is:

```
gfs50kmTemperature20130310T180000000Z_0600.000_.tiff
```

The geoserver imagemosaic plugin scans the filename and search for the first occurrence that match with the pattern specified. Here the content of **timeregex.properties**:

```
regex=(?<=_)(\\d{4}\\.\\d{3})(?=_)
```

timeregex.properties:

As you can see the time in this dataset is specified as ISO8601 format:

```
20130310T180000000Z
```

Instead of the form **yyyymmdd** as in the previous tutorial. So the regex to specify in timeregex.properties is:

```
regex=[0-9]{8}T[0-9]{9}Z(\?!.\*[0-9]{8}T[0-9]{9}Z.\*)
```

21.11.3 Coverage based on filestore

Once the mosaic configuration is ready the store mosaic could be loaded on geoserver.

The steps needed are the same shown the previous chapter. After the store is loaded and a layer published note the differences in WMS Capabilities document and in the table on postgres.

WMS Capabilities document

The WMS Capabilities document is a bit different, now there is also the dimension **elevation**. In this example both time and elevation dimension are set to **List** .

```
<Dimension name="time" default="current" units="ISO8601">
        2013-03-10T00:00:00.000Z,2013-03-11T00:00:00.000Z,2013-03-12T00:00:00.000Z,2013-03-13T00:00:0
</Dimension>
<Dimension name="elevation" default="200.0" units="EPSG:5030" unitSymbol="m">
        200.0,300.0,500.0,600.0,700.0,850.0,925.0,1000.0
</Dimension>
```

The table on postgres

With the elevation support enabled the table on postgres has, for each image, the field **elevation** filled with the elevation value.

	fid [PK] serial	the_geom geometry	location character varying	ingestion timestamp without time z	elevation double precis
1	1	0103000020E	gfs50kmTemperatu	2013-03-10 18:00:00	200
2	2	0103000020E	gfs50kmTemperatu	2013-03-11 00:00:00	200
3	3	0103000020E	gfs50kmTemperatu	2013-03-11 06:00:00	200
4	4	0103000020E	gfs50kmTemperatu	2013-03-11 12:00:00	200
5	5	0103000020E	gfs50kmTemperatu	2013-03-11 18:00:00	200
6	6	0103000020E	gfs50kmTemperatu	2013-03-12 00:00:00	200
7	7	0103000020E	gfs50kmTemperatu	2013-03-12 06:00:00	200
8	8	0103000020E	gfs50kmTemperatu	2013-03-12 12:00:00	200
9	9	0103000020E	gfs50kmTemperatu	2013-03-12 18:00:00	200
10	10	0103000020E	gfs50kmTemperatu	2013-03-13 00:00:00	200
11	11	0103000020E	gfs50kmTemperatu	2013-03-13 06:00:00	200
12	12	0103000020E	gfs50kmTemperatu	2013-03-13 12:00:00	200
13	13	0103000020E	gfs50kmTemperatu	2013-03-13 18:00:00	200
14	14	0103000020E	gfs50kmTemperatu	2013-03-14 00:00:00	200
15	15	0103000020E	gfs50kmTemperatu	2013-03-14 06:00:00	200
16	16	0103000020E	gfs50kmTemperatu	2013-03-14 12:00:00	200
17	17	0103000020E	gfs50kmTemperatu	2013-03-14 18:00:00	200
18	18	0103000020E	gfs50kmTemperatu	2013-03-15 00:00:00	200
19	19	0103000020E	gfs50kmTemperatu	2013-03-15 06:00:00	200
20	20	0103000020E	gfs50kmTemperatu	2013-03-15 12:00:00	200
21	21	0103000020E	gfs50kmTemperatu	2013-03-15 18:00:00	200

Note: The user must create manually the index on the table in order to speed up the search by attribute.

Query layer on timestamp:

In order to display a snapshot of the map at a specific time instant and elevation you have to pass in the request those parameters.

- **&time=** < **pattern** > , as shown before,
- **&elevation=** < **pattern** > where you pass the value of the elevation.

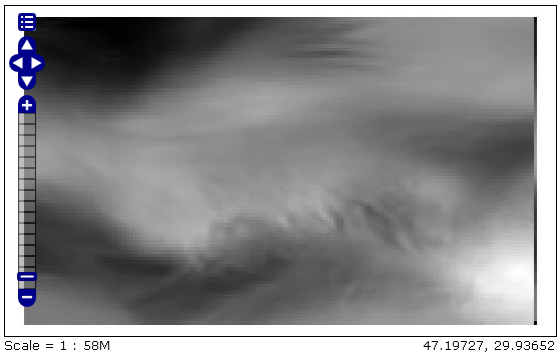
For example if an user wants to obtain the temperature coverage images for the day **2013-03-10 at 6 PM** at elevation **200 meters** must append to the request:

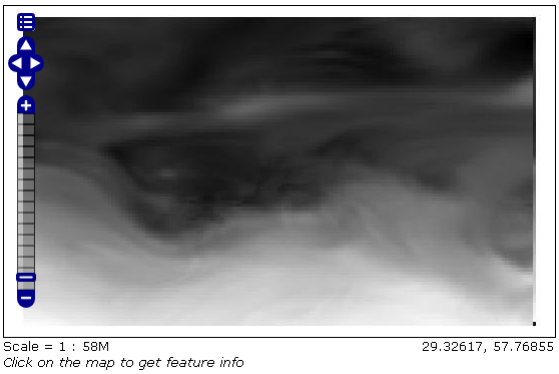
```
&time=2013-03-10T00:00:00.000Z&elevation=200.0
```

Same day at elevation **300.0 meters**:

```
&time=2013-03-10T00:00:00.000Z&elevation=300.0
```

Note that if just the time dimension is append to the request will be displayed the elevation **200 meters** (if present) because of the **default** attribute of the tag `<Dimension name="elevation" ...` in the WMS Capabilities document is set to **200**





21.12 Building and using an image pyramid

GeoServer can efficiently deal with large TIFF with overviews, as long as the TIFF is below the 2GB size limit.

Once the image size goes beyond such limit it's time to start considering an image pyramid instead.

An image pyramid builds multiple mosaics of images, each one at a different zoom level, making it so that each tile is stored in a separate file. This comes with a composition overhead to bring back the tiles into a single image, but can speed up image handling as each overview is tiled, and thus a sub-set of it can be accessed efficiently (as opposed to a single GeoTIFF, where the base level can be tiled, but the overviews never are).

This tutorial shows how to build an image pyramid with open source utilities and how to load it into GeoServer. The tutorial assumes you're running at least GeoServer 2.0.2.

21.12.1 Building a pyramid

For this tutorial we have prepared a sample BlueMarble TNG subset in GeoTIFF form. The image is tiled and JPEG compressed, without overviews. Not exactly what you'd want to use for high performance data serving, but good for redistribution and as a starting point to build a pyramid.

In order to build the pyramid we'll use the gdal_retile.py utility, part of the GDAL command line utilities and available for various operating systems (if you're using Microsoft Windows look for FWTools).

The following commands will build a pyramid on disk:

```
mkdir bmpyramid
gdal_retile.py -v -r bilinear -levels 4 -ps 2048 2048 -co "TILED=YES" -co "COMPRESS=JPEG" -targetDir
```

The gdal_retile.py user guide provides a detailed explanation for all the possible parameters, here is a description of the ones used in the command line above:

- *-v*: verbose output, allows the user to see each file creation scroll by, thus knowing progress is being made (a big pyramid construction can take hours)
- *-r bilinear*: use bilinear interpolation when building the lower resolution levels. This is key to get good image quality without asking GeoServer to perform expensive interpolations in memory
- *-levels 4*: the number of levels in the pyramid
- *-ps 2048 2048*: each tile in the pyramid will be a 2048x2048 GeoTIFF
- *-co "TILED=YES"*: each GeoTIFF tile in the pyramid will be inner tiled
- *-co "COMPRESS=JPEG"*: each GeoTIFF tile in the pyramid will be JPEG compressed (trades small size for higher performance, try out it without this parameter too)
- *-targetDir bmpyramid*: build the pyramid in the bmpyramid directory. The target directory must exist and be empty
- *bmreduced.tiff*: the source file

This will produce a number of TIFF files in bmpyramid along with the sub-directories *1*, *2*, *3*, and *4*.

Once that is done, and assuming the GeoServer image pyramid plug-in is already installed, it's possible to create the coverage store by pointing at the directory containing the pyramid and clicking save:

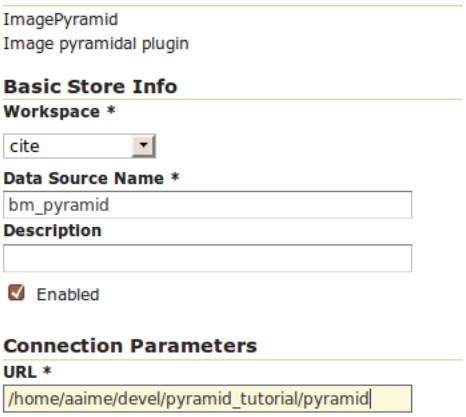


Figure 21.36: *Configuring a image pyramid store*

When clicking save the store will look into the directory, recognize a *gdal_retile* generated structure and perform some background operations:

- move all tiff files in the root to a newly create directory *0*
- create an image mosaic in all sub-directories (shapefile index plus property file)
- create the root property file describing the whole pyramid structure

Once that is done the user will be asked to choose a coverage, which will be named after the pyramid root directory:

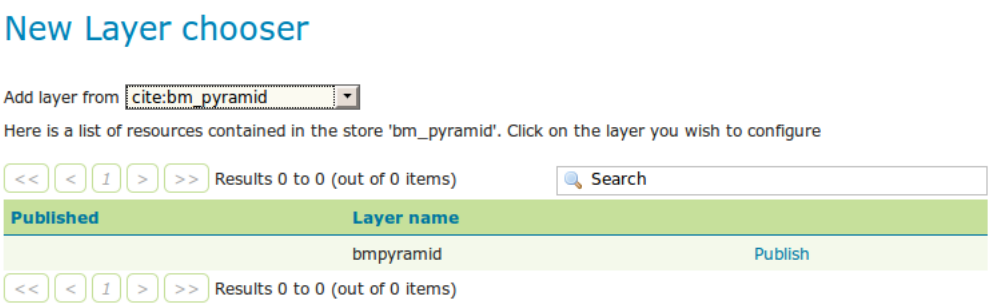


Figure 21.37: *Choosing the coverage for publishing*

Publish the layer, and then setup the layer parameter *USE_JAI_IMAGEREAD* to *false* to get better scalability:

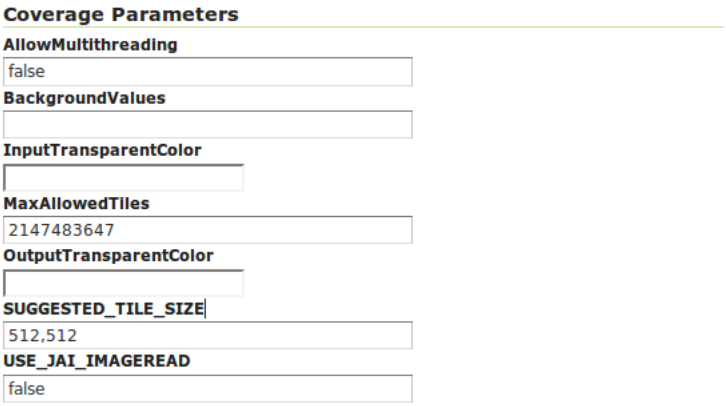


Figure 21.38: *Tuning the pyramid parameters*

Submit and go to the preview, the pyramid should be ready to use:

21.12.2 Notes on big pyramids

The code that is auto-creating the pyramid indexes and metadata files might take time to run, especially if:

- the pyramid zero level is composed of many thousands of files
- the system is busy with the disk already and that results in higher times to move all the files to the *0* directory

If the delay is too high the request to create the store will time out and might break the pyramid creation. So, in case of very big pyramids consider loosing some of the comfort and creating the *0* directory and moving the files by hand:

```
cd bmpyramid
mkdir 0
mv *.tiff 0
```

Figure 21.39: *Previewing the pyramid*

21.13 Storing a coverage in a JDBC database

Warning: The screenshots on this tutorial have not yet been updated for the 2.0.x user interface. But most all the rest of the information should be valid, and the user interface is roughly the same, but a bit more easy to use.

21.13.1 Introduction

This tutorial describes the process of storing a coverage along with its pyramids in a jdbc database. The ImageMosaic JDBC plugin is authored by Christian Mueller and is part of the geotools library.

The full documentation is available here:http://docs.geotools.org/latest/userguide/library/coverage/jdbc/index.html

This tutorial will show one possible scenario, explaining step by step what to do for using this module in GeoServer (since Version 1.7.2)

21.13.2 Getting Started

We use postgis/postgres as database engine, a database named "gis" and start with an image from openstreetmap. We also need this utility http://www.gdal.org/gdal_retile.html . The best way to install with all dependencies is downloading from here http://fwtools.maptools.org/

Create a working directory, lets call it `working` ,download this image with a right mouse click (Image save as ...) and save it as `start_rgb.png`

Check your image with:

```
gdalinfo start_rgb.png
```

This image has 4 Bands (Red,Green,Blue,Alpha) and needs much memory. As a rule, it is better to use images with a color table. We can transform with **rgb2pct** (**rgb2pct.py** on Unix).:

```
rgb2pct -of png start_rgb.png start.png
```

Compare the sizes of the 2 files.

Afterwards, create a world file `start.wld` in the `working` directory with the following content.:

```
0.0075471698
0.0000000000
0.0000000000
-0.0051020408
8.9999995849
48.9999999796
```

21.13.3 Preparing the pyramids and the tiles

If you are new to tiles and pyramids, take a quick look here http://star.pst.qub.ac.uk/idl/Image_Tiling.html

21.13.4 How many pyramids are needed ?

Lets do a simple example. Given an image with 1024x1024 pixels and a tile size with 256x256 pixels.We can calculate in our brain that we need 16 tiles. Each pyramid reduces the number of tiles by a factor of 4. The first pyramid has 16/4 = 4 tiles, the second pyramid has only 4/4 = 1 tile.

Solution: The second pyramid fits on one tile, we are finished and we need 2 pyramids.

The formula for this:

number of pyramids = log(pixelsize of image) / log(2) - log (pixelsize of tile) / log(2).

Try it: Go to Google and enter as search term "log(1024)/log(2) - log(256)/log(2)" and look at the result.

If your image is 16384 pixels , and your tile size is 512 pixels, it is

log(16384)/log(2) - log(512)/log(2) = 5

If your image is 18000 pixels, the result = 5.13570929. Thake the floor and use 5 pyramids. Remember, the last pyramid reduces 4 tiles to 1 tile, so this pyramid is not important.

If your image is 18000x12000 pixel, use the bigger dimension (18000) for the formula.

For creating pyramids and tiles, use http://www.gdal.org/gdal_retile.html from the gdal project.

The executeable for Windows users is **gdal_retile.bat** or only **gdal_retile**, Unix users call **gdal_retile.py**

Create a subdirectory `tiles` in your `working` directory and execute within the `working` directory:

```
gdal_retile -co "WORLDFILE=YES"  -r bilinear -ps 128 128 -of PNG -levels 2 -targetDir tiles start.png
```

What is happening ? We tell gdal_retile to create world files for our tiles (-co "WORLDFILE=YES"), use bilinear interpolation (-r bilinear), the tiles are 128x128 pixels in size (-ps 128 128) , the image format should be PNG (-of PNG), we need 2 pyramid levels (-levels 2) ,the directory for the result is `tiles` (-targetDir tiles) and the source image is `start.png`.

Note: A few words about the tile size. 128x128 pixel is proper for this example. Do not use such small sizes in a production environment. A size of 256x256 will reduce the number of tiles by a factor of 4, 512x512 by a factor of 16 and so on. Producing too much tiles will degrade performance on the database side (large tables) and will also raise cpu usage on the client side (more image operations).

Now you should have the following directories

- `working` containing `start.png` , `start.wld` and a subdirectory `tiles`.
- `working/tiles` containing many `*.png` files and associated `*.wld` files representing the tiles of `start.png`
- `working/tiles/1` containing many `*.png` files and associated `*.wld` files representing the tiles of the first pyramid
- `working/tiles/2` containing many `*.png` files and associated `*.wld` files representing the tiles of the second pyramid

21.13.5 Configuring the new map

The configuration for a map is done in a xml file. This file has 3 main parts.

1. The connect info for the jdbc driver
2. The mapping info for the sql tables

3. Configuration data for the map

Since the jdbc connect info and the sql mapping may be reused by more than one map, the best practice is to create xml fragments for both of them and to use xml entity references to include them into the map xml.

First, find the location of the GEOSERVER_DATA_DIR. This info is contained in the log file when starting GeoServer.:

```
---------------------------------
- GEOSERVER_DATA_DIR: /home/mcr/geoserver-1.7.x/1.7.x/data/release
---------------------------------
```

Put all configuration files into the `coverages` subdirectory of your GeoServer data directory. The location in this example is

`/home/mcr/geoserver-1.7.x/1.7.x/data/release/coverages`

1. Create a file `connect.postgis.xml.inc` with the following content

```
<connect>
  <!-- value DBCP or JNDI -->
  <dstype value="DBCP"/>
  <!--   <jndiReferenceName value=""/>  -->
  <username value="postgres" />
  <password value="postgres" />
  <jdbcUrl value="jdbc:postgresql://localhost:5432/gis" />
  <driverClassName value="org.postgresql.Driver"/>
  <maxActive value="10"/>
  <maxIdle value="0"/>
</connect>
```

The jdbc user is “postgres”, the password is “postgres”, maxActive and maxIdle are parameters of the apache connection pooling, jdbcUrl and driverClassName are postgres specific. The name of the database is “gis”.

If you deploy GeoServer into a J2EE container capable of handling jdbc data sources, a better approach is

```
<connect>
  <!-- value DBCP or JNDI -->
  <dstype value="JNDI"/>
  <jndiReferenceName value="jdbc/mydatasource"/>
</connect>
```

For this tutorial, we do not use data sources provided by a J2EE container.

2. The next xml fragment to create is `mapping.postgis.xml.inc`

```
<!-- possible values: universal,postgis,db2,mysql,oracle -->
<spatialExtension name="postgis"/>
<mapping>
    <masterTable name="mosaic" >
      <coverageNameAttribute name="name"/>
      <maxXAttribute name="maxX"/>
      <maxYAttribute name="maxY"/>
      <minXAttribute name="minX"/>
      <minYAttribute name="minY"/>
      <resXAttribute name="resX"/>
      <resYAttribute name="resY"/>
      <tileTableNameAtribute  name="TileTable" />
      <spatialTableNameAtribute name="SpatialTable" />
    </masterTable>
    <tileTable>
```

```
      <blobAttributeName name="data" />
      <keyAttributeName name="location" />
    </tileTable>
    <spatialTable>
      <keyAttributeName name="location" />
      <geomAttributeName name="geom" />
      <tileMaxXAttribute name="maxX"/>
      <tileMaxYAttribute name="maxY"/>
      <tileMinXAttribute name="minX"/>
      <tileMinYAttribute name="minY"/>
    </spatialTable>
</mapping>
```

The first element `<spatialExtension>` specifies which spatial extension the module should use. "universal" means that there is no spatial db extension at all, meaning the tile grid is not stored as a geometry, using simple double values instead.

This xml fragment describes 3 tables, first we need a master table where information for each pyramid level is saved. Second and third, the attribute mappings for storing image data, envelopes and tile names are specified. To keep this tutorial simple, we will not further discuss these xml elements. After creating the sql tables things will become clear.

3. Create the configuration xml `osm.postgis.xml` for the map (osm for "open street map")

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE ImageMosaicJDBCConfig [
  <!ENTITY mapping PUBLIC "mapping"  "mapping.postgis.xml.inc">
  <!ENTITY connect PUBLIC "connect"  "connect.postgis.xml.inc">]>
<config version="1.0">
  <coverageName name="osm"/>
  <coordsys name="EPSG:4326"/>
  <!-- interpolation 1 = nearest neighbour, 2 = bilinear, 3 = bicubic -->
  <scaleop  interpolation="1"/>
  <verify cardinality="false"/>
  &mapping;
  &connect;
</config>
```

This is the final xml configuration file, including our mapping and connect xml fragment. The coverage name is "osm", CRS is EPSG:4326. `<verify cardinality="false">` means no check if the number of tiles equals the number of rectangles stored in the db. (could be time consuming in case of large tile sets).

This configuration is the hard stuff, now, life becomes easier :-)

21.13.6 Using the java ddl generation utility

The full documentation is here: http://docs.geotools.org/latest/userguide/library/coverage/jdbc/ddl.html

To create the proper sql tables, we can use the java ddl generation utility. This utility is included in the `gt-imagemosaic-jdbc-version.jar`. Assure that this jar file is in your `WEB-INF/lib` directory of your GeoServer installation.

Change to your `working` directory and do a first test:

```
java -jar <your_geoserver_install_dir>/webapps/geoserver/WEB-INF/lib/gt-imagemosaic-jdbc-{version}.ja
```

The reply should be:

```
Missing cmd import | ddl
```

Create a subdirectory `sqlscripts` in your `working` directory. Within the `working` directory, execute:

```
java -jar <your_geoserver_install_dir>/webapps/geoserver/WEB-INF/lib/gt-imagemosaic-jdbc-{version}.ja
```

Explanation of parameters

parameter	description
ddl	create ddl statements
-config	the file name of our `osm.postgis.xml` file
-pyramids	number of pyramids we want
-statementDelim	The SQL statement delimiter to use
-srs	The db spatial reference identifier when using a spatial extension
-targetDir	output directory for the scripts
-spatialTNPrefix	A prefix for tablenames to be created.

In the directory `working/sqlscripts` you will find the following files after execution:

`createmeta.sql` `dropmeta.sql` `add_osm.sql` `remove_osm.sql`

Note: *IMPORTANT:*

Look into the files `createmeta.sql` and `add_osm.sql` and compare them with the content of `mapping.postgis.xml.inc`. If you understand this relationship, you understand the mapping.

The generated scripts are only templates, it is up to you to modify them for better performance or other reasons. But do not break the relationship to the xml mapping fragment.

21.13.7 Executing the DDL scripts

For user "postgres", databae "gis", execute in the following order:

```
psql -U postgres -d gis  -f createmeta.sql
psql -U postgres -d gis  -f add_osm.sql
```

To clean your database, you can execute `remove_osm.sql` and `dropmeta.sql` after finishing the tutorial.

21.13.8 Importing the image data

The full documentation is here: http://docs.geotools.org/latest/userguide/library/coverage/jdbc/ddl.html

First, the jdbc jar file has to be in the `lib/ext` directory of your java runtime. In my case I had to copy `postgresql-8.1-407.jdbc3.jar`.

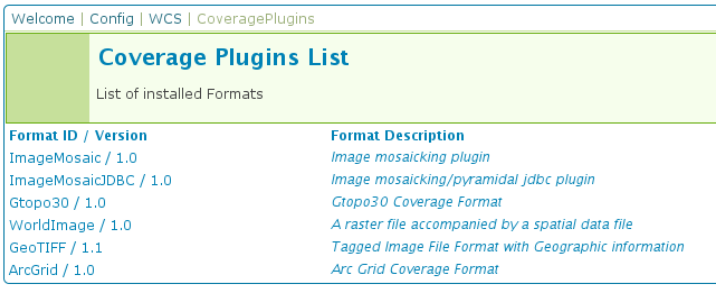
Change to the `working` directory and execute:

```
java -jar <your_geoserver_install_dir>/webapps/geoserver/WEB-INF/lib/gt-imagemosaic-jdbc-{version}.ja
```

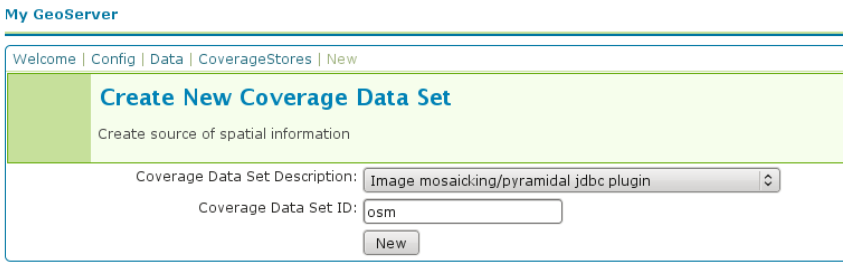
This statement imports your tiles including all pyramids into your database.

21.13.9 Configuring GeoServer

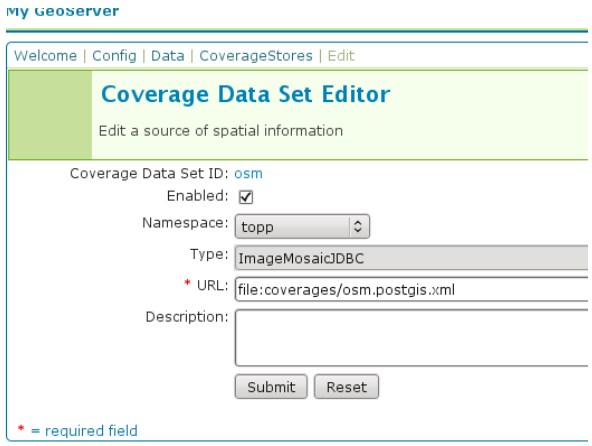
Start GeoServer and log in.Under *Config* → *WCS* → *CoveragePlugins* you should see



If there is no line starting with "ImageMosaicJDBC", the `gt-imagemosiac-jdbc-version.jar` file is not in your `WEB-INF/lib` folder. Go to *Config→Data→CoverageStores→New* and fill in the formular



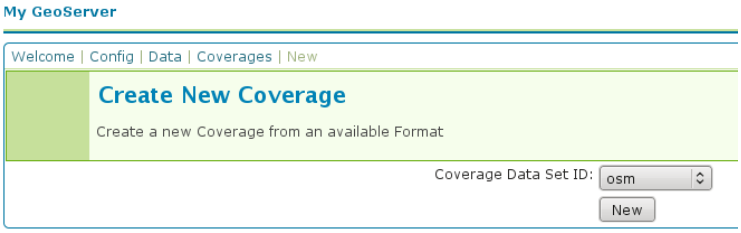
Press *New* and fill in the formular



Press *Submit*.

Press *Apply*, then *Save* to save your changes.

Next select *Config→Data→Coverages→New* and select "osm".



Press *New* and you will enter the Coverage Editor. Press *Submit*, *Apply* and *Save*.

Under *Welcome→Demo→Map Preview* you will find a new layer "topp:osm". Select it and see the results

If you think the image is stretched, you are right. The reason is that the original image is georeferenced with EPSG:900913, but there is no support for this CRS in postigs (at the time of this writing). So I used EPSG:4326. For the purpose of this tutorial, this is ok.

21.13.10 Conclusion

There are a lot of other configuration possibilities for specific databases. This tutorial shows a quick cookbook to demonstrate some of the features of this module. Follow the links to the full documentation to dig deeper, especially if you are concerned about performance and database design.

If there is something which is missing, proposals are welcome.

21.14 Using the GeoTools feature-pregeneralized module

Warning: The screenshots on this tutorial have not yet been updated for the 2.0.x user interface. But most all the rest of the information should be valid, and the user interface is roughly the same, but a bit more easy to use.

21.14.1 Introduction

This tutorial shows how to use the geotools feature-pregeneralized module in GeoServer. The feature-pregeneralized module is used to improve performance and lower memory usage and IO traffic.

Note: Vector generalization reduces the number of vertices of a geometry for a given purpose. It makes no sense drawing a polygon with 500000 vertices on a screen. A much smaller number of vertices is enough to draw a topological correct picture of the polygon.

This module needs features with already generalized geometries, selecting the best fit geometry on demand.

The full documentation is available here:http://docs.geotools.org/latest/userguide/library/data/pregeneralized.html

This tutorial will show two possible scenarios, explaining step by step what to do for using this module in GeoServer.

21.14.2 Getting Started

First, find the location of the GEOSERVER_DATA_DIR. This info is contained in the log file when starting GeoServer.:

```
---------------------------------
- GEOSERVER_DATA_DIR: /home/mcr/geoserver-1.7.x/1.7.x/data/release
---------------------------------
```

Within this directory, we have to place the shape files. There is already a sub directory `data` which will be used. Within this sub directory, create a directory `streams`.

Within *GEOSERVER_DATA_DIR*`/data/streams` create another sub directory called `0`. (0 meaning "no generalized geometries").

This tutorial is based on on a shape file, which you can download from here `Streams`. Unzip this file into *GEOSERVER_DATA_DIR*`/data/streams/0`.

Look for the `WEB-INF/lib/` directory of your GeoServer installation. There must be a file called `gt-feature-pregeneralized-`*version*`-jar`. This jar file includes a tool for generalizing shape files. Open a cmd line and execute the following:

```
cd <GEOSERVER_DATA_DIR>/data/streams/0
java -jar <GEOSERVER_INSTALLATION>/WEB-INF/lib/gt-feature-pregeneralized-{version}.jar generalize 0/s
```

You should see the following output:

```
Shape file            0/streams.shp
Target directory      .
Distances             5,10,20,50
% |################################|
```

Now there are four additional directories `5.0` , `10.0` , `20.0` , `50.0` . Look at the size of files with the extension `shp` within these directories, increasing the generalization distance reduces the file size.

Note: The generalized geometries can be stored in additional properties of a feature or the features can be duplicated. Mixed variations are also possible. Since we are working with shape files we have to duplicate the features.

There are two possibilities how we can deploy our generalized shape files.

1. Deploy hidden (not visible to the user)
2. Deploy each generalized shape file as a separate GeoServer feature

21.14.3 Hidden Deployment

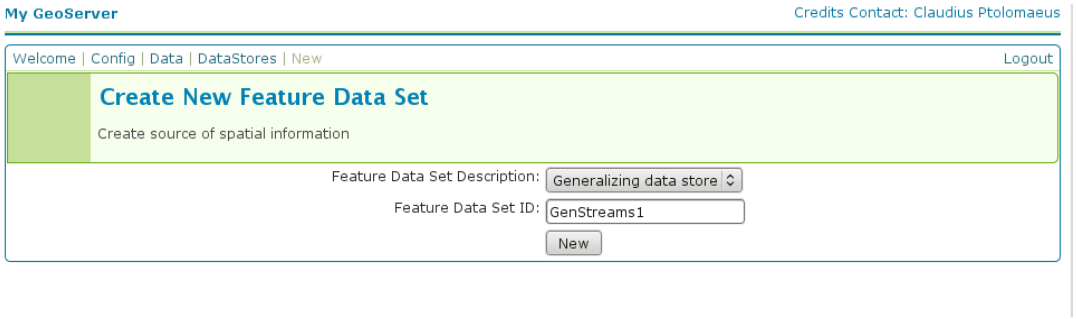
First we need a XML config file

```
<?xml version="1.0" encoding="UTF-8"?>
<GeneralizationInfos version="1.0">
      <GeneralizationInfo dataSourceName="file:data/streams/0/streams.shp"  featureName="GenStreams"
            <Generalization dataSourceName="file:data/streams/5.0/streams.shp"  distance="5" featu
            <Generalization dataSourceName="file:data/streams/10.0/streams.shp"  distance="10" feat
            <Generalization dataSourceName="file:data/streams/20.0/streams.shp"  distance="20" feat
            <Generalization dataSourceName="file:data/streams/50.0/streams.shp"  distance="50" feat
      </GeneralizationInfo>
</GeneralizationInfos>
```

Save this file as `geninfo_shapefile.xml` into *GEOSERVER_DATA_DIR*`/data/streams`.

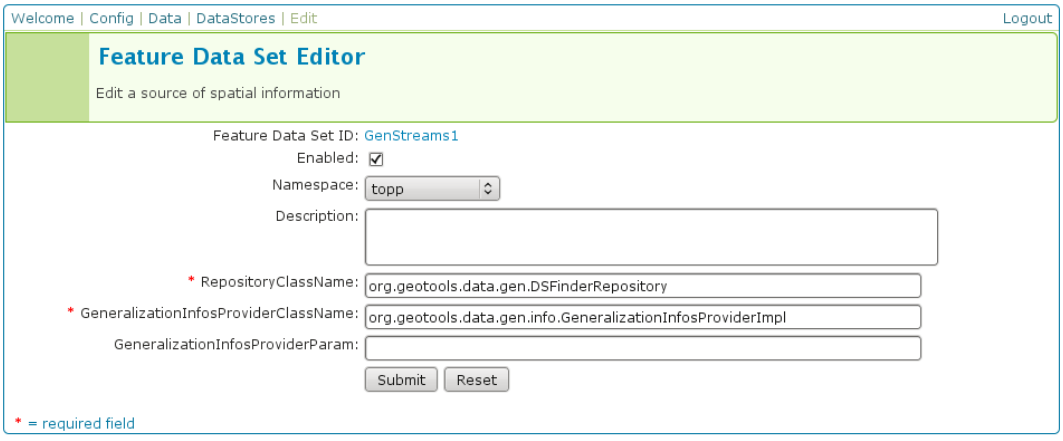
Note: The **dataSourceName** attribute in the XML config is not interpreted as a name, it could be the URL for a shape file or for a property file containing properties for data store creation (e. g. jdbc connect parameters). Remember, this is a hidden deployment and no names are needed. The only *official* name is the value of the attribute **featureName** in the **GeneralizationInfo** Element.

Start GeoServer and go to *Config→Data→DataStores→New* and fill in the form



Press *Submit*.

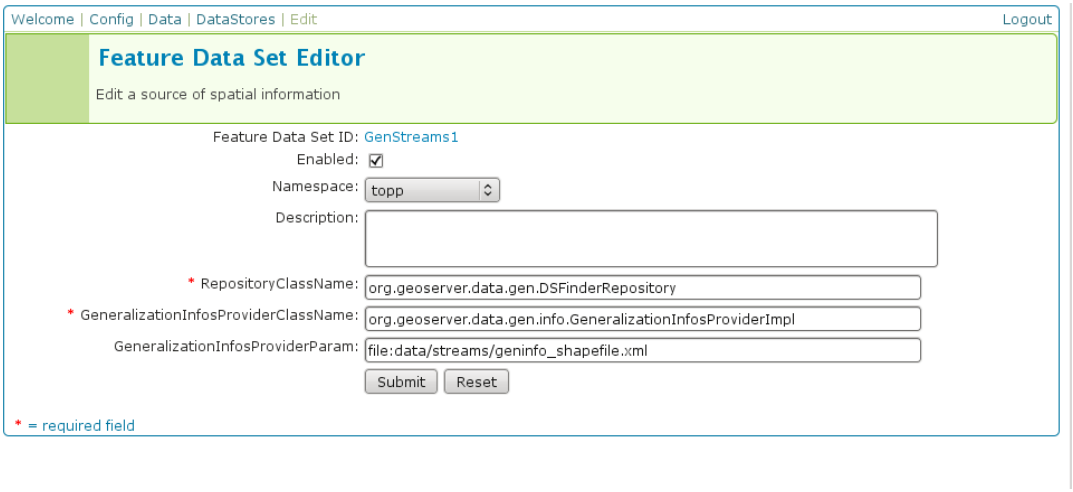
The next form you see is



Note: **RepositoryClassName** and **GeneralizationInfosProviderClassName** have default values which suit for GeoTools, not for GeoServer. Change **GeoTools** to **GeoServer** in the package names to instantiate the correct objects for GeoServer. **GeneralizationInfosProviderParam** could be an URL or a datastore from

the Geoserver catalog. A datastore is referenced by using *workspacename:datastorename*. This makes sense if you have your own implementation for the **GeneralizationInfosProvider** interface and this implementation reads the infos from a database.

The configuration should look like this

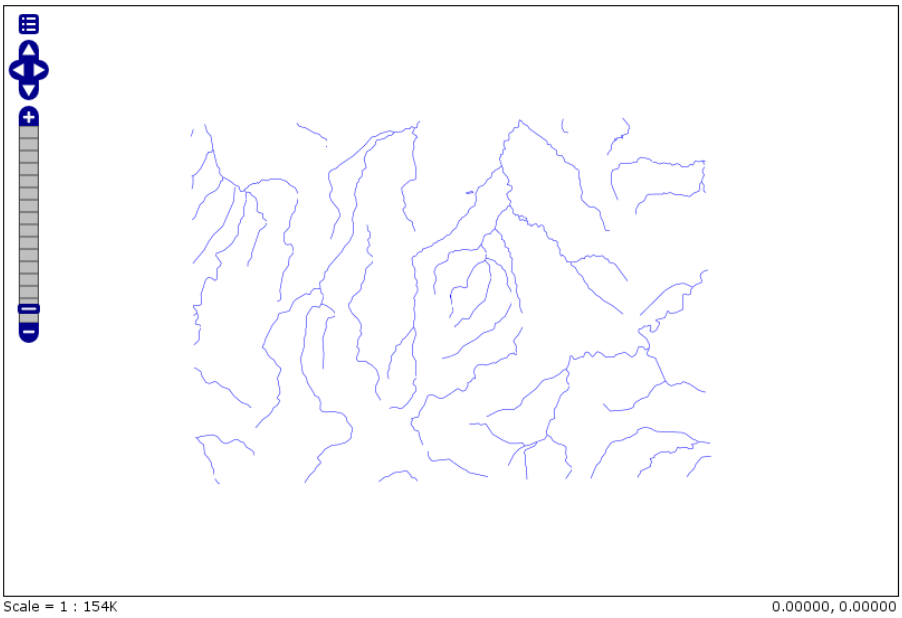


Press *Submit*, afterward a form for the feature type opens.

Alter the **Style** to *line*, **SRS** is *26713* and press the *Generate* button labeled by **Bounding Box**.

Afterward, press *Submit*, *Apply* and *Save*.

Examine the result by pressing "**My GeoServer**, **Demo** and **Map Preview**. In this list there must be an entry **topp:GenStreams**. Press it and you will see

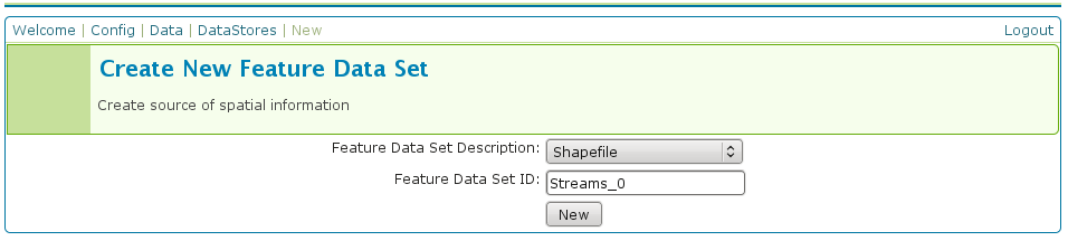


Now start zooming in and out and look at the log file of GeoServer. If the deployment is correct you should see something like this:

```
May 20, 2009 4:53:05 PM org.geotools.data.gen.PreGeneralizedFeatureSource logDistanceInfo
INFO: Using generalizsation: file:data/streams/20.0/streams.shp streams the_geom 20.0
May 20, 2009 4:53:41 PM org.geotools.data.gen.PreGeneralizedFeatureSource logDistanceInfo
INFO: Using generalizsation: file:data/streams/5.0/streams.shp streams the_geom 5.0
May 20, 2009 4:54:08 PM org.geotools.data.gen.PreGeneralizedFeatureSource logDistanceInfo
INFO: Using generalizsation: file:data/streams/5.0/streams.shp streams the_geom 5.0
May 20, 2009 4:54:09 PM org.geotools.data.gen.PreGeneralizedFeatureSource logDistanceInfo
INFO: Using generalizsation: file:data/streams/20.0/streams.shp streams the_geom 20.0
```

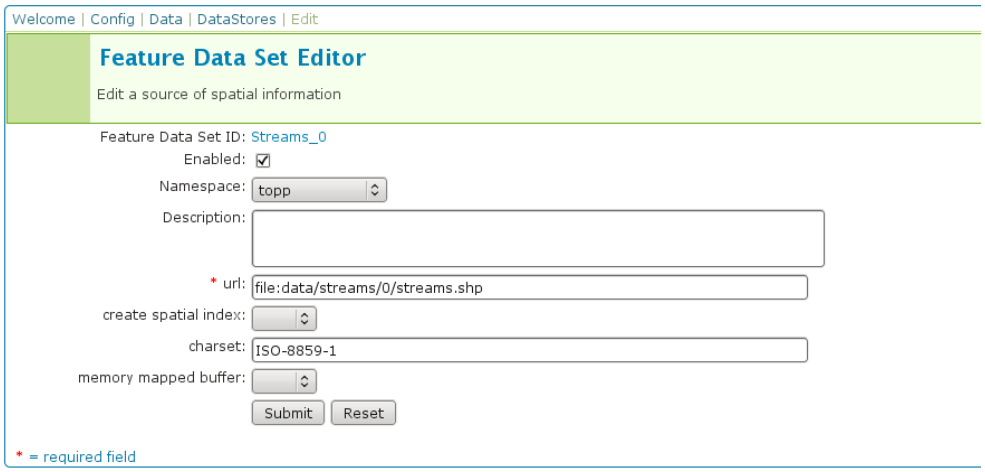
21.14.4 Public Deployment

First we have to configure all our shape files



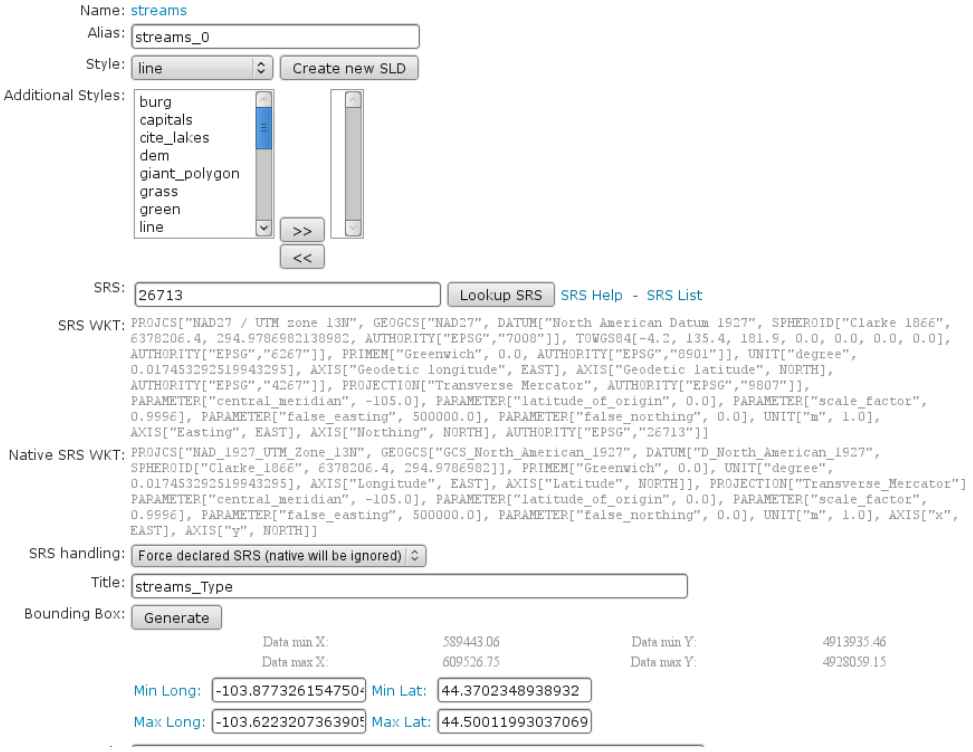
The **Feature Data Set ID** for the other shape files is

1. Streams_5
2. Streams_10
3. Streams_20
4. Streams_50



The **URL** needed for the other shape files

1. `file:data/streams/5.0/streams.shp`
2. `file:data/streams/10.0/streams.shp`
3. `file:data/streams/20.0/streams.shp`
4. `file:data/streams/50.0/streams.shp`



Each feature needs an **Alias**, here it is *streams_0*. For the other shape files use

1. streams_5
2. streams_10
3. streams_20

4. streams_50

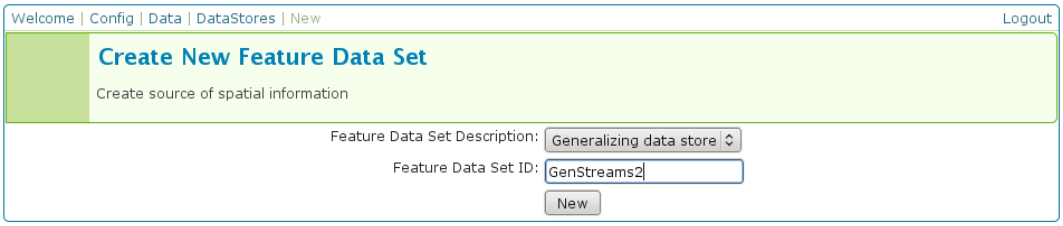
Check the result by pressing *My GeoServer*, *Demo* and *Map Preview*. You should see your additional layers.

No we need another XML configuration file

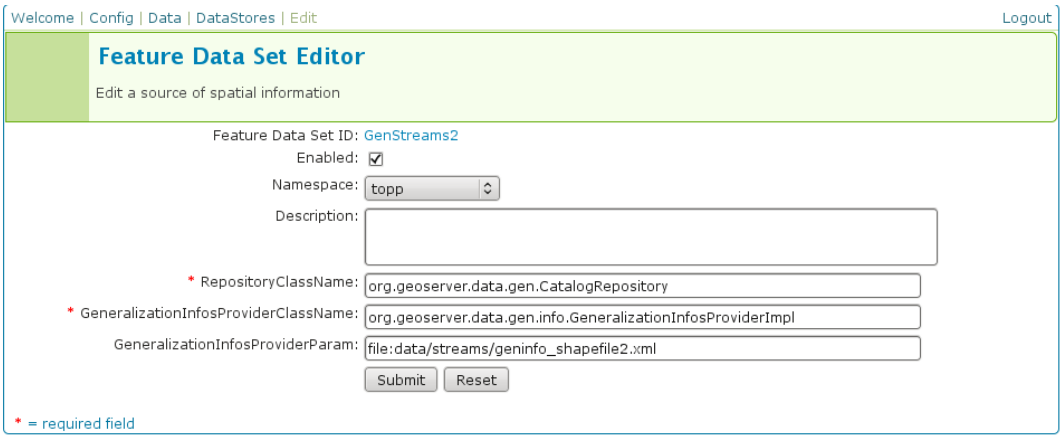
```
<?xml version="1.0" encoding="UTF-8"?>
<GeneralizationInfos version="1.0">
      <GeneralizationInfo dataSourceNameSpace="topp" dataSourceName="Streams_0"  featureName="GenStre
              <Generalization dataSourceNameSpace="topp" dataSourceName="Streams_5"  distance="5" fea
              <Generalization dataSourceNameSpace="topp" dataSourceName="Streams_10"  distance="10" f
              <Generalization dataSourceNameSpace="topp" dataSourceName="Streams_20"  distance="20" f
              <Generalization dataSourceNameSpace="topp" dataSourceName="Streams_50"  distance="50" f
      </GeneralizationInfo>
</GeneralizationInfos>
```

Save this file as `geninfo_shapefile2.xml` into *GEOSERVER_DATA_DIR*`/data/streams`.

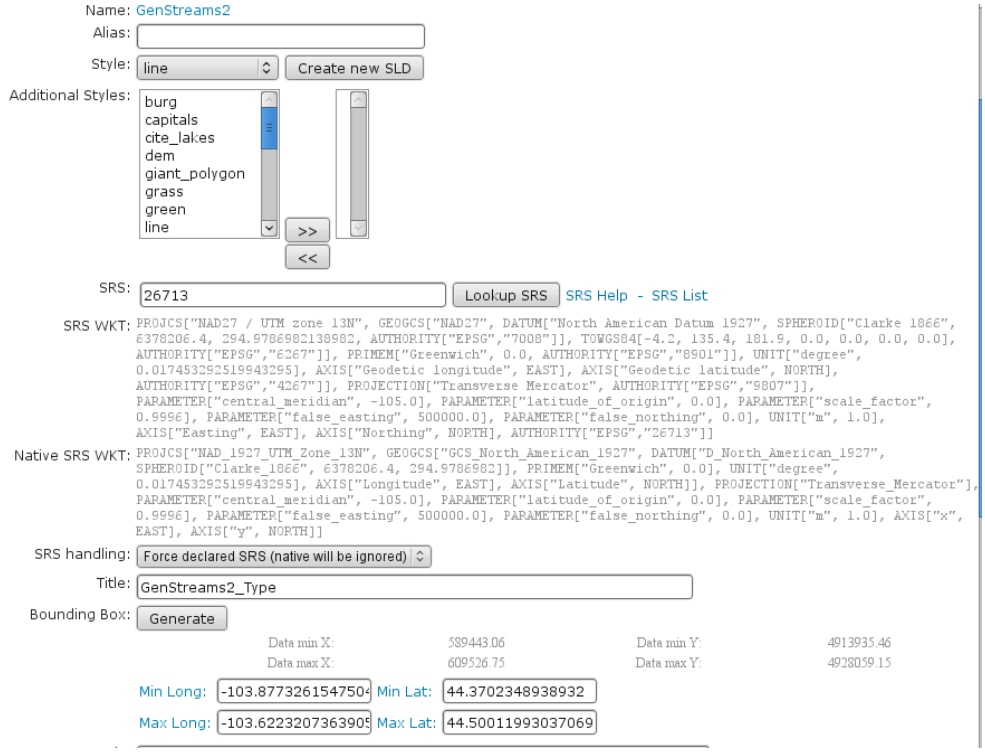
Create the pregeneralized datastore



Now we use the **CatalogRepository** class to find our needed data stores



Last step



In the *Map Preview* you should find **topp:GenStreams2** and all other generalizations. Test in the same manner we discussed in the hidden deployment and you should see something like this in the GeoServer log:

```
May 20, 2009 6:11:06 PM org.geotools.data.gen.PreGeneralizedFeatureSource logDistanceInfo
INFO: Using generalizsation: Streams_20 streams the_geom 20.0
May 20, 2009 6:11:08 PM org.geotools.data.gen.PreGeneralizedFeatureSource logDistanceInfo
INFO: Using generalizsation: Streams_10 streams the_geom 10.0
May 20, 2009 6:11:12 PM org.geotools.data.gen.PreGeneralizedFeatureSource logDistanceInfo
INFO: Using generalizsation: Streams_10 streams the_geom 10.0
```

21.14.5 Conclusion

This is only a very simple example using shape files. The plugin architecture allows you to get your data and generalizations from anywhere. The used dataset is a very small one, so you will not feel a big difference in response time. Having big geometries (in the sense of many vertices) and creating maps with some different layers will show the difference.

21.15 Setting up a JNDI connection pool with Tomcat

> **Warning:** The screenshots on this tutorial have not yet been updated for the 2.0.x user interface. But most all the rest of the information should be valid, and the user interface is roughly the same.

This tutorial walks the reader through the procedures necessary to setup a Oracle JNDI connection pool in Tomcat 6 and how to retrieve it from GeoServer

21.15.1 Tomcat setup

In order to setup a connection pool Tomcat needs a JDBC driver and the necessary pool configurations.

First off, you need to find the JDBC driver for your database. Most often it is distributed on the web site of your DBMS provider, or available in the installed version of your database. For example, a Oracle XE install on a Linux system provides the driver at `/usr/lib/oracle/xe/app/oracle/product/10.2.0/server/jdbc/lib/ojdbc14.jar`, and that file needs to be copied into Tomcat shared libs directory, *TOMCAT_HOME*`/lib`

Once that is done, the Tomcat configuration file *TOMCAT_HOME*`/conf/context.xml` needs to be edited in order to setup the connection pool. In the case of a local Oracle XE the setup might look like:

```
<Context>
   ...
   <Resource name="jdbc/oralocal" auth="Container" type="javax.sql.DataSource"
             url="jdbc:oracle:thin:@localhost:1521:xe"
             driverClassName="oracle.jdbc.driver.OracleDriver"
             username="dbuser" password="dbpasswd"
             maxActive="20" maxIdle="3" maxWait="10000"
             poolPreparedStatements="true"
             maxOpenPreparedStatements="100"
             validationQuery="SELECT SYSDATE FROM DUAL" />
</Context>
```

The example sets up a connection pool connecting to the local Oracle XE instance. The pool configuration shows is quite full fledged:

- at most 20 active connections (max number of connection that will ever be used in parallel)
- at most 3 connections kept in the pool unused
- prepared statement pooling (very important for good performance)
- at most 100 prepared statements in the pool
- a validation query that double checks the connection is still alive before actually using it (this is not necessary if there is guarantee the connections will never drop, either due to the server forcefully closing them, or to network/maintenance issues).

Warning: Modify following settings only if you really know what you are doing. Using too low values for `removedAbandonedTimeout` and `minEvictableIdleTimeMillis` may result in connection failures, if so try to setup `logAbandoned` to `true` and check your `catalina.out` log file.

Other parameters to setup connection pool:

- timeBetweenEvictionRunsMillis (default -1) The number of milliseconds to sleep between runs of the idle object evictor thread. When non-positive, no idle object evictor thread will be run.
- numTestsPerEvictionRun (default 3) The number of objects to examine during each run of the idle object evictor thread (if any).
- minEvictableIdleTimeMillis (default 1000 * 60 * 30) The minimum amount of time an object may sit idle in the pool before it is eligable for eviction by the idle object evictor (if any).
- removeAbandoned (default false) Flag to remove abandoned connections if they exceed the removeAbandonedTimout. If set to true a connection is considered abandoned and eligible for removal if it has been idle longer than the removeAbandonedTimeout. Setting this to true can recover db connections from poorly written applications which fail to close a connection.

- removeAbandonedTimeout (default 300) Timeout in seconds before an abandoned connection can be removed.
- logAbandoned (default false) Flag to log stack traces for application code which abandoned a Statement or Connection.

For more information about the possible parameters and their values refer to the DBCP documentation.

21.15.2 GeoServer setup

To allow a web application reference to a JNDI resource its `web.xml` file must be modified so that the reference is explicit. Following the above example, we have to modify *TOMCAT_HOME*`/webapps/geoserver/WEB-INF/web.xml` and add at its very end the following declaration:

```
<web-app>
  ...
  <resource-ref>
    <description>Oracle Datasource</description>
    <res-ref-name>jdbc/oralocal</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
  </resource-ref>
</web-app>
```

Once that is done, it is possible to login into the GeoServer web administration interface and configure the datastore.

First, choose the *Oracle (JNDI)* datastore and give it a name:

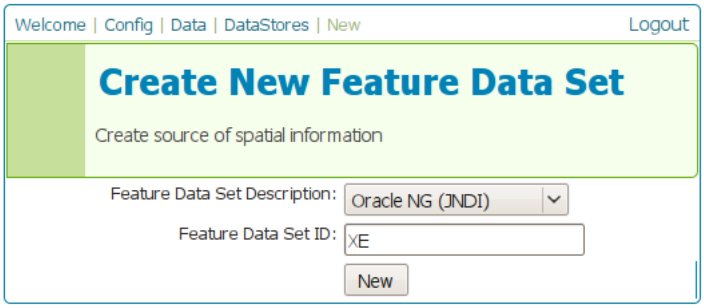


Figure 21.40: *Choosing a JNDI enabled datastore*

Then, configure the connection parameters so that the JNDI path matches the one specified in the Tomcat configuration:

When you are doing this, make sure the *schema* is properly setup, or the datastore will list all the tables it can find in the schema it can access. In the case of Oracle the schema is usually the user name, upper cased.

Once the datastore is accepted the GeoServer usage proceeds as normal.

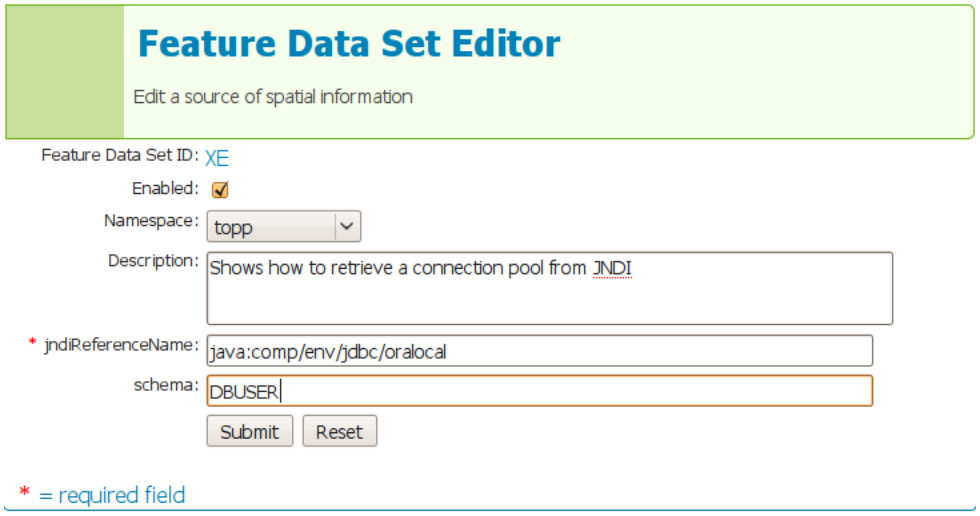


Figure 21.41: *Configuring the JNDI connection*

CHAPTER 22

Community

This section is devoted to GeoServer community modules. Community modules are considered "pending" in that they are not officially part of the GeoServer releases. They are however built along with the nightly builds, so you can download and play with them.

Warning: Community modules are generally considered experimental in nature and are often under constant development. For that reason documentation in this section should not be considered solid or final and will be subject to change.

22.1 Key authentication module

The `authkey` module for GeoServer allows for a very simple authentication protocol designed for OGC clients that cannot handle any kind of security protocol, not even the HTTP basic authentication.

For these clients the module allows a minimal form of authentication by appending a unique key in the URL that is used as the sole authentication token. Obviously this approach is open to security token sniffing and must always be associated with the use of HTTPS connections.

A sample authenticated request looks like:

```
http://localhost:8080/geoserver/topp/wms?service=WMS&version=1.3.0&request=GetCapabilities&authkey=ef
```

Where `authkey=ef18d7e7-963b-470f-9230-c7f9de166888` is associated to a specific user (more on this later). The capabilities document contains backlinks to the server itself, linking to the URLs that can be used to perform GetMap, GetFeatureInfo and so on. When the `authkey` parameter is provided the backlinks will contain the authentication key as well, allowing any compliant WMS client to access secured resources.

22.1.1 Limitations

The `authkey` module is meant to be used with OGC services. It won't work properly against the administration GUI, nor RESTConfig.

22.1.2 Key providers

Key providers are responsible for mapping the authentication keys to a user. The authentication key itself is a UUID (Universal unique Identifier). A key provider needs a user/group service and it is responsible for synchronizing the authentication keys with the users contained in this service.

Key provider using user properties

This key provider uses a user property `UUID` to map the authentication key to the user. User properties are stored in the user/group service. Synchronizing is simple since the logic has to search for users not having the property `UUID` and add it. The property value is a generated UUID.

UUID=b52d2068-0a9b-45d7-aacc-144d16322018

If the user/group service is read only, the property has to be added from outside, no synchronizing is possible.

Key provider using a property file

This key provider uses a property file named `authkeys.properties`. The default user/group service is named `default`. The `authkeys.properties` for this service would be located at

`$GEOSERVER_DATA_DIR/security/usergroup/default/authkeys.propeties`

A sample file looks as follows:

```
# Format is authkey=username
b52d2068-0a9b-45d7-aacc-144d16322018=admin
1825efd3-20e1-4c18-9648-62c97d3ee5cb=sf
ef18d7e7-963b-470f-9230-c7f9de166888=topp
```

This key provider also works for read only user/group services. Synchronizing adds new users not having an entry in this file and removes entries for users deleted in the user/group service.

22.1.3 Configuration

Configuration can be done using the administrator GUI. There is a new type of authentication filter named **authkey** offering the following options.

1. URL parameter name. This the name of URL parameter used in client HTTP requests. Default is `authkey`.
2. Key Provider. GeoSever offers the providers described above.
3. User/group service to be used.

After configuring the filter it is necessary to put this filter on the authentication filter chain(s).

Note: The administrator GUI for this filter has button **Synchronize**. Clicking on this button saves the current configuration and triggers a synchronize. If users are added/removed from the backing user/group service, the synchronize logic should be triggered.

22.1.4 Provider pluggability

With some Java programming it is possible to programmatically create and register a new key to user name mapper that works under a different logic. For example, you could have daily tokens, token generators and the like.

In order to provide your custom mapper you have to implement the `org.geoserver.securityAuthenticationKeyMapper` interface and then register said bean in the Spring application context. Alternatively it is possible to subclass from

`org.geoserver.security.AbstractAuthenticationKeyMapper`. A mapper (key provider) has to implement

```
/**
 *
 * Maps a unique authentication key to a user name. Since user names are
 * unique within a {@link GeoServerUserGroupService} an individual mapper
 * is needed for each service offering this feature.
 *
 * @author Andrea Aime - GeoSolution
 */
public interface AuthenticationKeyMapper extends BeanNameAware {

    /**
     * Maps the key provided in the request to the {@link GeoServerUser} object
     * of the corresponding user, or returns null
     * if no corresponding user is found
     *
     * Returns <code>null</code> if the user is disabled
     *
     * @param key
     * @return
     */
    GeoServerUser getUser(String key) throws IOException;

    /**
     * Assures that each user in the corresponding {@link GeoServerUserGroupService} has
     * an authentication key.
     *
     * returns the number of added authentication keys
     *
     * @throws IOException
     */
    int synchronize() throws IOException;

    /**
     * Returns <code>true</code> it the mapper can deal with read only u
     * user/group services
     *
     * @return
     */
    boolean supportsReadOnlyUserGroupService();

    String getBeanName();

    void setUserGroupServiceName(String serviceName);
    String getUserGroupServiceName();

    public GeoServerSecurityManager getSecurityManager();
    public void setSecurityManager(GeoServerSecurityManager securityManager);

}
```

The mapper would have to be registered in the Spring application context in a `applicationContext.xml` file in the root of your jar. Example for an implementation named `com.mycompany.security.SuperpowersMapper`:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/spring-beans.dt
```

```
<beans>
  <bean id="superpowersMapper" class="com.mycompany.security.SuperpowersMapper"/>
</beans>
```

At this point you can drop the `authkey` jar along with your custom mapper jar and use it in the administrator GUI of the authentication key filter.

22.2 DXF OutputFormat for WFS

The DXF OutputFormat for WFS adds the support for two additional output formats for WFS GetFeature requests. The new formats, DXF and DXF-ZIP are associated to the "application/dxf" and "application/zip" mime type, respectively. They produce a standard DXF file or a DXF file compressed in zip format.

DXF is a CAD interchange format, useful to import data in several CAD systems. Being a textual format it can be easily compressed to a much smaller version, so the need for a DXF-ZIP format, for low bandwidth usage.

There have been multiple revisions of the format, so we need to choose a "version" of DXF to write. The extension implements version 14, but can be easily extended (through SPI providers) to write other versions too.

22.2.1 USAGE

Request Example:

```
http://localhost:8080/geoserver/wfs?request=GetFeature&typeName=Polygons&
outputFormat=dxf
```

Output Example (portion):

```
  0
SECTION
  2
HEADER
  9
$ACADVER
  1
AC1014
...
  0
ENDSEC
...
  0
SECTION
  2
TABLES
...
  0
TABLE
  2
LAYER
...
  0
LAYER
  5
```

```
2E
330
2
100
AcDbSymbolTableRecord
100
AcDbLayerTableRecord
  2
POLYGONS
 70
         0
 62
         7
  6
CONTINUOUS
  0
ENDTAB
...
  0
ENDSEC
  0
SECTION
  2
BLOCKS
  ...
  0
ENDSEC
  0
SECTION
  2
ENTITIES
  0
LWPOLYLINE
  5
927C0
330
1F
100
AcDbEntity
  8
POLYGONS
100
AcDbPolyline
 90
         5
 70
         1
 43
0.0
 10
500225.0
 20
500025.0
 10
500225.0
 20
500075.0
 10
```

```
500275.0
 20
500050.0
 10
500275.0
 20
500025.0
 10
500225.0
 20
500025.0
  0
ENDSEC
  0
SECTION
  2
OBJECTS
...
  0
ENDSEC
  0
EOF
```

Each single query is rendered as a layer. Geometries are encoded as entities (if simple enough to be expressed by a single DXF geometry type) or blocks (if complex, such as polygons with holes or collections).

Some options are available to control the output generated. They are described in the following paragraphs.

22.2.2 GET requests format_options

The following format_options are supported:

1. version: (number) creates a DXF in the specified version format (only 14 is currently supported)
2. asblock: (true/false) if true, all geometries are written as blocks and then inserted as entities. If false, simple geometries are directly written as entities.
3. colors: (comma delimited list of numbers): colors to be used for the DXF layers, in sequence. If layers are more than the specified colors, they will be reused many times. A set of default colors is used if the option is not used. Colors are AutoCad color numbers (7=white, etc.).
4. ltypes: (comma delimited list of line type descriptors): line types to be used for the DXF layers, in sequence. If layers are more than the specified line types, they will be reused many times. If not specified, all layers will be given a solid, continuous line type. A descriptor has the following format: <name>!<repeatable pattern>[!<base length>], where <name> is the name assigned to the line type, <base length> (optional) is a real number that tells how long is each part of the line pattern (defaults to 0.125), and <repeatable pattern> is a visual description of the repeatable part of the line pattern, as a sequence of - (solid line),* (dot) and _ (empty space). For example a dash-dot pattern would be expressed as –_*_.
5. layers: (comma delimited list of strings) names to be assigned to the DXF layers. If specified, must contain a name for each requested query. By default a standard name will be assigned to layers.
6. withattributes: (true/false) enables writing an extra layer with attributes from each feature, the layer has a punctual geometry, with a point in the centroid of the original feature

22.2.3 POST options

Unfortunately, it's not currently possibile to use format_options in POST requests. The only thing we chose to implement is the layers options, via the handle attribute of Query attributes. So, if specified, the layer of a Query will be named as its handle attribute. The handle attribute of the GetFeature tag can also be used to override the name of the file produced.

22.3 DDS/BIL(World Wind Data Formats) Extension

This output module allows GeoServer to output imagery and terrain in formats understood by NASA World Wind. The mime-types supported are:

1. Direct Draw Surface (DDS) - image/dds. This format allows efficient loading of textures to the GPU and takes the task off the WorldWind client CPU in converting downloaded PNG, JPEG or TIFF tiles. The DDS compression is done using DXT3 with help from the worldwind library on server side.
2. Binary Interleaved by Line(BIL) - image/bil. This is actually a very simple raw binary format produced using the RAW Image Writer. The supplied GridCoverage2D undergoes appropriate subsampling, reprojection and bit-depth conversion. The output can be requested as 16bit Int or 32bit Float.

22.3.1 Installing the DDS/BIL extension

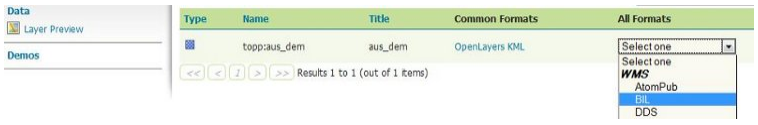
1. Download the DDS/BIL extension from the nightly GeoServer community module builds. A prebuilt version for Geoserver 2.0.x can be found on Jira - GEOS-3586.

   Warning: Make sure to match the version of the extension to the version of the GeoServer instance!

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

22.3.2 Checking if the extension is enabled

Once the extension is installed, the provided mime-types should appear in the layer preview dropbox as shown:



The mime-types will also be listed in the `GetCapabilities` document:

```
<Format>image/bil</Format>
<Format>image/dds</Format>
```

22.3.3 Configuring World Wind to access Imagery/Terrain from GeoServer

Please refer to the WorldWind Forums for instructions on how to setup World Wind to work with layers published via GeoServer. For image layers(DDS) the user need to create a WMSTiledImageLayer either via XML configuration or programmatically. For terrain layers (BIL) the equivalent class is WMSBasicElevationModel.

22.4 NetCDF

22.4.1 Adding a NetCDF data store

Figure 22.1: *NetCDF in the list of raster data stores*

22.4.2 Configuring a NetCDF data store

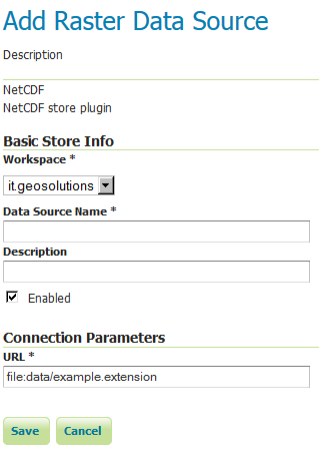


Figure 22.2: *Configuring a NetCDF data store*

Option	Description
`Workspace`	
`Data Source Name`	
`Description`	
`Enabled`	
`URL`	

22.5 GeoServer Printing Module

The `printing` module for GeoServer allows easy hosting of the Mapfish printing service within a GeoServer instance. The Mapfish printing module provides an HTTP API for printing that is useful within JavaScript mapping applications. User interface components for interacting with the print service are available from the Mapfish and GeoExt projects.

22.5.1 Installation

The printing module is built nightly and published to the nightly build server. The installation process is similar to other GeoServer plugins:

- Download the file (named like `geoserver-2.0.2-SNAPSHOT-printing-plugin.zip`)

- Extract the contents of the ZIP archive into the `/WEB-INF/lib/` in the GeoServer webapp. For example, if you have installed the GeoServer binary to `/opt/geoserver-2.0.1/`, the printing extension JAR files should be placed in `/opt/geoserver-2.0.1/webapps/geoserver/WEB-INF/lib/`.
- After extracting the extension, restart GeoServer in order for the changes to take effect. All further configuration can be done with GeoServer running.

22.5.2 Verifying Installation

On the first startup after installation, GeoServer should create a print module configuration file in *GEOSERVER_DATA_DIR*`/printing/config.yaml`. Checking for this file's existence is a quick way to verify the module is installed properly. It is safe to edit this file; in fact there is currently no way to modify the print module settings other than by opening this configuration file in a text editor. Details about the configuration file are available from the *Mapfish website <http://www.mapfish.org/doc/print/>*.

If the module is installed and configured properly, then you will also be able to retrieve a list of configured printing parameters from http://localhost:8080/geoserver/pdf/info.json . This service must be working properly for JavaScript clients to use the printing service.

Finally, you can test printing in this `sample page`. You can load it directly to attempt to produce a map from a GeoServer running at http://localhost:8080/geoserver/. If you are running at a different host and port, you can download the file and modify it with your HTML editor of choice to use the proper URL.

Warning: This sample script points at the development version of GeoExt. You can modify it for production use, but if you are going to do so you should also host your own, minified build of GeoExt and OpenLayers. The libraries used in the sample are subject to change without notice, so pages using them may change behavior without warning.

22.5.3 Using the Print Module in Applications

See the print documentation on the GeoExt web site for information about using the print service in web applications.

22.6 Python

The Python extension allows users to extend GeoServer dynamically by writing Python scripts via jython, the Java implementation of Python.

22.6.1 Installing the Python Extension

1. Download the Python extension from the GeoServer download page.

   Warning: Ensure the extension matching the version of the GeoServer installation is downloaded.

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

Verifiying the Installation

To verify the extension has been installed properly start the GeoServer instance and navigate to the data directory. Upo
named `python` will be created.

22.6.2 Python Extension Overview

The python extension provides a number of scripting hooks throughout GeoServer. These scripting hooks correspond to GeoServer "extension points". An extension point in GeoServer is a class or interface that is designed to be implemented and dynamically loaded to provide a specific function. The classic example is a WMS or WFS output format, but GeoServer contains many extension points.

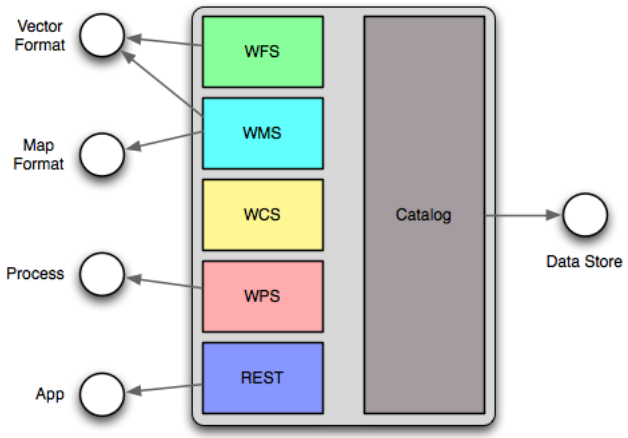


Figure 22.3: Python scripting extension hooks

Implementing a GeoServer extension point in python involves writing scripts and placing them in the appropriate directory under the GeoServer data directory. When the python extension is installed it creates the following directory structure:

```
GEOSERVER_DATA_DIR/
   ...
   python/
      app/
      datastore/
      filter/
      format/
      lib/
      process/
```

Each directory correponds to a GeoServer extension point.

The `app` directory consists of python scripts that are intended to be invoked over http through a wsgi interface.

The `datastore` directory consists of python modules that implement the geotools data store interface. The geotools data store interface is the extension point used to contribute support for vector spatial data formats from shapefiles to postgis.

The `filter` directory consists of modules that implement filter functions. Filter functions are used in WFS queries and in SLD documents.

The `format` directory consists of modules that implement the various output format extension points in GeoServer. This includes WMS GetMap, GetFeatureInfo and WFS GetFeature.

The `lib` directory contains common modules that can be used in implementing the other types of modules. These types of modules are typically utility modules.

The `process` directory consists of modules that implement the geotools process interface. Implements of this extension point are used as processes in the GeoServer WPS.

Continue to *Python Scripting Hooks* for more details.

22.6.3 Python Scripting Hooks

app

The *app* hook provides a way to add scripts that are invoked via http. Scripts are provided with a WSGI environment for execution. A simple hello world example looks like this:

```
def app(environ, start_response):
   start_response('200 OK', [('Content-type','text/plain')])
   return 'Hello world!'
```

The script must define a function named *app* that takes an *environ* which is a dict instance that contains information about the current request, and the executing environment. The *start_response* method starts the response and takes a status code and a set of response headers.

The *app* method returns an iterator that generates the response content, or just a single string representing the entire body.

For more information about WSGI go to http://wsgi.org.

datastore

TODO

filter

The *filter* hook provides filter function implementations to be used in an OGC filter. These filters appear in WFS queries, and in SLD styling rules.

A simple filter function looks like this:

```
from geosever.filter import function
from geoscript.geom import Polygon

@function
def areaGreaterThan(feature, area):
  return feature.geom.area > area
```

The above function returns true or false depending on if the area of a feature is greater than a certain threshold.

format

The *format* hook provides output format implementations for various ows service operations. Examples include png for WMS GetMap, geojson and gml for WFS GetFeature, html and plain text for WMS GetFeatureInfo.

Currently formats fall into two categories. The first are formats that can encode vector data (features). A simple example looks like:

```
from geoserver.format import vector_format

@vector_format('property', 'text/plain')
def write(data, out):
 for feature in data.features:
   out.write("%s=%s\n" % (f.id, '|'.join([str(val) for val in f.values()])))
```

The above function encodes a set of features as a java property file. Given the following feature set:

```
Feature(id="fid.0", geometry="POINT(0 0)", name="zero")
Feature(id="fid.1", geometry="POINT(1 1)", name="one")
Feature(id="fid.2", geometry="POINT(1 1)", name="two")
```

The above function would output:

```
fid.0=POINT(0 0)|one
fid.1=POINT(1 1)|two
fid.2=POINT(2 2)|three
```

Vector formats can be invoked by the following service operations:

- WFS GetFeature (`?outputFormat=property`)
- WMS GetMap (`?format=property`)
- WMS GetFeatureInfo (`?info_format=property`)

A vector format is a python function that is decorated by the `vector_format` decorator. The decorator accepts two arguments. The first is the *name* of the output format. This is the identifier that clients use to request the format. The second parameter is the *mime type* that describes the type of content the format creates.

The second type of output format is one that encodes a complete map. This format can only be used with the WMS GetMap operation.

TODO: example

process

The *process* hook provides process implementations that are invoked by the GeoServer WPS. A simple example looks like:

```
from geoserver import process
from geoscript.geom import Geometry

@process('Buffer', 'Buffer a geometry', args=[('geom', Geometry)],
         result=('The buffered result', Geometry))
def buffer(geom):
   return geom.buffer(10)
```

A process is a function that is decorated by the `process` decorator. The decorator takes the following arguments:

title	The title of the process to displayed to clients
description	The description of the process.
version	The version of the process
args	The arguments the process accepts as a list of tuples
result	The result of a process as a tuple

The `args` parameter is a list of tuples describing the input arguments of the process. Each tuple can contain up to three values. The first value is the name of the parameter and is mandatory. The second value is the type of the parameter and is optional. The third value is a description of the parameter and is optional.

The `result` parameter describes the result of the process and is a tuple containing up to two values. This parameter is optional. The first value is the type of the result and the second value is a description of the result.

22.7 Scripting

The GeoServer scripting extension allows users to extend GeoServer dynamically by writing scripts in languages other than Java.

22.7.1 Installing the Scripting Extension

Note: The various language runtime libraries increase GeoServer's memory footprint, specifically the "PermGen" (Permanent Generation) space. When installing the scripting extension we recommended that you increase PermGen capacity to 256m. This is done with the option `-XX:MaxPermSize=256m`. If installing multiple language extensions this size may need to be increased even further.

Python

Currently, the only scripting language that is distributed as a package for download is Python. This extension is a community extension, in that it is not included with the list of extensions on the standard GeoServer download page. Instead, the community extensions are built each night on the nightly build server.

To access the Python scripting extension:

1. Navigate to the nightly build server.
2. Click the folder that contains the correct branch of GeoServer for your version (for example: for 2.2.2, click on *2.2.x*):
3. Click *community-latest*. This folder contains the most recently built community extensions for the branch.
4. Download the file that contains the string "python". For example: `geoserver-2.2-SNAPSHOT-python-plugin.zip`.
5. Extract the contents of the archive into the `/WEB-INF/lib/` directory of GeoServer. For example, if GeoServer was installed at `/opt/geoserver-2.2.2/`, extract the archive contents in `/opt/geoserver-2.1.0/webapps/geoserver/WEB-INF/lib/`.
6. Restart GeoServer.

Upon a successful install a new directory named `scripts` will be created inside the data directory.

22.7.2 Supported Languages

Support for the following scripting languages is available:

- Python
- JavaScript
- Groovy
- Beanshell
- Ruby

Adding support for additional languages is relatively straight forward. The requirements for adding a new language are:

1. The language has an implementation that runs on the Java virtual machine
2. The language runtime provides a JSR-223 compliant script engine

GeoScript

GeoScript is a project that adds scripting capabilities to the GeoTools library. It can be viewed as bindings for GeoTools in various other languages that are supposed on the JVM. It is the equivalent of the various language bindings that GDAL and OGR provide.

Currently GeoScript is available for the following languages:

- Python
- JavaScript
- Groovy

The associated GeoServer scripting extension for these languages come with GeoScript for that language enabled. This means that when writing scripts one has access to the GeoScript modules and packages like they would any other standard library package.

Those languages that don't have a GeoScript implementation can still implement the same functionality that GeoScript provides but must do it against the GeoTools api directly. The downside being that usually the GeoTools api is much more verbose than the GeoScript equivalent. But the upside is that going straight against the GeoTools api is usually more efficient.

Therefore GeoScript can be viewed purely as a convenience for script writers.

22.7.3 Scripting Extension Overview

The scripting extension provides a number of extension points called "hooks" throughout GeoServer. Each hook provides a way to plug in functionality via a script. See the *Scripting Hooks* section for details on each of the individual scripting hooks.

Scripts are located in the GeoServer data directory under a directory named `scripts`. Under this directory exist a number of other directories, one for each scripting hook:

```
GEOSERVER_DATA_DIR/
  ...
  scripts/
    apps/
    lib/
    wps/
```

The `apps` directory provides an "application" hook allowing for one to provide a script invokable over http.

The `wps` directory provides a Web Processing Service (WPS) process hook to contribute a process invokable via WPS.

The `lib` directory is not a hook but is meant to be a location where common scripts may be placed. For instance this directory may be used as a common location for data structures and utility functions that may be utilized across many different scripts.

Note: How the `lib` directory (or if it is utilized at all) is language specific.

See *Scripting Hooks* for more details.

Creating scripts involves creating a script in one of these hook directories. New scripts are picked up automatically by GeoServer without a need to ever restart the server as is the case with a pure Java GeoServer extension.

22.7.4 Scripting Hooks

This page describes all available scripting hooks. Every hook listed on this page is implemented by all the supported language extensions. However, depending on the language, the interfaces and api used to write a script may differ. Continue reading for more details.

Applications

The "app" hook provides a way to contribute scripts that are intended to be run over http. An app corresponds to a named directory under the `scripts/apps` directory. For example:

```
GEOSERVER_DATA_DIR/
  ...
  scripts/
    apps/
      hello/
```

An app directory must contain a *main* file that contains the "entry point" into the application. Every time the app is invoked via an http request this main file is executed.

The contains of the main file differ depending on the language. The default for all languages is simply that the main file contain a function named "run" that takes two arguments, the http request and response. For example, in beanshell:

```
import org.restlet.data.*;

run(request,response) {
  response.setEntity("Hello World!", MediaType.TEXT_PLAIN);
}
```

As explained above this api can differ depending on the language. For example in Python we have the well defined WSGI specification that gives us a standard interface for Python web development. The equivalent Python script to that above is:

```
def app(environ, start_response):
  start_response('200 OK', [('Content-Type', 'text/plain')])
  return ['Hello World!']
```

For the JavaScript app hook, scripts are expected to export an `app` function that conforms to the JSGI specification (v0.3). The equivalent 'Hello World' app in JavaScript would look like the following (in `/scripts/apps/hello/main.js`):

```
exports.app = function(request) {
  return {
    status: 200,
    headers: {"Content-Type": "text/plain"},
    body: ["Hello World"]
  }
};
```

Applications are http accessible at the path `/script/apps/{app}` where `{app}` is the name of the application. For example assuming a local GeoServer the url for for the application would be:

```
http://localhost:8080/geoserver/script/apps/hello
```

Web Processing Service

The wps hook provides a way to provides a way to contribute scripts runnable as a WPS process. The process is invoked using the standard WPS protocol the same way an existing well-known process would be.

All processes are located under the `scripts/wps` directory. Each process is located in a file named for the process. For example:

```
GEOSERVER_DATA_DIR/
  ...
  scripts/
    wps/
      buffer.bsh
```

A process script must define two things:

1. The process metadata: title, description, inputs, and outputs
2. The process routine itself

The default for languages is to define the metadata as global variables in the script and the process routine as a function named "run". For example, in groovy:

```
import com.vividsolutions.jts.geom.Geometry

title = 'Buffer'
description = 'Buffers a geometry'

inputs = [
  geom: [name: 'geom', title: 'The geometry to buffer', type: Geometry.class],
  distance: [name: 'distance', title: 'The buffer distance', type: Double.class]
]
```

```
outputs = [
  result: [name: 'result', title: 'The buffered geometry',  type: Geometry.class]
]

def run(input) {
  return [result: input.geom.buffer(input.distance)]
}
```

In Python the api is slightly different and makes use of Python decorators:

```
from geoserver.wps import process
from com.vividsolutions.jts.geom import Geometry

@process(
  title='Buffer',
  description='Buffers a geometry',
  inputs={
    'geom': (Geometry, 'The geometry to buffer'),
    'distance':(float,'The buffer distance')
  },
  outputs={
    'result': (Geometry, 'The buffered geometry')
  }
)
def run(geom, distance):
  return geom.buffer(distance);
```

In JavaScript, a script exports a `process` object (see the GeoScript JS API docs for more detail) in order to be exposed as a WPS process. The following is an example of a simple buffer process (saved in `scripts/wps/buffer.js`):

```
var Process = require("geoscript/process").Process;

exports.process = new Process({
  title: "JavaScript Buffer Process",
  description: "Process that buffers a geometry.",
  inputs: {
    geom: {
      type: "Geometry",
      title: "Input Geometry",
      description: "The target geometry."
    },
    distance: {
      type: "Double",
      title: "Buffer Distance",
      description: "The distance by which to buffer the geometry."
    }
  },
  outputs: {
    result: {
      type: "Geometry",
      title: "Result",
      description: "The buffered geometry."
    }
  },
  run: function(inputs) {
    return {result: inputs.geom.buffer(inputs.distance)};
  }
});
```

Once implemented a process is invoked using the standard WPS protocol. For example assuming a local GeoServer the url to execute the process would be:

```
http://localhost:8080/geoserver/wps
  ?service=WPS
  &version=1.0.0
  &request=Execute
  &identifier=XX:buffer
  &datainputs=geom=POINT(0 0)@mimetype=application/wkt;distance=10
```

(Substitue `XX:buffer` for the script name followed by the extension. E.g. `py:buffer` for Python or `js:buffer` for JavaScript.)

22.7.5 Scripting Reference

Python

GeoServer Python API Documentation

Script Hooks

app In Python the app hook is based on WSGI which provides a common interface for Python web application development. This is not a comprehensive introduction to WSGI, that can be found here, but the app script must provide a function named `app` that takes a dictionary containing information about the environment, and a function to start the response.

```
def app(environ, start_response):
  # do stuff here
```

The function must be present in a file named `main.py` in a named *application directory*. Application directories live under the `scripts/apps` directory under the root of the data directory:

```
GEOSERVER_DATA_DIR/
  ...
  scripts/
    apps/
      app1/
        main.py
        ...
      app2/
        main.py
        ...
```

The application is web accessible from the path `/script/apps/{app}` where `{app}` is the name of the application. All requests that start with this path are dispatched to the `app` function in `main.py`.

Hello World Example In this example a simple "Hello World" application is built. First step is to create a directory for the app named `hello`:

```
cd $GEOSERVER_DATA_DIR/scripts/apps
mkdir hello
```

Next step is to create the `main.py` file:

```
cd hello
touch main.py
```

Next the app function is created and stubbed out:

```
def app(environ, start_response):
  pass
```

Within the app function the following things will happen:

1. Report an HTTP status code of 200
2. Declare the content type of the response, in this case "text/plain"
3. Generate the response, in this case the string "Hello World"

Steps 1 and 2 are accomplished by invoking the start_response function:

```
start_response('200 OK', [('Content-Type', 'text/plain')])
```

Step 3 is achieved by returning an array of string content:

```
return ['Hello World']
```

The final completed version:

```
def app(environ, start_response):
  start_response('200 OK', [('Content-Type', 'text/plain')])
  return ['Hello World!']
```

Note: WSGI allows for additional methods of generating responses rather than returning an array. In particular it supports returning an generator for the response content. Consult the WSGI documentation for more details.

---

wps In Python the wps/process interface is much like the other languages, with a few differences. A process is defined with a function named run that is decorated with the geoserver.wps.process decorator:

```
from geoserver.wps import process

@process(...)
def run(...):
   # do something
```

The function is located in a file under the scripts/wps directory under the root of the data directory. A WPS process requires metadata to describe itself to the outside world including:

- A **name** identifying the process
- A short **title** describing the process
- An optionally longer **description** that describes the process in more detail
- A dictionary of **inputs** describing the parameters the process accepts as input
- A dictionary of **outputs** describing the results the process generates as output

In python the name is implicitly derived from the name of the file that contains the process function. The rest of the metadata is passed in as arguments to the process decorator. The title and description are simple strings:

```
@process(title='Short Process Title',
         description='Longer and more detailed process description')
def run():
   pass
```

The `inputs` metadata is a dictionary keyed with strings matching the names of the process inputs. The values of the dictionary are tuples in which the first element is the type of the input and the second value is the description of the input. The keys of the dictionary must match those declared in the process function itself:

```
@process(
  ...
  inputs={'arg1': (<arg1 type>, 'Arg1 description'),
          'arg2': (<arg2 type>, 'Arg2 description')}
)
def run(arg1, arg2):
  pass
```

The `outputs` metadata is the same structure as the inputs dictionary except that for it describes the output arguments of the process:

```
@process(
  ...
  outputs={'result1': (<result1 type>, 'Result1 description'),
           'result2': (<result2 type>, 'Result2 description')}
)
def run(arg1, arg2):
  pass
```

A process must generate and return results matching the `outputs` arguments. For processes that return a single value this is implicitly determined but processes that return multiple values must be explicit by returning a dictionary of the return values:

```
@process(
  ...
  outputs={'result1': (<result1 type>, 'Result1 description'),
           'result2': (<result2 type>, 'Result2 description')}
)
def run(arg1, arg2):
  # do something
  return {
    'result1': ...,
    'result2': ...
  }
```

Buffer Example In this example a simple buffer process is created. First step is to create a file named `buffer.py` in the `scripts/wps` directory:

```
cd $GEOSERVER_DATA_DIR/scripts/wps
touch buffer.py
```

Next the `run` function is created and stubbed out. The function will take two arguments:

1. A geometry object to buffer
2. A floating point value to use as the buffer value/distance

```
def run(geom, distance):
  pass
```

In order for the function to picked up it must first be decorated with the `process` decorator:

```
from geoserver.wps import process

@process(title='Buffer', description='Buffers a geometry')
def run(geom, distance):
  pass
```

Next the process inputs and outputs must be described:

```
from geoscript.geom import Geometry

@process(
   ...,
   inputs={ 'geom': (Geometry, 'The geometry to buffer'),
            'distance': (float,'The buffer distance')},
   outputs={'result': (Geometry, 'The buffered geometry')}
)
def run(geom, distance):
  pass
```

And finally writing the buffer routine which simply just invokes the `buffer` method of the geometry argument:

```
@process(...)
def run(geom, distance):
  return geom.buffer(distance)
```

In this case since the process returns only a single argument it can be returned directly without wrapping it in a dictionary.

The final completed version:

```
from geoserver.wps import process
from geoscript.geom import Geometry

@process(
   title='Buffer',
   description='Buffers a geometry',
   inputs={'geom': (Geometry, 'The geometry to buffer'),
           'distance':(float,'The buffer distance')},
   outputs={'result': (Geometry, 'The buffered geometry')}
)
def run(geom, distance):
  return geom.buffer(distance);
```

GeoScript-PY

As mentioned *previously* GeoScript provides scripting apis for GeoTools in various languages. Naturally the GeoServer Python extension comes with GeoScript Python enabled. In the buffer example above an example of importing a GeoScript class was shown.

The GeoScript Python api is documented here.

API Reference

In much the same way as GeoScript provides a convenient scripting layer on top of GeoTools the Python scripting extension provides a `geoserver` Python module that provides convenient access to some of the GeoServer internals.

The GeoServer Python api is documented *here*.

JavaScript

The GeoServer scripting extension provides a number of scripting *hooks* that allow script authors to take advantage of extension points in GeoServer.

Hooks

The App Hook In JavaScript the app hook is based on JSGI which provides a common interface for JavaScript web application development. The app script must export a function named `app` that accepts a `request` object and returns a `response` object.

```
export.app = function(request) {
  // handle the request and return a response
}
```

The function must be exported from a file named `main.js` in a named *application directory*. Application directories live under the `scripts/apps` directory in the root of the data directory:

```
GEOSERVER_DATA_DIR/
  ...
  scripts/
    apps/
      app1/
        main.js
        ...
      app2/
        main.js
        ...
```

The application is web accessible from the path `/script/apps/{app}` where `{app}` is the name of the application. All requests that start with this path are dispatched to the `app` function in `main.js`.

Hello World Example In this example a simple "Hello World" application is built. The first step is to create a directory for the app named `hello`:

```
cd $GEOSERVER_DATA_DIR/scripts/apps
mkdir hello
```

Next step is to create the `main.js` file:

```
cd hello
touch main.js
```

Within the app function the following things will happen:

1. Report an HTTP status code of 200
2. Declare the content type of the response, in this case "text/plain"
3. Generate the body of response, in this case the string "Hello World"

This is accomplished with the following code:

```
export.app = function(request) {
  return {
    status: 200, // step 1
    headers: {"Content-Type": "text/plain"}, // step 2
    body: ["Hello World"] // step 3
  };
};
```

The body of the response shown above is an array. In general, this can be any object with a `forEach` method. In this way, an app can returned chunked content instead of returning the entire body content at once.

The WPS Hook In GeoScript JS, the `geoscript/process` module provides a `Process` constructor. A process object wraps a function with a title, description, and additional metadata about the inputs and outputs. With the GeoServer scripting extension, when a script exports a process, it is exposed in GeoServer via the WPS interface.

To better understand how to construct a well described process, we'll examine the parts of the previously provided `buffer.js` script:

```
var Process = require("geoscript/process").Process;

exports.process = new Process({
  title: "JavaScript Buffer Process",
  description: "Process that buffers a geometry.",
  inputs: {
    geom: {
      type: "Geometry",
      title: "Input Geometry",
      description: "The target geometry."
    },
    distance: {
      type: "Double",
      title: "Buffer Distance",
      description: "The distance by which to buffer the geometry."
    }
  },
  outputs: {
    result: {
      type: "Geometry",
      title: "Result",
      description: "The buffered geometry."
    }
  },
  run: function(inputs) {
    return {result: inputs.geom.buffer(inputs.distance)};
  }
});
```

When this script is saved in the `$GEOSERVER_DATA_DIR/scripts/wps` directory, it will be available to WPS clients with the identifier `js:buffer`. In general, the process identifier is the name of the script prefixed by the language extension.

First, the `require` function is used to pull in the `Process` constructor from the `geoscript/process` module:

```
var Process = require("geoscript/process").Process;
```

Next, a process is constructed and assigned to the `process` property of the `exports` object. This makes it available to other JavaScript modules that may want to import this process with the `require` function in addition to exposing the process to GeoServer's WPS. The title and description provide WPS clients with human readable information about what the process does.

```
exports.process = new Process({
  title: "JavaScript Buffer Process",
  description: "Process that buffers a geometry.",
```

All the work of a process is handled by the `run` method. Before clients can execute a process, they need to know some detail about what to provide as input and what to expect as output. In general, processes accept multiple inputs and may return multiple outputs. These are described by the process' `inputs` and `outputs` properties.

```
inputs: {
  geom: {
    type: "Geometry",
    title: "Input Geometry",
    description: "The target geometry."
  },
  distance: {
    type: "Double",
    title: "Buffer Distance",
    description: "The distance by which to buffer the geometry."
  }
},
```

The buffer process expects two inputs, named `geom` and `distance`. As with the process itself, each of these inputs has a human readable title and description that will be provided to WPS clients. The `type` property is a shorthand string identifying the data type of the input. See the Process API docs for more detail on supported input and output types.

```
outputs: {
  result: {
    type: "Geometry",
    title: "Result",
    description: "The buffered geometry."
  }
},
```

The buffer process provides a single output identified as `result`. As with each of the inputs, this output is described with `type`, `title`, and `description` properties.

To see what this process metadata looks like to a WPS client, call the WPS DescribeProcess method:

```
http://localhost:8080/geoserver/wps
  ?service=WPS
  &version=1.0.0
  &request=DescribeProcess
  &identifier=js:buffer
```

Finally, the `run` method is provided.

```
  run: function(inputs) {
    return {result: inputs.geom.buffer(inputs.distance)};
  }
});
```

The `run` method takes a single `inputs` argument. This object will have named properties corresponding the the client provided inputs. In this case, the `geom` property is a `Geometry` object from the

`geoscript/geom` module. This geometry has a `buffer` method that is called with the provided distance. See the Geometry API docs for more detail on available geometry properties and methods.

The `run` method returns an object with properties corresponding to the above described outputs - in this case, just a single `result` property.

To see the results of this processs in action, call the WPS Execute method:

```
http://localhost:8080/geoserver/wps
  ?service=WPS
  &version=1.0.0
  &request=Execute
  &identifier=js:buffer
  &datainputs=geom=POINT(0 0)@mimetype=application/wkt;distance=10
```

GeoScript JS

To provide a JavaScript interface for data access and manipulation via GeoTools, the GeoServer scripting extension includes the GeoScript JS library. To best leverage the scripting hooks in GeoServer, read through the GeoScript JS API docs for detail on scripting access to GeoTools functionality with JavaScript.

GeoServer JavaScript Reference

In much the same way as GeoScript JS provides a convenient set of modules for scripting access to GeoTools, the GeoServer scripting extension includes a `geoserver` JavaScript module that allows convenient access to some of the GeoServer internals. See the *GeoServer JavaScript API Documentation* for more detail.

GeoServer JavaScript API Documentation The scripting extension includes a `geoserver/catalog` module that allows scripts to access resources in the GeoServer catalog.

The `catalog` module

```
var catalog = require("geoserver/catalog");
```

Properties

`namespaces`

`Array` A list of namespace objects. Namespaces have `alias` and `uri` properties.

```
catalog.namespaces.forEach(function(namespace) {
  // do something with namespace.alias or namespace.uri
});
```

Methods

`getVectorLayer`(*id*)

Parameters **id** – `String` The fully qualified feature type identifier (e.g. "topp:states")

Returns `geoscript.layer.Layer`

Access a feature type in the catalog as a GeoScript Layer.

```
var states = catalog.getVectorLayer("topp:states");
```

22.8 SpatiaLite

SpatiaLite is the spatial extension of the popular SQLite embedded relational database.

Note: GeoServer does not come built-in with support for SpatiaLite; it must be installed through an extension. Furthermore it requires that additional native libraries be available on the system. Proceed to *Installing the SpatiaLite extension* for installation details.

22.8.1 SpatiaLite version

The GeoServer SpatiaLite extension includes its own versions of SQLite (3.7.2) and SpatiaLite (2.4.0) and therefore these libraries need not be installed on the system in order to use the extension. However this internal version of SpatiaLite is compiled against the PROJ and GEOS libraries so they must be installed on the system in order for the extension to function. See *Native Libraries* for more details.

22.8.2 Supported platforms

This extension is supported for Windows, Linux, and Mac OS X. Both 32-bit and 64-bit platforms are supported. For Mac OS X only Intel based machines are supported (ie. not PowerPC).

22.8.3 Installing the SpatiaLite extension

1. Download the SpatiaLite extension from the nightly GeoServer community module builds.

   Warning: Make sure to match the version of the extension to the version of the GeoServer instance.

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.
3. Ensure the native library dependencies are satisfied.

22.8.4 Native Libraries

The version of SpatiaLite included in the extension is compiled against the GEOS and PROJ libraries so they must be installed on the system. If the libraries are not installed on the system the extension will not functionand remain disabled.

Note: Pre-compiled libraries are available for the following platforms and can be found here.

In general if the libraries are installed in a "default" location then they should be picked up by java with no problem. However some systems will require further configuration that differs based on operating system.

Windows

The DLL's must be copied into the `C:\WINDOWS\system32` directory.

Linux

If the libraries are not installed in a default search location like `/usr/lib` then the `LD_LIBRARY_PATH` environment variable must be set and visible to Java.

Mac OS X

Same as Linux except that the `DYLD_LIBRARY_PATH` environment variable is used.

22.8.5 Adding a SpatiaLite database

Once the extension is properly installed `SpatiaLite` will show up as an option when creating a new data store.

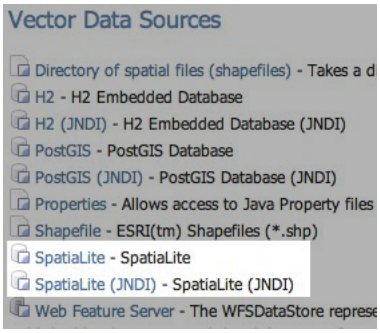


Figure 22.4: *SpatiaLite in the list of vector data sources*

22.8.6 Configuring a SpatiaLite data store

`database`	The name of the database to connect to. See *notes* below.
`schema`	The database schema to access tables from. Optional.
`user`	The name of the user to connect to the database as. Optional.
`password`	The password to use when connecting to the database. Optional, leave blank for no password.
`max connections` `min connections`	Connection pool configuration parameters. See the *Database Connection Pooling* section for details.

The *database* parameter may be specified as an absolute path or a relative one. When specified as a relative path the database will created in the `spatialite` directory, located directly under the root of the GeoServer data directory.

22.9 libjpeg-turbo Map Encoder Extension

This plugin brings in the ability to encode JPEG images as WMS output using the libjpeg-turbo library. Citing its website the libjpeg-turbo library is a derivative of libjpeg that uses SIMD instructions (MMX, SSE2, NEON) to accelerate baseline JPEG compression and decompression on x86, x86-64, and ARM systems. On such systems, libjpeg-turbo is generally 2-4x as fast as the unmodified version of libjpeg, all else being equal. I guess it is pretty clear why we wrote this plugin! Note that the underlying imageio-ext-turbojpeg

Figure 22.5: *Configuring a SpatiaLite data store*

uses TurboJpeg which is a higher level set of API (providing more user-friendly methods like "Compress") built on top of libjpeg-turbo.

Warning: The speedup may vary depending on the target infrastructure.

The module, once installed, simply replace the standard JPEG encoder for GeoServer and allows us to use the libjpeg-turbo library to encode JPEG response for GetMap requests.

Note: It is worth to point out that the module depends on a successful installation of the libjpeg-turbo native libraries (more on this later).

22.9.1 Installing the libjpeg-turbo native library

Installing the libjpeg-turbo native library is a precondition to have the relative GeoServer Map Encoder properly installed; once the GeoServer extension has been installed as we explain in the following section, the needed JARs with the Java bridge to the library are in the classpath, therefore all we need to do is to install the native library itself to start encoding JPEG at turbo speed.

To perform the installation of the libjpeg-turbo binaries (or native library) you have to perform the following steps:

1. go to the download site here and download the latest available stable release (1.2.90 at the time of writing)
2. select the package that matches the target platform in terms of Operating System (e.g. Linux rather than Windows) and Architecture (32 vs 64 bits)

3. perform the installation using the target platform conventions. As an instance for Windows you should be using an installer that installs all the needed libs in a location at user's choice. On Ubuntu Linux systems you can use the *deb* files insted.
4. Once the native libraries are installed, you have to make sure the GeoServer can load them. This should happen automatically after Step 2 on Linux, while on Windows you should make sure that the location where you placed the DLLs is part of the PATH environment variable for the Java Process for the GeoServer.

Warning: When installing on Windows, always make sure that the location where you placed the DLLs is part of the PATH environment variable for the Java Process for the GeoServer. This usually means that you have to add such location to the PATH environmental variable for the user that is used to run GeoServer or the system wide variables.

Warning: When installing on Linux, make sure that the location where you placed the DLLs is part of the LD_LIBRARY_PATH environment variable for the Java Process for the GeoServer. This usually happens automatically for the various Linux packages, but in some cases you might be forced to do that manually

Note: It does not hurt to add also the location where where the native libraries where installed to the Java startup options -Djava.library.path=<absolute_and_valid_path>

22.9.2 Installing the GeoServer libjpeg-turbo extension

Warning: Before moving on make sure you installed the libjpeg-turbo binaries as per the section above.

1. Download the extension from the nightly GeoServer community module builds.

   Warning: Make sure to match the version of the extension to the version of the GeoServer instance!

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

22.9.3 Checking if the extension is enabled

Once the extension is installed, the following lines should appear in the GeoServer log:

```
10-mar-2013 19.16.28 it.geosolutions.imageio.plugins.turbojpeg.TurboJpegUtilities load
INFO: TurboJPEG library loaded (turbojpeg)
```

or:

```
10 mar 19:17:12 WARN [turbojpeg.TurboJPEGMapResponse] - The turbo jpeg encoder is available for usage
```

22.9.4 Disabling the extension

When running GeoServer the turb encoder can be disabled by using the Java switch for the JVM process:

```
-Ddisable.turbojpeg=true
```

In this case a message like the following should be found in the log:

```
WARN [map.turbojpeg] - The turbo jpeg encoder has been explicitly disabled
```

Note: We will soon add a section in the GUI to check the status of the extension and to allow users to enable/disable it at runtime.

22.10 NetCDF Output format

This plugin brings in the ability to encode WCS 2.0.1 Multidimensional output as NetCDF files using the Unidata NetCDF Java library.

22.10.1 Installing the GeoServer NetCDF Output format extension

1. Download the extension from the nightly GeoServer community module builds.

   Warning: Make sure to match the version of the extension to the version of the GeoServer instance!

2. Extract the contents of the archive into the `WEB-INF/lib` directory of the GeoServer installation.

22.10.2 Getting a NetCDF output file

Make sure to specify NetCDF as value of the format parameter within the getCoverage request. As an instance: http://localhost:8080/geoserver/wcs?request=GetCoverage&service=WCS&version=2.0.1&coverageId=it.geosolut netcdf...

22.10.3 Current limitations

- Only WGS84 output CRS is supported
- Input coverages/slices should share the same bounding box (lon/lat coordinates are the same for the whole ND cube)
- NetCDF output will be produced only when input coverages come from a StructuredGridCoverage2D reader (This will allows to query the GranuleSource to get the list of granules in order to setup dimensions slices for each sub-coverage)

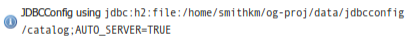
22.11 JDBCConfig

The `JDBCConfig module` enhances the scalibility performance of the GeoServer Catalog. It allows externalising the storage of the Catalog configuration objects (such as workspaces, stores, layers) to a Relational Database Management System, rather than using xml files in the *GeoServer Data Directory*. This way the Catalog can support access to unlimited numbers of those configuration objects efficiently.

22.11.1 Installing JDBCConfig

To install the JDBCConfig module:

1. Download the module. The file name is called `geoserver-*-jdbcconfig-plugin.zip`, where `*` is the version/snapshot name.
2. Extract this file and place the JARs in `WEB-INF/lib`.
3. Perform any configuration required by your servlet container, and then restart. On startup, JDBCConfig will create a configuration directory `jdbcconfig` in the *GeoServer Data Directory*.
4. Verify that the configuration directory was created to be sure installation worked then turn off GeoServer.
5. Configure JDBCConfig (:ref:'community_jdbcconfig_config'), being sure to set `enabled`, `initdb`, and `import` to `true`, and to provide the connection information for an empty database.
6. Start GeoServer again. This time JDBCConfig will connect to the specified database, initialize it, import the old catalog into it, and take over from the old catalog. Subsequent start ups will skip the initialize and import steps unless you re-enable them in `jdbcconfig.properties`.
7. Log in as admin and a message should appear on the welcome page:



22.11.2 JDBCConfig configuration

The JDBCConfig module is configured in the file `jdbcconfig/jdbcconfig.properties` inside the *GeoServer Data Directory*. The following properties may be set:

- `enabled`: Use JDBCConfig. Turn off to use the data directory for all configuration instead.
- `initdb`: Initialize an empty database if this is set on true.
- `import` : The import configuration option tells GeoServer whether to import the current catalog from the file system to the database or not. If set to true, it will be imported and the config option will be set the value 'false' for the next start up to avoid trying to re-import the catalog configuration.
- `initScript`: Path to initialisation script .sql file. Only used if initdb = true.
- `jndiName`: The JNDI name for the data source. Only set this if you want to use JNDI, the JDBC configuration properties may still be set for in case the JNDI Lookup fails.
- `jdbcUrl`: JDBC direct connection parameters.
- `username`: JDBC connection username.
- `password`: JDBC connection password.
- `pool.minIdle`: minimum connections in pool
- `pool.maxActive`: maximum connections in pool
- `pool.poolPreparedStatements`: whether to pool prepared statements
- `pool.maxOpenPreparedStatements`: size of prepared statement cache, only used if pool.poolPreparedStatements = true
- `pool.testOnBorrow`: whether to validate connections when obtaining from the pool
- `pool.validationQuery`: validation query for connections from pool, must be set when pool.testOnBorrow = true