



# *GeoServer*

## **GeoServer User Manual**

*Release 2.1-RC4*

**GeoServer**

April 05, 2011



---

# Contents

---

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Overview . . . . .	3
1.2	History . . . . .	3
1.3	Getting Involved . . . . .	4
1.4	License . . . . .	5
<b>2</b>	<b>Installation</b>	<b>7</b>
2.1	OS-independent binary . . . . .	7
2.2	Web archive (WAR) . . . . .	9
2.3	Windows Installer . . . . .	10
2.4	Mac OS X Installer . . . . .	10
<b>3</b>	<b>Getting Started</b>	<b>13</b>
3.1	Web Administration Interface Quickstart . . . . .	13
3.2	Adding a Shapefile . . . . .	23
3.3	Adding a PostGIS Table . . . . .	30
3.4	Styling a Map . . . . .	39
<b>4</b>	<b>GeoServer Data Directory</b>	<b>41</b>
4.1	Creating a New Data Directory . . . . .	41
4.2	Setting the Data Directory . . . . .	41
4.3	Structure of the Data Directory . . . . .	44
4.4	Migrating a Data Directory between different versions . . . . .	47
<b>5</b>	<b>Web Administration Interface</b>	<b>49</b>
5.1	Interface basics . . . . .	49
5.2	Server . . . . .	51
5.3	Services . . . . .	61
5.4	Data . . . . .	71
5.5	Demos . . . . .	99
5.6	Layer Preview . . . . .	103
<b>6</b>	<b>Working with Data</b>	<b>111</b>
6.1	Shapefile . . . . .	111

6.2	PostGIS	112
6.3	Directory of spatial files	118
6.4	External Web Feature Server	120
6.5	External Web Map Server	122
6.6	Java Properties	125
6.7	ArcGrid	127
6.8	GeoTIFF	128
6.9	GTOPO30	128
6.10	ImageMosaic	132
6.11	WorldImage	132
6.12	ArcSDE	135
6.13	GML	139
6.14	DB2	141
6.15	H2	145
6.16	MySQL	145
6.17	Pregeneralized Features	147
6.18	Oracle	149
6.19	Microsoft SQL Server	151
6.20	VPF	154
6.21	GDAL Image Formats	156
6.22	ImagePyramid	162
6.23	Image Mosaic JDBC	162
6.24	Oracle Georaster	166
6.25	Custom JDBC Access for image data	166
6.26	Database Connection Pooling	168
6.27	SQL views	168
6.28	Application Schema Support	173
<b>7</b>	<b>Filtering in GeoServer</b>	<b>219</b>
7.1	GeoServer supported filter languages	219
7.2	Filter functions	219
7.3	Filter Function Reference	221
<b>8</b>	<b>Styling</b>	<b>229</b>
8.1	Introduction to SLD	229
8.2	SLD Cookbook	231
8.3	SLD Reference	304
8.4	SLD Extensions in GeoServer	328
8.5	SLD Tips and Tricks	338
<b>9</b>	<b>Services</b>	<b>343</b>
9.1	Web Feature Service	343
9.2	Web Map Service	354
9.3	Web Coverage Service	372
9.4	Virtual OWS Services	376
<b>10</b>	<b>RESTful Configuration</b>	<b>381</b>
10.1	Overview of REST	381
10.2	REST Configuration API Reference	381
10.3	REST Configuration Examples	393
<b>11</b>	<b>Advanced GeoServer Configuration</b>	<b>403</b>
11.1	Coordinate Reference System Handling	403
11.2	Advanced log configuration	407
11.3	WMS Decorations	409



<b>12 Security</b>	<b>413</b>
12.1 Accessing secured resources	413
12.2 Users and roles	413
12.3 Service-level security	414
12.4 Layer-level security	415
12.5 REST Security	417
12.6 Disabling security	419
<b>13 Running in a Production Environment</b>	<b>421</b>
13.1 Java Considerations	421
13.2 Container Considerations	423
13.3 Configuration Considerations	424
13.4 Data Considerations	426
13.5 Linux init scripts	428
13.6 Other Considerations	428
13.7 Troubleshooting	429
<b>14 Caching with GeoWebCache</b>	<b>435</b>
14.1 Using GeoWebCache	435
14.2 GeoWebCache Configuration	437
14.3 GeoWebCache Demo page	438
14.4 Seeding and refreshing	439
14.5 Troubleshooting	441
<b>15 Google Earth</b>	<b>443</b>
15.1 Overview	443
15.2 Quickstart	443
15.3 KML Styling	448
15.4 Tutorials	466
15.5 Features	486
<b>16 Extensions</b>	<b>501</b>
16.1 GeoSearch	501
16.2 Imagemap	502
16.3 OGR based WFS Output Format	503
16.4 Cross layer filtering	507
16.5 GeoExt Styler	511
16.6 WFS Versioning	515
16.7 Web Processing Service	516
<b>17 Tutorials</b>	<b>525</b>
17.1 Freemarker Templates	525
17.2 GeoRSS	527
17.3 GetFeatureInfo Templates	531
17.4 Paletted Images	537
17.5 Serving Static Files	549
17.6 WMS Reflector	549
17.7 CQL and ECQL	552
17.8 Using the ImageMosaic plugin	559
17.9 Building and using an image pyramid	578
17.10 Storing a coverage in a JDBC database	582
17.11 Using the GeoTools feature-pregeneralized module	590
17.12 Setting up a JNDI connection pool with Tomcat	599
<b>18 Community</b>	<b>603</b>

18.1	Control flow module . . . . .	603
18.2	GeoServer CSS Module . . . . .	605
18.3	DDS/BIL(World Wind Data Formats) Extension . . . . .	627
18.4	Monitoring . . . . .	628
18.5	GeoServer Printing Module . . . . .	635
18.6	Python . . . . .	635
18.7	SpatiaLite . . . . .	639

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 of this software, we hope that this documentation will be a helpful reference.



---

# 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 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 as you learn and get acquainted, and we try our hardest to keep our code clean and well documented.

### 1.3.2 Documentation

One of the best and most needed ways to help out is with documentation. Our official documentation is contained as part of our official code repository in order to maintain a uniform look and feel. However, we also maintain a *wiki* <<http://geoserver.org>>, where any and all users can post their own documentation, tips and tricks, and any other information that is useful to GeoServer. Like code contributions, if the GeoServer Project Steering Committee deems that user-contributed documentation is a valuable addition to the official documentation base, it will be added.

### 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 code. As GeoServer is open source, everyone is encouraged to fix bugs and submit patches. Even if you are not a core developer, you can still submit a patch through JIRA, and a developer will assess the patch and apply it to the code.

### 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.

### 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 or in 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 commercial organizations offer support for GeoServer, and any improvements made due to that funding will benefit the entire project.

## 1.4 License

GeoServer is free software and is licensed under the [GNU General Public License](#).





---

# Installation

---

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

## 2.1 OS-independent binary

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.

### 2.1.1 Windows

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

#### 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.

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.

## 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.

### 2.1.2 Linux

### 2.1.3 OS X

**Note:** This section is for the OS-independent binary. Please see the section on the [Mac OS X Installer](#) for the wizard-based installer for OS X.

## 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→Utilities→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.

## 2.2 Web archive (WAR)

GeoServer is packaged as a standalone servlet for use with existing servlet container applications such as [Apache Tomcat](#) and [Glassfish](#).

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

### 2.2.1 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.

## 2.2.2 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`.

## 2.2.3 Uninstallation

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

## 2.3 Windows Installer

## 2.4 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.  
**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*.
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!



Figure 2.1: Starting the Mac OSX Installer of GeoServer

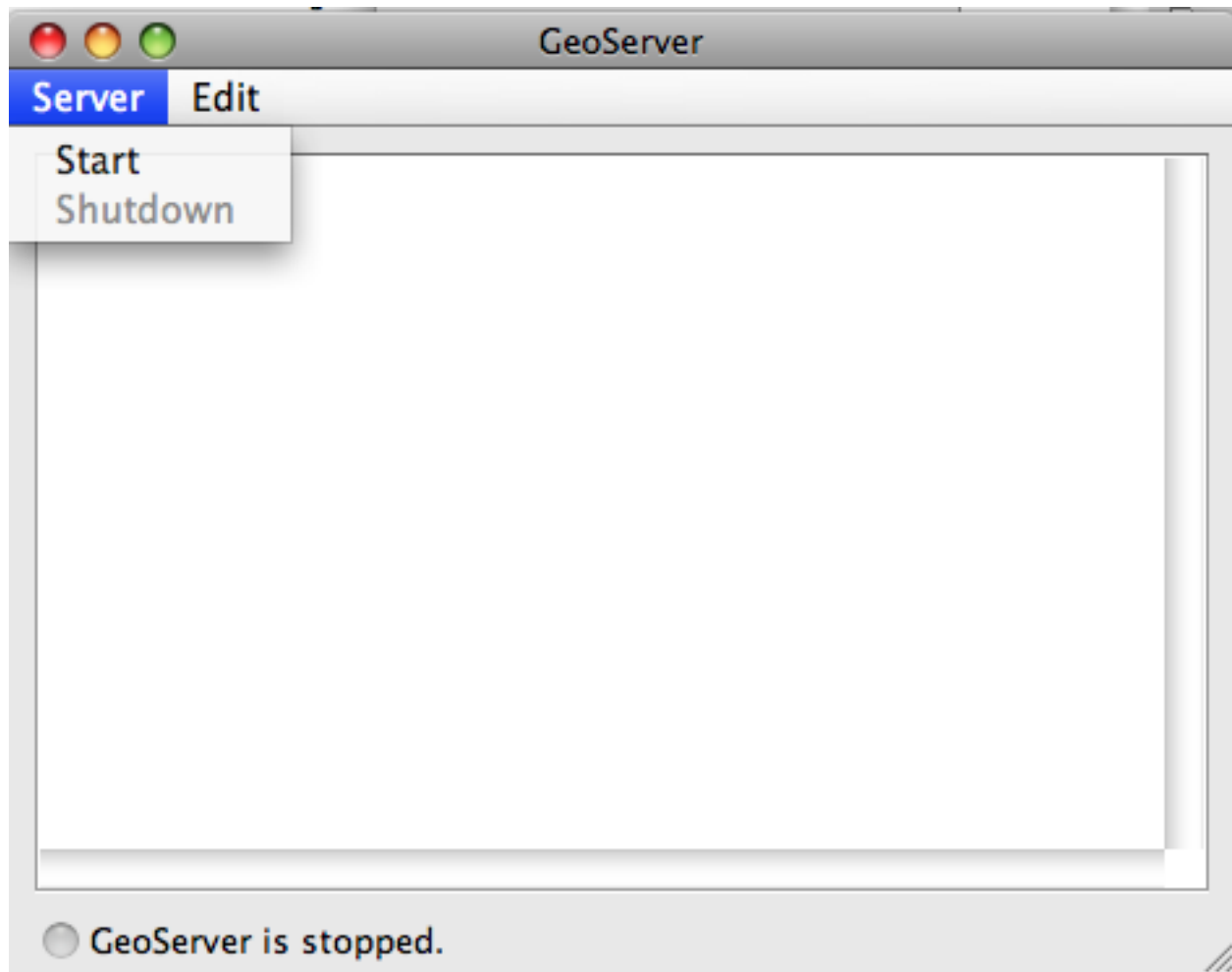


Figure 2.2: *Starting GeoServer*

# Getting Started

This section of the user guide contains an quick overview of GeoServer to get new users performing common tasks quickly and easily.

## 3.1 Web Administration Interface Quickstart

The *Web Administration Tool* is a web based used to configure all aspects of GeoServer, from adding data to tweaking service settings. The web admin tool is accessed via web browser at `http://<host>:<port>/geoserver`. <http://localhost:8080/geoserver/web> in a default installation running on the local host.

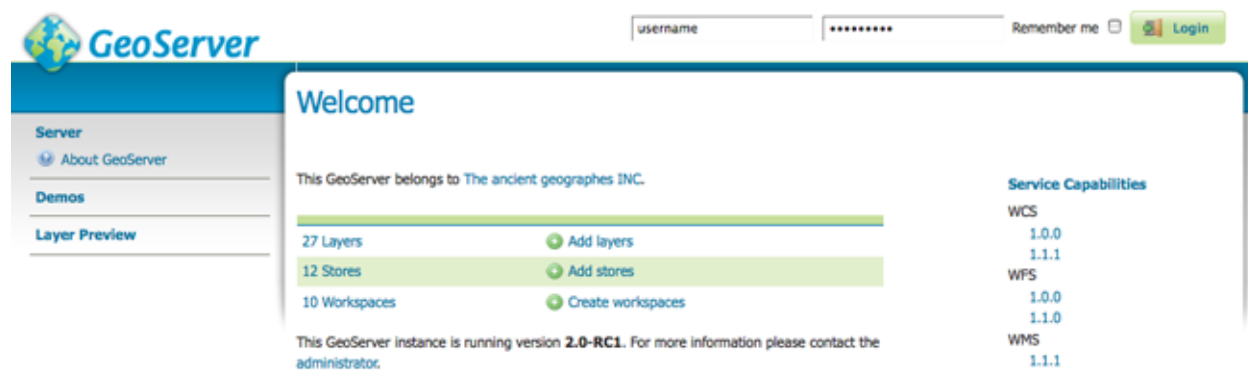


Figure 3.1: Welcome Page

### 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 only by editing the `security/users.properties` file in the *GeoServer Data Directory*.



Figure 3.2: *Login*

### 3.1.2 Server

The `:guilabel: Server` section of the web admin 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 offers a summary of server configuration parameters and run-time status

The **Contact Information** section 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.

### 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 sub-sections 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.)



## Server Status

Summary of server configuration and status

		Action
<b>Locks</b>	0	<a href="#">Free locks</a>
<b>Connections</b>	6	
<b>Memory Usage</b>	28 MB	<a href="#">Free memory</a>
<b>JVM Version</b>	Sun Microsystems Inc.: 1.7.0-internal (Java HotSpot(TM) Server VM)	
<b>Native JAI</b>	false	
<b>Native JAI ImageIO</b>	false	
<b>JAI Maximum Memory</b>	377 MB	
<b>JAI Memory Usage</b>	0 KB	<a href="#">Free memory</a>
<b>JAI Memory Threshold</b>	75.0	
<b>Number of JAI Tile Threads</b>	8	
<b>JAI Tile Thread Priority</b>	5	
<b>Update Sequence</b>	35	
<b>Resource Cache</b>		<a href="#">Clear</a>
<b>Configuration and catalog</b>		<a href="#">Reload</a>

GeoServer 

Timestamps	
<b>GeoServer</b>	Jul 14, 3:07 PM
<b>Configuration</b>	Jul 14, 3:07 PM
<b>XML</b>	Mar 14, 2:15 PM

Figure 3.3: Status Page

## Contact Information

Set the contact information for this server.

### Contact

### Organization

### Position

### Address Type

### Address

### City

### State

### ZIP code

### Country

Figure 3.4: *Contact Page*

## Global Settings

Settings that apply to the entire server.

☐ Verbose Messages

☐ Verbose Exception Reporting

### Number of Decimals

### Character Set

### Proxy Base URL

### Logging Profile

DEFAULT\_LOGGING.properties  
VERBOSE\_LOGGING.properties  
PRODUCTION\_LOGGING.properties  
GEOTOOLS\_DEVELOPER\_LOGGING.properties  
GEOSERVER\_DEVELOPER\_LOGGING.properties

☒ Log to StdOut

### Log Location

Figure 3.5: Global Settings Page

## JAI Settings

Administer settings related to Java Advanced Imaging.

### Memory Capacity (0-1)

### Memory Threshold (0-1)

### Tile Threads

### Tile Threads Priority

☐ Tile Recycling☐ Image I/O Caching☒ JPEG Native Acceleration☒ PNG Native Acceleration☐ Mosaic Native Acceleration

Figure 3.6: *JAI Settings*

## About GeoServer

General information about GeoServer

### GeoServer 2.0-RC1

The GeoServer project is a full transactional Java (J2EE) implementation of the OpenGIS Consortium's Web Feature Server specification and Web Coverage Server specification, with an integrated Web Map Server.

The documentation for this release is available online at the following link. The GeoServer wiki is used for the latest updates; please share your experiences, hints and tips with GeoServer there. The task tracker is the place to report feature requests and bugs. Also please take a moment to add yourself to the User Map to show your support for GeoServer.


- [Documentation](#)
- [Wiki](#)
- [Bug Tracker](#)

Figure 3.7: *About Section*

## Styles

Manage the Styles published by GeoServer

 Add a new style

 Removed selected style(s)

     Results 1 to 22 (out of 22 items)

 Search

☐ Style Name

☐ burg

☐ giant\_polygon

☐ capitals

☐ simple\_streams

☐ pophatch

☐ restricted

☐ tiger\_roads

☐ poly\_landmarks

☐ green


☐ rain

Figure 3.8: *Styles View page*

## Layers





















Manage the layers being published by GeoServer

 Add a new resource

 Remove selected resources

<< < 1 2 > >> Results 1 to 10 (out of 19 items)

 Search

<input type="checkbox"/>	Type	Workspace	Store	Layer Name	Enabled?	Native SRS
<input type="checkbox"/>		nurc	arcGridSample	Arc_Sample		EPSG:4326
<input type="checkbox"/>		nurc	img_sample2	Pk50095		EPSG:32633
<input type="checkbox"/>		nurc	mosaic	mosaic		EPSG:4326
<input type="checkbox"/>		nurc	worldImageSample	Img_Sample		EPSG:4326
<input type="checkbox"/>		sf	sf	archsites		EPSG:26713
<input type="checkbox"/>		sf	sf	bugsites		EPSG:26713
<input type="checkbox"/>		sf	sf	restricted		EPSG:26713
<input type="checkbox"/>		sf	sf	roads		EPSG:26713
<input type="checkbox"/>		sf	sf	streams		EPSG:26713
<input type="checkbox"/>		sf	sfdem	sfdem		EPSG:26713

<< < 1 2 > >> Results 1 to 10 (out of 19 items)

Figure 3.9: *Layers View*

<input type="checkbox"/> Style Name	<input type="checkbox"/> Style Name
<input type="checkbox"/> burg	<input type="checkbox"/> burg
<input type="checkbox"/> giant_polygon	<input type="checkbox"/> capitals
<input type="checkbox"/> capitals	<input type="checkbox"/> cite_lakes
<input type="checkbox"/> simple_streams	<input type="checkbox"/> concat
<input type="checkbox"/> pophatch	<input type="checkbox"/> dem
<input type="checkbox"/> restricted	<input type="checkbox"/> flags
<input type="checkbox"/> tiger_roads	<input type="checkbox"/> giant_polygon
<input type="checkbox"/> poly_landmarks	<input type="checkbox"/> grass
<input type="checkbox"/> green	<input type="checkbox"/> green
<input type="checkbox"/> rain	<input type="checkbox"/> line

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

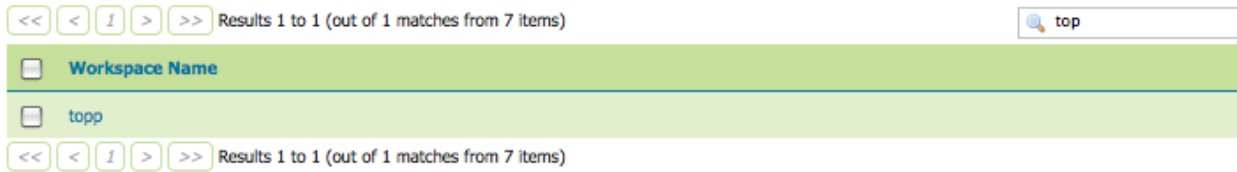


Figure 3.11: Search results for the query “top”.

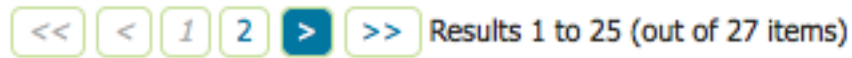


Figure 3.12: Page scroll for data types.

## Stores

Manage the stores providing data to GeoServer

-  [Add new Store](#)
-  [Remove selected Stores](#)

Figure 3.13: Buttons to add and remove Stores

## Stores

Manage the stores providing data to GeoServer

-  [Add new Store](#)
-  [Remove selected Stores](#)



















Results 1 to 9 (out of 9 items)					Search
<input type="checkbox"/>	Type	Workspace	Store Name	Enabled?	
<input type="checkbox"/>		nurc	arcGridSample		
<input type="checkbox"/>		nurc	img_sample2		
<input type="checkbox"/>		nurc	mosaic		
<input checked="" type="checkbox"/>		nurc	worldImageSample		
<input type="checkbox"/>		sf	sfdem		
<input type="checkbox"/>		sf	sf		
<input checked="" type="checkbox"/>		tiger	nyc		
<input type="checkbox"/>		topp	states_shapefile		
<input type="checkbox"/>		topp	taz_shapes		
Results 1 to 9 (out of 9 items)					

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 Layers Preview

The **Layers Preview** page provides layer views 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.

#### Layer Preview

List of all layers configured in GeoServer and provides previews in various formats for each.

<<

<

1

>

>>

Results 1 to 19 (out of 19 items)

Search

Type	Name	Title	Common Formats		All Formats
	nunc:Arc_Sample	A sample ArcGrid file	OpenLayers	KML	Select one
	nunc:Pk50095	Pk50095 is a A raster file accompanied by a spatial data file	OpenLayers	KML	Select one
	nunc:mosaic	Sample PNG mosaic	OpenLayers	KML	Select one
	nunc:Img_Sample	North America sample imagery	OpenLayers	KML	Select one
	sf:archsites	Spearfish archeological sites	OpenLayers	KML GML	Select one
	sf:bugsites	Spearfish bug locations	OpenLayers	KML GML	Select one
	sf:restricted	Spearfish restricted areas	OpenLayers	KML GML	Select one
	sf:roads	Spearfish roads	OpenLayers	KML GML	Select one
	sf:streams	Spearfish streams	OpenLayers	KML GML	Select one
	sf:sfдем	sfдем is a Tagged Image File Format with Geographic information	OpenLayers	KML	Select one
	tiger:poi	Manhattan (NY) points of interest	OpenLayers	KML GML	Select one

Figure 3.16: *Layer's Preview page*



Each layer row consists of a **type**, **name**, **title**, and available formats for viewing. **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. **Common Formats** include OpenLayers and KML output, while the **All Formats** include additional output formats for further use or data sharing.

Type	Name	Title	Common Formats	All Formats
	nurc:Arc_Sample	A sample ArcGrid file	OpenLayers KML	Select one ▾

Figure 3.17: Single Layer preview row

## 3.2 Adding a Shapefile

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

**Note:** This tutorial assumes that GeoServer is running on <http://localhost:8090/geoserver/web>.

### 3.2.1 Getting Started

1. Download the file **nyc\_roads.zip**. This file 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 consists of the following four files:

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

3. 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 were made to the GeoServer file structure, the path should be `geoserver/data_dir/data/nyc_roads`.

### 3.2.2 Create a New Workspace


The first step is to create a *workspace* for the Shapefile. The 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 the [Logging In](#) quick start.
3. Navigate to *Data*→*Workspaces*.
4. To create a new workspace click, select the **Add new workspace** button. You will be prompted to enter a workspace **Name** and **Namespace URI**.
5. Enter the name *nyc\_roads* and the URI `http://opengeo.org/nyc_roads` A workspace name is a name describing your project and cannot exceed ten characters or contain a space. A Namespace URI (Uniform Resource Identifier), is typically a URL associated with your project, with perhaps a different trailing identifier.


## Workspaces

Manage GeoServer workspaces

 Add new workspace

 Remove selected workspace(s)

     Results 1 to 7 (out of 7 items)

 Search

<input type="checkbox"/> Workspace Name
<input type="checkbox"/> sf
<input type="checkbox"/> topp
<input type="checkbox"/> it.geosolutions
<input type="checkbox"/> sde
<input type="checkbox"/> nurc
<input type="checkbox"/> tiger
<input type="checkbox"/> cite






     Results 1 to 7 (out of 7 items)

Figure 3.18: *Workspaces page*

## New Workspace

Configure a new workspace

**Name**

**Namespace URI**

The namespace uri associated with this workspace

**Submit**

**Cancel**

Figure 3.19: *Configure a New Worksapce*

# New Workspace

Configure a new workspace

**Name**

**Namespace URI**

The namespace uri associated with this workspace



Figure 3.20: NYC Roads Workspace

- Click the **Submit** button. GeoServer will append the nyc\_roads workspace to the bottom of the Workspace View list.

## 3.2.3 Create a Store

- Navigate to *Data→Stores*.
- In order to add the nyc\_roads data, we need to create a new Store. Click on the **Add new store** button. You will be redirected to a list of data types GeoServer supports.
- Because nyc\_roads is a shapefile, select **Shapefile: ESRI(tm) Shapefiles (.shp)**.
- On the **New Vector Data Source** page begin by configuring the **Basic Store Info**. Select the workspace nyc\_roads from the drop down menu, type NYC Roads for the name and enter a brief description, such as Roads in New York City.
- Under the **Connections Parameters** specify the location of the shapefile—`file:data/nyc_roads/nyc_roads.shp`.
- Press Save. You will be redirected to **New Layer chooser** page in order to configure nyc\_roads layer.







## 3.2.4 Layer Configuration

- On the **New Layer chooser** page, select the Layer name nyc\_roads.

## New data source

Choose the type of data source you wish to configure

### Vector Data Sources

-  [Directory of spatial files](#) - Takes a directory of spatial data files and exposes it as a data store
-  [PostGIS NG](#) - PostGIS Database
-  [PostGIS NG \(JNDI\)](#) - 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 provides access to the Features published by the server, and the ability to perform transactions on the server (when supported / allowed).

### 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

Figure 3.21: *Data Sources*

2. The following configuration define the data and publishing parameters for a layer. Enter a short **Title** and **Abstract** for the nyc\_roads shapefile.
3. Generate the shapefile's *bounds* by clicking the **Compute from data** and then **Compute from Native bounds**.
4. Set the shapefile's *style* by first moving over to the **Publishing** tab.
5. The select **line** from the **Default Style** drop down list.
6. Finalize your data and publishing configuration by scrolling to the bottom and clicking **Save**.

### 3.2.5 Preview the Layer

1. In order to verify that the nyc\_roads is probably published we will preview the layer. Navigate to the **Map Preview** and search for the nyc\_roads:nyc\_roads link.
2. Click on the **OpenLayers** link under the **Common Formats** column.
3. Success! An OpenLayers map should load with the default line style.

## New Vector Data Source

Shapefile

ESRI(tm) Shapefiles (\*.shp)

### Basic Store Info

#### Workspace

nyc\_roads

#### Data Source Name

NYC Roads

#### Description

Roads in New York City

☒ Enabled

### Connection Parameters

#### URL

file:data/nyc\_roads/nyc\_roads.shp

#### namespace

http://opengeo.org/nyc\_roads

☐ create spatial index

#### charset

ISO-8859-1

☐ memory mapped buffer

Figure 3.22: Data Info and Parameters for nyc\_roads

## New Layer chooser

Here is a list of resources contained in the store 'NYC Roads'. Click on the layer you wish to configure

<<
<
1
>
>>

Results 1 to 1 (out of 1 items)

Search

Layer with namespace and prefix	Published
nyc_roads	

<<
<
1
>
>>

Results 1 to 1 (out of 1 items)

Figure 3.23: New Layer Chooser

## nyc\_roads:nyc\_roads

Configure the resource and publishing information for the current layer

Data

Publishing

---

### Basic Resource Info

**Name**

**Title**

**Abstract**

Shapefile of NYC roads

Figure 3.24: Basic Resource Information for Shapefile

### Bounding Boxes

#### Native Bounding Box

Min X	Min Y	Max X	Max Y
984,018.166	207,673.095	991,906.497	219,622.54

Compute from data

#### Lat/Lon Bounding Box

Min X	Min Y	Max X	Max Y
-74.001	40.737	-73.972	40.769

Compute from native bounds

Figure 3.25: Generate Bounding Box

## nyc\_roads:nyc\_roads

Configure the resource and publishing information for the current layer

Data

Publishing

---

### Basic Settings

**Name**

nyc\_roads

StyleImpl[cite\_lakes]

StyleImpl[concat]

StyleImpl[dem]

StyleImpl[flags]

StyleImpl[giant\_polygon]

StyleImpl[grass]

StyleImpl[green]

StyleImpl[line]

StyleImpl[medford\_buildings]

StyleImpl[medford\_citylimits]

StyleImpl[medford\_parks]

StyleImpl[medford\_streets]

StyleImpl[medford\_zoning]

StyleImpl[poi]

StyleImpl[point]

StyleImpl[poly\_landmarks]

StyleImpl[polygon]

StyleImpl[pophatch]

StyleImpl[population]

StyleImpl[rain]

StyleImpl[line]

Figure 3.26: Select Default Style

## Layer Preview

List of all layers configured in GeoServer and provides previews in various formats for each.

<< < 1 2 > >> Results 1 to 25 (out of 26 items)

Type	Name	Title	Common Formats	All Formats
	nurc:Arc_Sample	A sample ArcGrid file	OpenLayers KML	Select one
	nurc:Img_Sample	North America sample imagery	OpenLayers KML	Select one
	nurc:Pk50095	Pk50095 is a A raster file accompanied by a spatial data file	OpenLayers KML	Select one
	nurc:mosaic	Sample PNG mosaic	OpenLayers KML	Select one
	nyc_roads:nyc_roads	Subset of NYC roads	OpenLayers KML GML	Select one

Figure 3.27: Layer Preview

## 3.3 Adding a PostGIS Table

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

**Note:** This tutorial assumes that GeoServer is running on <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 subset 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:

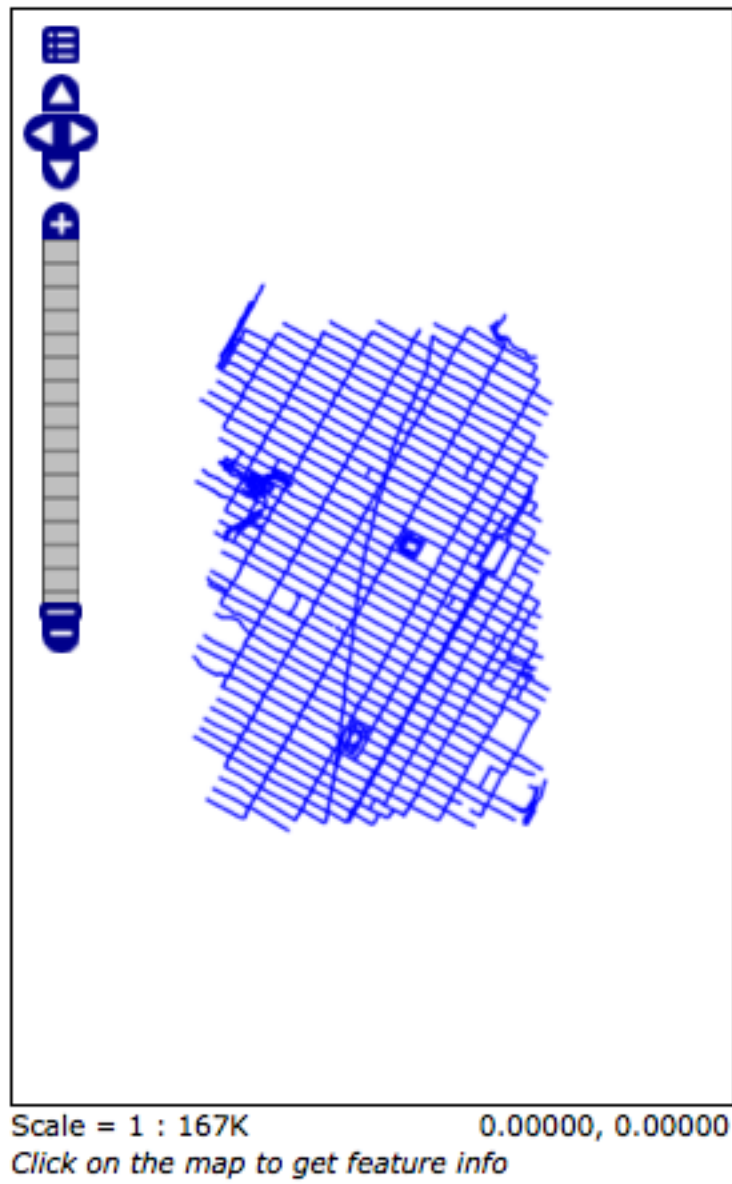
```
psql -f nyc_buildings.sql nyc
```

### 3.3.2 Create a new 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*.









Figure 3.28: *OpenLayers map of nyc\_roads*

## New data source

Choose the type of data source you wish to configure

### Vector Data Sources

-  [Directory of spatial files](#) - Takes a directory of spatial data files and exposes it as a data store
-  [PostGIS NG](#) - PostGIS Database
-  [PostGIS NG \(JNDI\)](#) - 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 provides access to the Features published by the server, and the ability to perform transactions on the server (when supported / allowed).

### 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

Figure 3.29: *Adding a New Data Source*

3. Create a new data store by clicking the `PostGIS NG` link.
4. Keeping the default **Workspace** enter **Basic Store Info** of Name and Description.

### Basic Store Info

---

#### Workspace

#### Data Source Name

#### Description

☒ Enabled

Figure 3.30: *Basic Store Info*

5. Specify the PostGIS database **Connection Parameters**

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

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

6. Click the **Save** button.

### 3.3.3 Layer Configuration

1. Navigate to *Data*→*Layers*.
2. Select **Add a new resource** button.
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 following configurations define the data and publishing parameters for a layer. Enter the **Basic Resource Info** for nyc\_buildings.
6. Generate the database *bounds* by clicking the **Compute from data** and then **Compute from Native bounds**.
7. Set the layer's *style* by first moving over to the **Publishing** tab.
8. The select **polygon** from the **Default Style** drop down list.
9. Finalize your data and publishing configuration by scrolling to the bottom and clicking **Save**.

### 3.3.4 Preview the Layer

1. In order to verify that the nyc\_building is probably published we will preview the layer. Navigate to the **Map Preview** and search for the cite:nyc\_buildings link.
2. Click on the **OpenLayers** link under the **Common Formats** column.
3. Success! An OpenLayers map should load with the default polygon style.

### Connection Parameters

---

**dbtype**

**host**

**port**

**database**

**schema**

**user**

**passwd**

**namespace**

<http://www.opengeospatial.net/cite>

**max connections**

**min connections**

**fetch size**

**Connection timeout**

☒ validate connections

☒ Loose bbox

☐ preparedStatements

Figure 3.31: *Connection Parameters*

## New Layer chooser

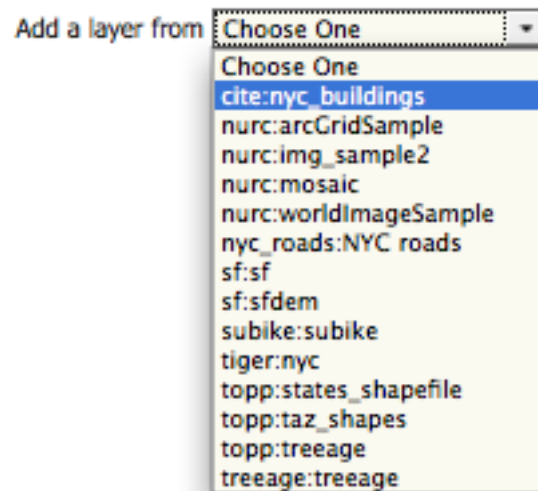


Figure 3.32: *New Layer drop down selection*

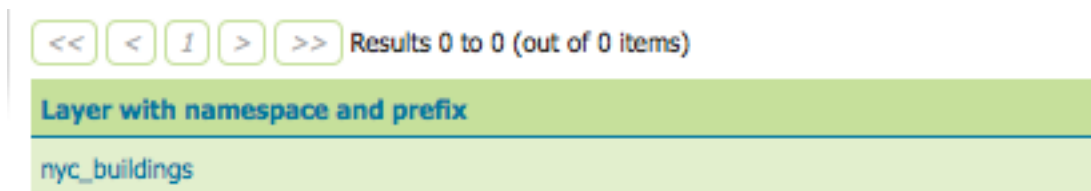


Figure 3.33: *New Layer row*

## cite:nyc\_buildings

Configure the resource and publishing information for the current layer

Data

Publishing

---

### Basic Resource Info

---

**Name**

**Title**

**Abstract**

Figure 3.34: Basic Resource Info

### Bounding Boxes

#### Native Bounding Box

Min X	Min Y	Max X	Max Y
983,837.625	207,499.469	991,858.938	218,794.359

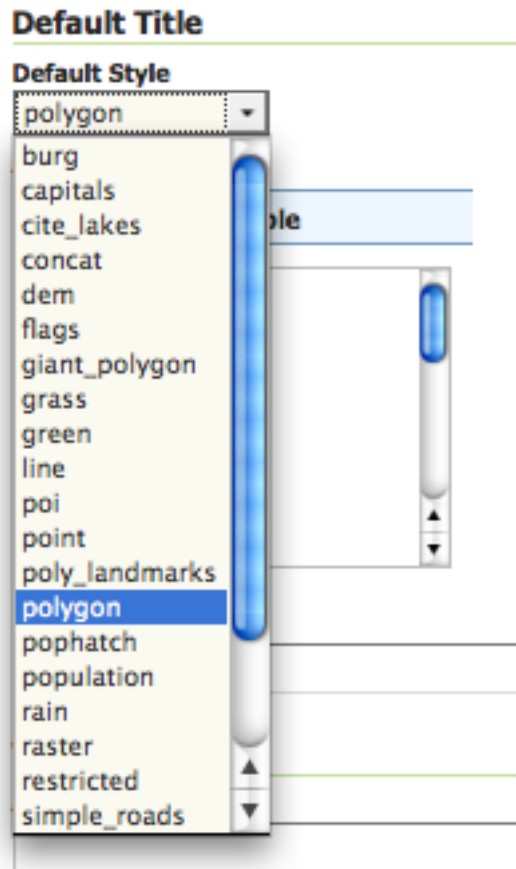
[Compute from data](#)

#### Lat/Lon Bounding Box

Min X	Min Y	Max X	Max Y
-74.002	40.736	-73.973	40.767

[Compute from native bounds](#)

Figure 3.35: Generate Bounding Box

Figure 3.36: *Select Default Style*

## Layer Preview

List of all layers configured in GeoServer and provides previews in various formats for each.

<< < 1 2 > >> Results 1 to 25 (out of 27 items)

Type	Name	Title	Common Formats	All Formats
	cite:nyc_buildings	NYC Buildings	OpenLayers KML GML	Select one <input type="text"/>

Figure 3.37: *Layer Preview*

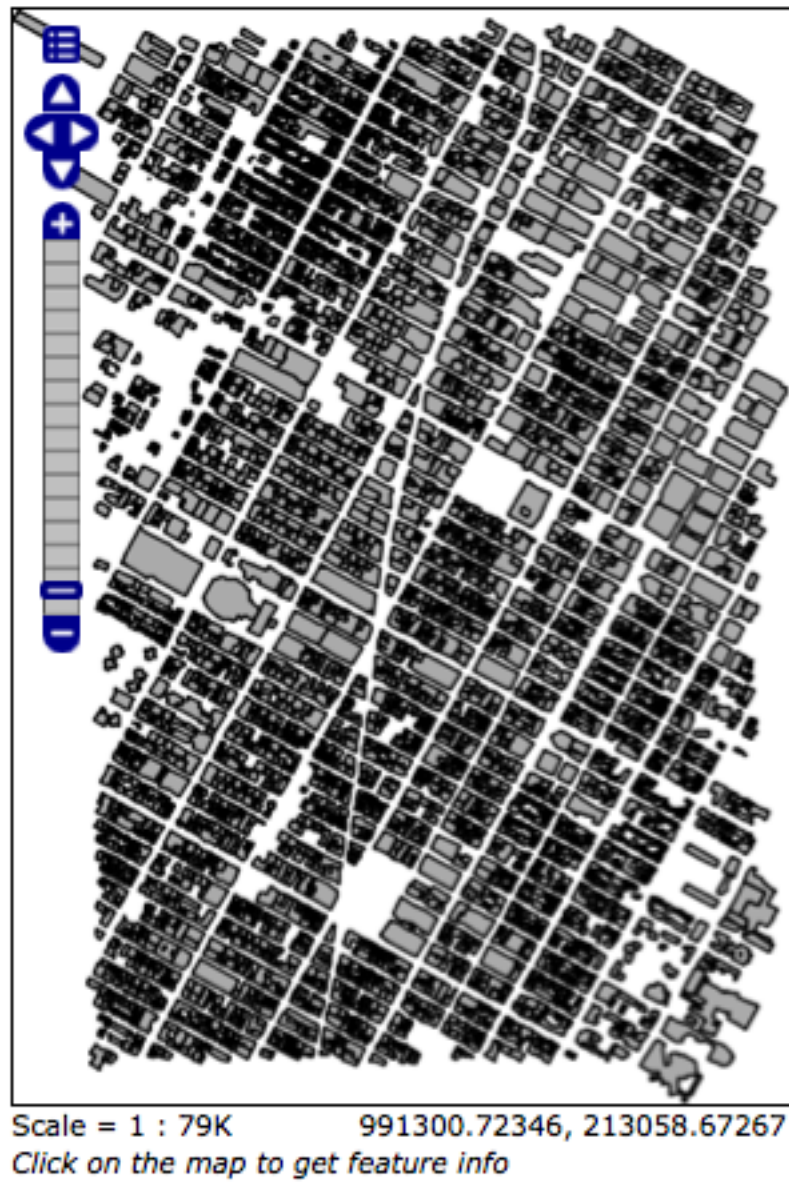


Figure 3.38: *OpenLayers map of nyc\_buildings*



## 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.

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 [Adding a Shapefile](#) and [Adding 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



---

# GeoServer Data Directory

---

The GeoServer *data directory* is the location on the file system where GeoServer stores all of its configuration. This configuration defines such things as: What data is served by GeoServer? Where is that data located? How should services such as WFS and WMS interact with and server that data? And more. The data directory also contains a number of support files used by GeoServer for various purposes.

In general users does not need to know about the structure of the data directory. But it is a good idea to define an external data directory when going to production, to make it easier to upgrade.

Or to learn how to create a directory for a GeoServer installation jump to the [Creating a New Data Directory](#) section. [Creating a New Data Directory](#) contains details on how to make a GeoServer use an existing data directory. To learn more about the structure of the GeoServer data directory continue onto 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 1.7.0`.

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

### 4.2.1 Windows

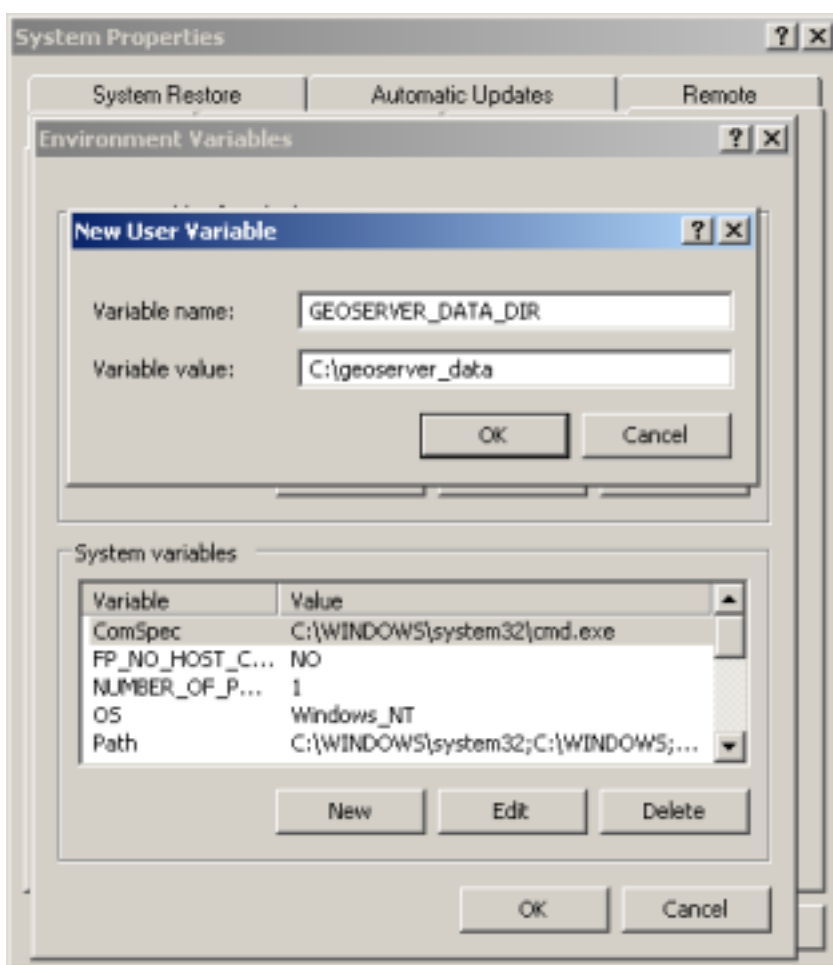
On Windows platforms the location of the GeoServer data directory is controlled by the `GEOSERVER_DATA_DIR` environment variable.

**Note:** When the `GEOSERVER_DATA_DIR` environment variable is not set, the directory `data_dir` under the root of the GeoServer installation is used.

To set the `GEOSERVER_DATA_DIR`:

On **Windows XP** systems:

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.



On **Windows Vista** systems:

### 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 is done for the user GeoServer will be run by.

### 4.2.3 Mac OS X

If running the binary version of GeoServer on Mac OS X then the data directory is set in the exact same way as linux.

If using the Mac OS X binary, then set the `GEOSERVER_DATA_DIR` environment variable to the file location. See [this page](#) for details on how to set an environment variable in Mac OS X

### 4.2.4 Web Archive

When running GeoServer inside of 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

Servlet context parameter's are specified 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

Depending on the servlet container used it is also possible to specify the data directory location with a Java System Property. This method can be useful during upgrades, as it prevents the need to set the data directory on every single upgrade.

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

Setting the Java System Property is dependent on the servlet container.

#### In 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"
```

**In 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>` inside of 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

### 4.3.1 Introduction

The structure of the data directory at this point is likely only of interest to core developers. Previously users would often modify their data directory directly to programmatically make changes to their GeoServer configuration. The new route to do this is with the [RESTful Configuration](#) API, and is the only recommended option.

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

```
data_directory/
  global.xml
  logging.xml
  wms.xml
  wfs.xml
  wcs.xml
  data/
  demo/
  geosearch/
  gwc/
  layergroups/
  palettes/
  plugIns/
  security/
  styles/
  templates/
  user_projections/
  workspaces
  www/
```

### 4.3.2 The .xml files

The top level xml files save the information about the services and various global options.

File	Description
global.xml	Contains settings that go across services, including contact information, JAI settings, character sets and verbosity.
logging.xml	Specifies the logging level, location, and whether it should log to std out.
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.

### 4.3.3 workspaces

The various workspaces directories contain metadata about “layers” which are published by GeoServer. Each layer will have a layer.xml file associated with it, as well as either a coverage.xml or a featuretype.xml file depending on whether it’s a *raster* or *vector*.

### 4.3.4 data

Not to be confused with the “GeoServer data directory” itself, the `data` directory is a location where actual data can be stored. 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 inside of the `data` directory is portability. Consider a shapefile located external to the data directory at a location `C:\gis_data\foo.shp`. The `datastore` entry in `catalog.xml` for this shapefile would 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 problem exists in that the location `C:\gis_data\foo.shp` probably does not exist on the second host. So either the file must be copied to the new host, or `catalog.xml` must be changed to reflect a new location.

Such steps can be avoided by storing `foo.shp` inside of the `data` directory. In such a 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 re-written to be relative. In this way the entire data directory can be archived, copied to the new host, un-archived, and used directly with no additional changes.

### 4.3.5 demo

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

### 4.3.6 geosearch

The `geosearch` directory is not named quite correctly. It contains information for regionation of KML files.

### 4.3.7 gwc

This directory holds the cache created by the embedded GeoWebCache service.

### 4.3.8 layergroups

Contains information on the layer groups configurations.

### 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.

### 4.3.10 security

The `security` directory contains all 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 a number of 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](#) for more information about styling and SLD .

### 4.3.12 templates

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

### 4.3.13 user\_projections

The `user_projections` directory contains a single file called `epsg.properties` which is used to define *custom* spatial reference systems which are not part of the official [EPSG database](#).

### 4.3.14 www

The `www` directory is used to allow GeoServer to act like a regular web server and serve regular files. While not a replacement for a full blown web server the `www` directory can be useful for serving [OpenLayers](#) map applications.



## 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`.



---

# Web Administration Interface

---

The Web Administration Interface is a web-based tool for configuring all aspects of GeoServer, from adding data to tweaking service settings. This is accessed via a web browser at <http://localhost:8080/geoserver/web> in a default GeoServer installation, although this URL may vary depending on how GeoServer is installed.

## 5.1 Interface basics

This section will introduce the basic concepts of the web administration interface (often abbreviated to “web admin” in the text.)

### 5.1.1 Welcome Page

When using most common installations, GeoServer will start a web server on localhost at port 8080 accessible by 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 show up in your browser.

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 persons 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](#).

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

When logged in, additional options will be presented.

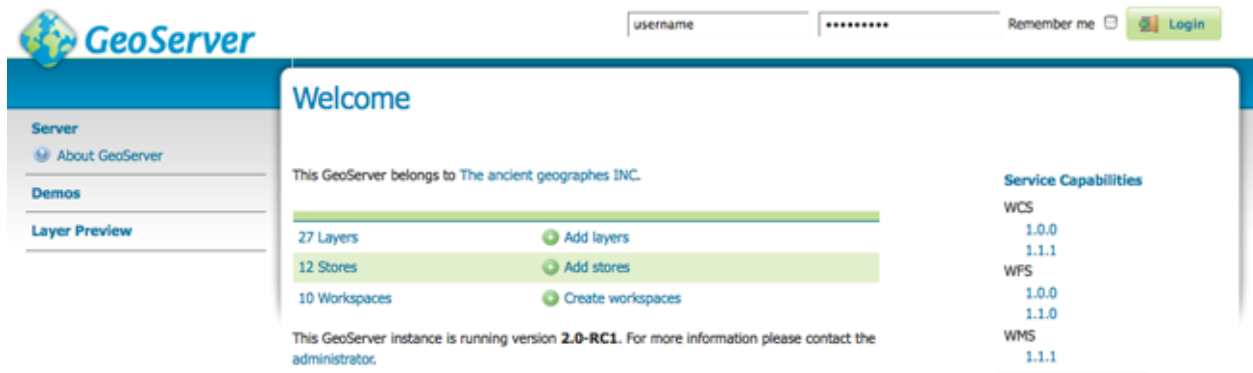


Figure 5.1: Welcome Page



Figure 5.2: Login

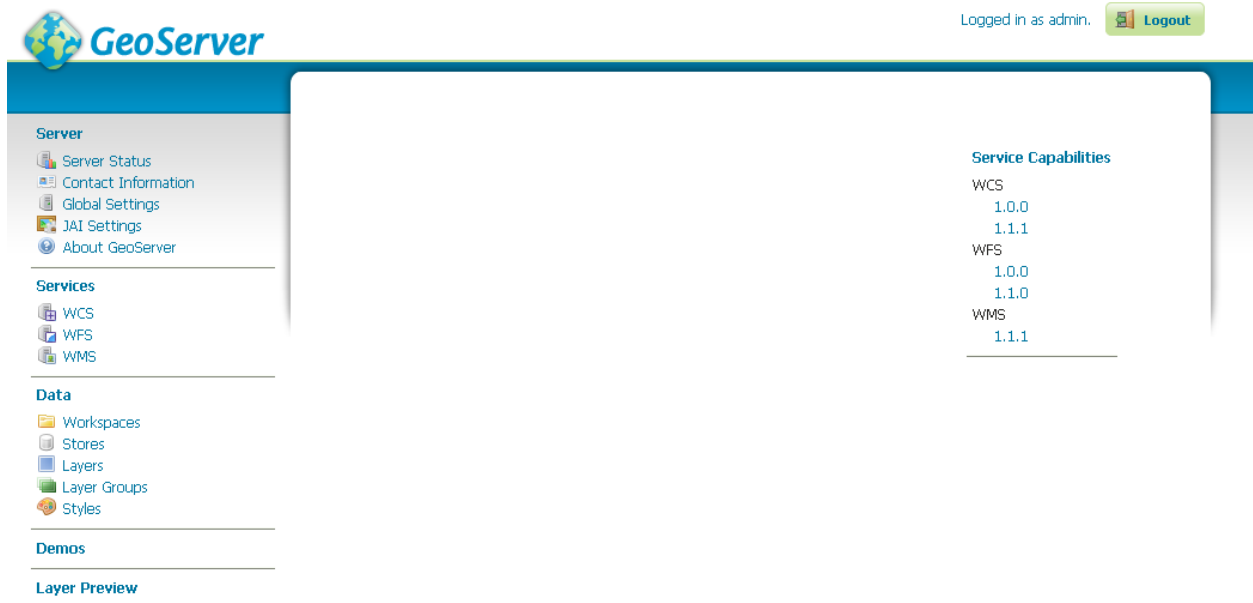


Figure 5.3: Additional options when logged in

Geoserver Web Coverage Service (WCS), Web Feature Service (WFS), and Web Map Service (WMS) configurations specifications can be accessed from this welcome page as well. See the section on [Services](#) for more information.

## 5.2 Server

The Server section of the [Web Administration Interface](#) provides access to GeoServer environment information. It is a combination of diagnostic and configuration tools, and can be particularly useful for debugging.

### 5.2.1 Status

The Server Status page offers a summary of server configuration parameters and run-time status. It is intended to assist diagnostics within a testing environment.

#### Status Field Descriptions

The following describes current status indicators

## Server Status

Summary of server configuration and status

		Action
Locks	0	<a href="#">Free locks</a>
Connections	6	
Memory Usage	28 MB	<a href="#">Free memory</a>
JVM Version	Sun Microsystems Inc.: 1.7.0-internal (Java HotSpot(TM) Server VM)	
Native JAI	false	
Native JAI ImageIO	false	
JAI Maximum Memory	377 MB	
JAI Memory Usage	0 KB	<a href="#">Free memory</a>
JAI Memory Threshold	75.0	
Number of JAI Tile Threads	8	
JAI Tile Thread Priority	5	
Update Sequence	35	
Resource Cache		<a href="#">Clear</a>
Configuration and catalog		<a href="#">Reload</a>

GeoServer 

Timestamps	
GeoServer	Jul 14, 3:07 PM
Configuration	Jul 14, 3:07 PM
XML	Mar 14, 2:15 PM

Figure 5.4: *Status Page*

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 held 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 currently 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 being used to power the server. Here the JVM is Apple Inc.: 1.5.0_16.
Native JAI	GeoServer uses <a href="#">Java Advanced Imaging</a> (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 <a href="#">JAI Image IO</a> (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 the scheduler to handle tiles.
JAI Tile Thread Priority	Schedules the global tile scheduler priority. The priority value 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 <code>\$(GEOSERVER_DATA_DIR)/user_projections/epsg.properties</code> .
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 publically accessible. We recommend filling out this form with appropriate information, in order for people to contact you.

## Contact Information

Set the contact information for this server.

### Contact

### Organization

### Position

### Address Type

### Address

### City

### State

### ZIP code

### Country

Figure 5.5: *Contact Page*



## Contact Information Fields

Field	Description
Contact	Contact information for webmaster
Organization	The name of the organization with which the contact is affiliated.
Position	The position of the contact within their organization.
Address Type	The type of address specified, such as postal.
Address	The actual street address.
City	The city of the address.
State	The state or province of the address.
Zip code	The postal code for the address.
Country	The country of the address.
Telephone	The contact phone number.
Fax	The contact Fax number.
Email	The contact email address.

### 5.2.3 Global Settings

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

#### Global Setting Fields

**Verbose Messages:** When enabled, Verbose Messages tells 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 bandwidth we recommended this option only for testing purposes.

**Verbose Exception Reporting:** Instead of the one line error message, enabled Verbose Exception Reporting returns service exceptions with full Java stack traces. Verbose exception reporting writes to the GeoServer log file and offers one of the most useful configuration options for debugging.

**Number of Decimals:** Refers to the number of decimal places returned in a GetFeature response. Also useful in optimizing bandwidth.

*Character Set:\** Specifies the global character encoding that will be used in XML responses. We recommend the default UTF-8 for most users but support all character sets listed on the [IANACharset Registry](#), and have an available Java implementation.

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

**Logging Profile:** Corresponds to a log4j configuration file in GeoServer's data directory. (Apache [log4j](#) is a Java-based logging utility.) By default, there are five logging profiles set in GeoServer's configurations file; customized profiles can be added by editing the log4j file.

There are six logging levels used by log. 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* 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. *Verbose Logging* provides much more detail by enables DEBUG level logging on GeoTools, GeoServer, and VFNY.

## Global Settings

Settings that apply to the entire server.

☐ Verbose Messages

☐ Verbose Exception Reporting

### Number of Decimals

### Character Set

### Proxy Base URL

### Logging Profile

DEFAULT\_LOGGING.properties

VERBOSE\_LOGGING.properties

PRODUCTION\_LOGGING.properties

GEOTOOLS\_DEVELOPER\_LOGGING.properties

GEOSERVER\_DEVELOPER\_LOGGING.properties

☒ Log to StdOut

### Log Location

Figure 5.6: *Global Settings Page*

3. *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.
4. *GeoTools Developer Logging* is a verbose logging profile that includes DEBUG information only on GeoTools. This developer profile is recommended for active debugging of GeoTools.
5. *GeoServer Developer Logging* is a verbose logging profile that includes DEBUG information on GeoServer and VFNY. This developer profile is recommended for active debugging of GeoServer.

**Log to StdOut:** In general, StdOut (Standard output) refers to where a program writes its output data. In GeoServer, the Log to StdOut checkbox enables logging to the text terminal which initiated the program, most often the console. If you are running GeoServer in a large J2EE container, you might not want your container-wide logs filled with GeoServer information. Un-checking this option will suppress most GeoServer logging, with only fatal exceptions still outputted 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 (e.g., `GeoServer.log`). Relative paths are relative to the GeoServer data directory.

## 5.2.4 GeoWebCache Settings

The GeoWebCache Settings page in the Server menu in the [Web Administration Interface](#) shows some configuration options for GeoWebCache, a tile server that comes embedded by default inside GeoServer. For more information about this embedded version, please see the section on [Caching with GeoWebCache](#).

### Enable direct WMS integration

GeoWebCache acts as a proxy between GeoServer and map client. By default, GeoWebCache has a separate endpoint from the GeoServer WMS. (See the section on [Using GeoWebCache](#) for more details.)

Enabling direct WMS integration allows WMS requests served through GeoServer to be cached as if they were received and processed by GeoWebCache. This yields the flexibility of a WMS with the speed of a tile server. See the section on [Using GeoWebCache](#) for more details about this feature.

### Disk quota

This section manages the disk usage for tiles saved with GeoWebCache.

By default, disk usage with GeoWebCache is unbounded, regardless of integration with the GeoServer WMS, so every tile served from GeoWebCache will be stored in the cache directory (typically the `gwc` directory inside the data directory). When direct WMS integration is enabled but disk quotas not enabled, every tile that is served through both the GeoServer WMS and GeoWebCache will be stored in the cache directory, which could cause disk capacity issues. Setting a disk quota allows disk usage to be constrained.

## GeoWebCache Settings

Configure the global settings for the embedded GeoWebCache

[Go to the GWC Home Page](#)

[Go to the GWC Demos Page](#)

### WMS Integration

---

☐ Enable direct WMS integration

### Disk Quota

---

☐ Enable Disk Quota limits

**Compute cache usage based on a disk block size of:**

Bytes

**Check if the cache disk quota is exceeded every:**

Seconds

(Quota never forced since server start up)

**Set maximum tile cache size:**

 Used 0.0B out of 100.0MiB available.

**When forcing disk quota limits, remove first tiles that are:**

- ☒ Least Frequently Used
- ☐ Least Recently Used

**Submit**

**Cancel**

Option	Default value	Description
<b>Enable Disk Quota limits</b>	Off	Turns on the disk quota. When disabled, the cache directory will grow unbounded. When enabled, the disk quota will be set according to the options below.
<b>Compute cache usage based on a disk block size of</b>	4096 bytes	This field should be set equal to the disk block size of the storage medium where the cache is located.
<b>Check if the cache disk quota is exceeded every</b>	10 seconds	Time interval at which the cache is polled. Smaller values (more frequent polling) will slightly increase disk activity, but larger values (less frequent polling) might cause the disk quota to be temporarily exceeded.
<b>Set maximum tile cache size</b>	100 MiB (Mebibytes)	The maximum size for the cache. When this value is exceeded and the cache is polled, tiles will be removed according to the policy choice listed below. Note that the unit options are <b>mebibytes</b> (approx. 1.05MB), <b>gibibytes</b> (approx. 1.07GB), and <b>tebibytes</b> (approx. 1.10TB).
<b>When forcing disk quota limits, remove first tiles that are</b>	Least Frequently Used	Sets the policy for tile removal when the disk quota is exceeded. Options are <b>Least Frequently Used</b> (removes tiles based on how often the tile was accessed) or <b>Least Recently Used</b> (removes tiles based on date of last access).

**Note:** See the [GeoWebCache documentation](#) for more about disk quotas.

When finished making changes, click **Submit**.

This section also shows how much disk space is being used compared to the disk quota size, as well as the last time (if any) the quota was reached.

## Links

This page contains links to the embedded GWC homepage (containing runtime statistics and status updates) and [GeoWebCache Demo page](#) where you can view all layers known to GeoWebCache and reload configuration.

### 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

## JAI Settings

Administer settings related to Java Advanced Imaging.

### Memory Capacity (0-1)

### Memory Threshold (0-1)

### Tile Threads

### Tile Threads Priority

☐ Tile Recycling☐ Image I/O Caching☒ JPEG Native Acceleration☒ PNG Native Acceleration☐ Mosaic Native Acceleration

Figure 5.7: *JAI Settings*

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.

**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 checked, Tile Recycling allows JAI to re-use already loaded tiles, with vital capability for performances.

**Image I/O Caching:** Enables/disable Image I/O Caching. When checked, indicates that raw tiles read from disk should be cached.

**Native Acceleration:** In order 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. Thus, 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 checked, enables JPEG native code, which may speed performance, but compromise security and crash protection.

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

**Mosaic Native Acceleration:** In order to reduce the overhead of handling them, very large data sets are often split into smaller chunks and then combined to create an image mosaic. An example of this can be found in aerial imagery which is usually comprised of thousands and thousands of small images at very high resolution (order of cm). Both native and JAI implementations of mosaic are provided. When checked, Mosaic Native Acceleration use the native implementation for creating mosaics.

## 5.3 Services

GeoServer serves data using standard protocols established by the [Open Geospatial Consortium](#) (OGC). Web Coverage Service (WCS) allows for requests of coverage data (rasters); Web Feature Service (WFS) allows for 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 allows for the configuration of these services as used by GeoServer.

### 5.3.1 WCS

The Web Coverage Service (WCS) offers 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**, are elements common to WCS, WFS and WMS requests.

### Service Metadata

---

☒ Enable WCS

☐ Strict CITE compliance

#### Maintainer

<http://jira.codehaus.org/secure/BrowseProject.jspa>

#### Online resource

<http://geoserver.sourceforge.net/html/index.php>

#### Title

Web Coverage Service

#### Abstract

This server implements the WCS specification 1.0 and 1.1.1, it's reference implementation of WCS 1.1.1. All layers published by this service are available on WMS also.

#### Fees

NONE

#### Access Constraints

NONE

#### Current Keywords

WCS

WMS

GEOSERVER

Remove selected

#### New Keyword

Figure 5.8: WCS Configuration page



## Service Metadata

WCS, WFS, and WMS use common metadata definitions. These nine elements are briefly 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.

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 checked, 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. searching.

## 5.3.2 WFS

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

### Service Metadata

See the section on [Service Metadata](#).

### Features

The [Open Geospatial Consortium](#) (OGC) Web Feature Service (WFS) is a standard 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:** A WFS request can potentially contain a very large dataset that is impractical to download to a client, and/or too large for a client’s renderer. Maximum number of features sets the global feature limit that a WFS GetFeature operation should generate, regardless of the actual number of query hits. 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

## Features

### Maximum number of features

☐ Return bounding box with every feature

## Service Level

☐ Basic

☐ Transactional

☒ Complete

## GML2

### SRS Style

XML ▾

## GML3

### SRS Style

URN ▾

Submit

Cancel

Figure 5.9: WFS configuration options

**Basic:** The Basic service level 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 facilitates 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 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 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).

### 5.3.3 WMS

The Web Map Service (WMS) page allows for configuration of raster rendering and SVG 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

### Raster Rendering Options

---

#### Default Interpolation

Nearest neighbor ▾

### Watermark Settings

---

☐ Enable watermark

#### Watermark URL

#### Watermark Transparency (0 - 100)

#### Watermark Position

Bottom right ▾

### SVG Options

---

#### SVG Producer

Batik ▾

☒ Enable Antialiasing

Submit

Cancel

Figure 5.10: WMS *configuration options*

are outputted. These sampling methods—Nearest Neighbor, Bilinear Interpolation and Bicubic—are available on the Default Interpolation drop-down 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 at significant costs to speed.

## Watermark Settings

Watermarking is the process of embedding an image into a map. Watermarks are usually used for branding, copyright and security measures. Configuring watermarking is done in the WMS watermark settings section.

**Enable Watermark:** Turns on watermarking. When checked, 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:** This is the 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 desired, 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.

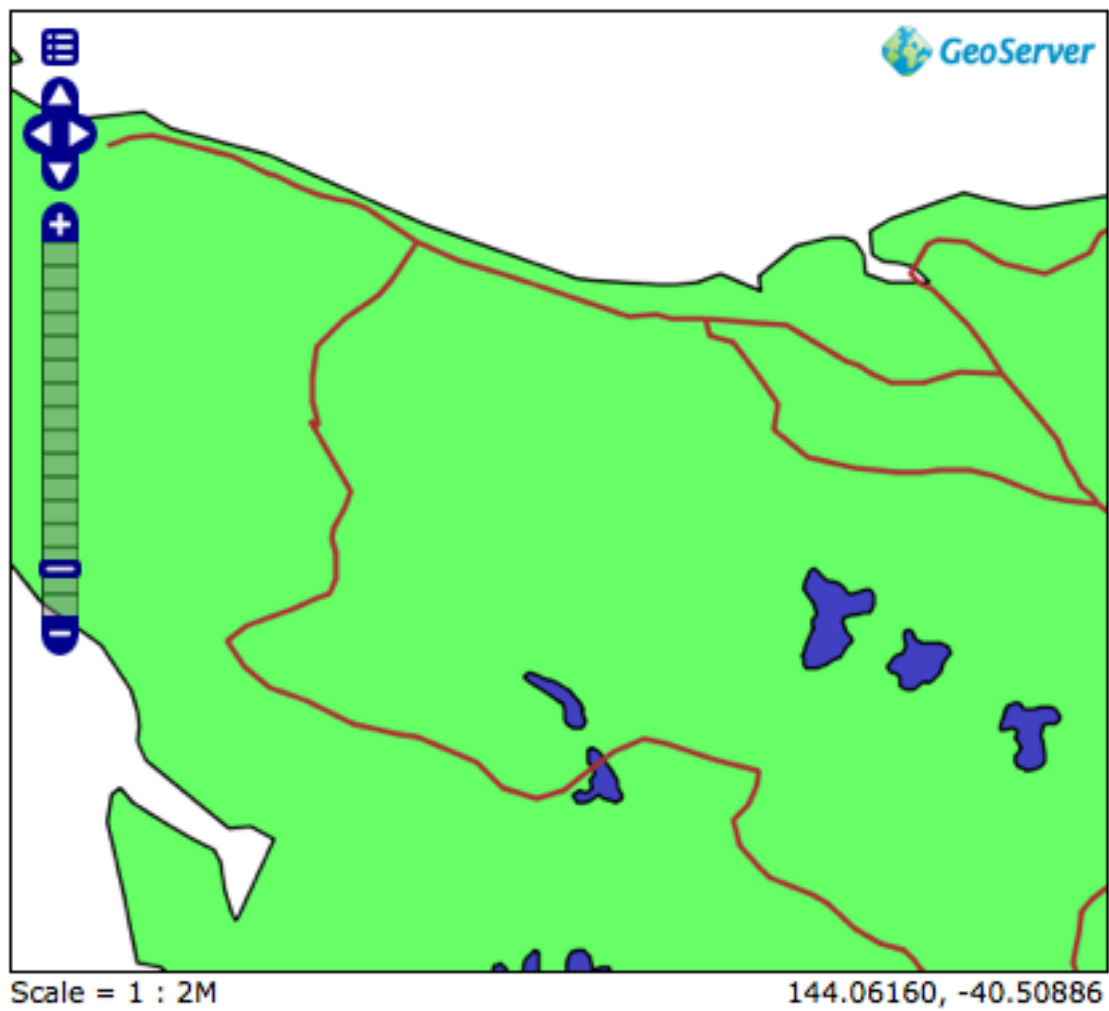


Figure 5.11: *Single tile watermark (aligned top-right, transparency=0)*

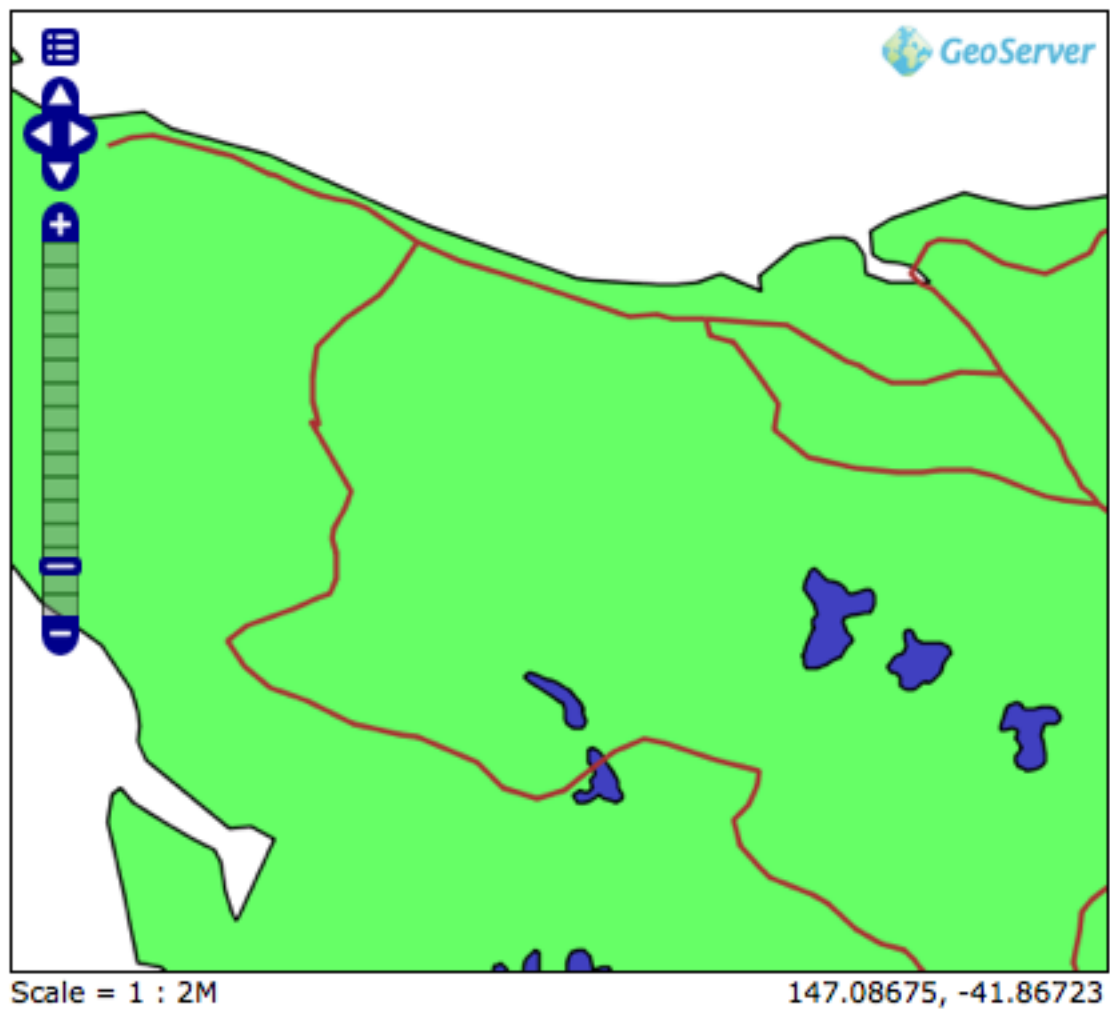


Figure 5.12: *Single tile watermark (aligned top-right, transparency=90)*

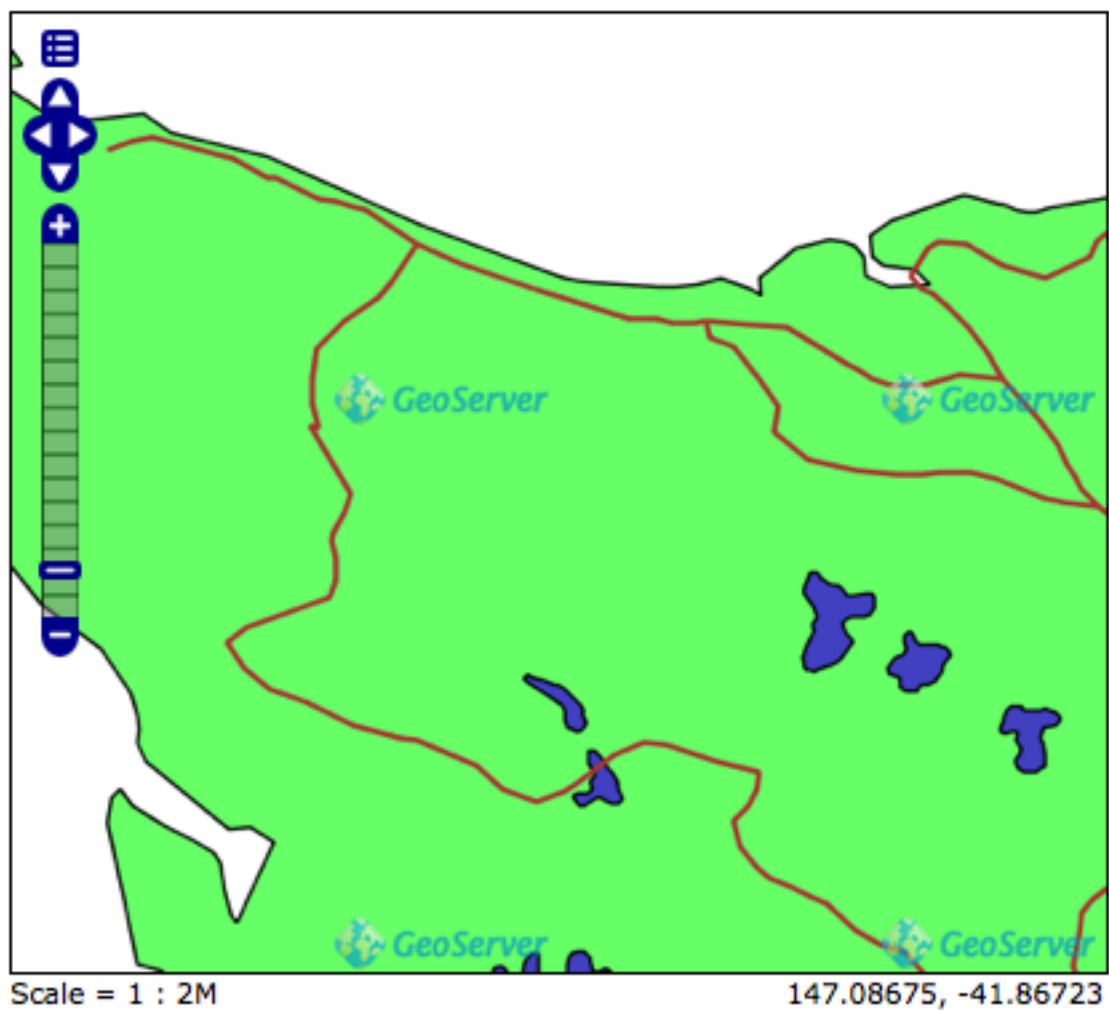


Figure 5.13: Tiled watermark (aligned top-right, transparency=90)



## SVG Options

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

### SVG Producer:

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 your maps look nicer, but will increase the size of the images returned, and will take a slight bit longer. Note that if you are overlaying the anti-aliased map on top of others it can sometimes backfire with transparencies, since it mixes with the colors behind and can create a "halo" effect.

## 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.

As seen in the example below, the data view page displays a table of indexed data.

**Layers**

Manage the layers being published by GeoServer

[+ Add a new resource](#)  
[- Remove selected resources](#)

<< < 1 2 > >> Results 1 to 10 (out of 19 items)

<input type="checkbox"/>	Type	Workspace	Store	Layer Name	Enabled?	Native SRS
<input type="checkbox"/>		nurc	arcGridSample	Arc_Sample	✓	EPSG:4326
<input type="checkbox"/>		nurc	img_sample2	Pk50095	✓	EPSG:32633
<input type="checkbox"/>		nurc	mosaic	mosaic	✓	EPSG:4326
<input type="checkbox"/>		nurc	worldImageSample	Img_Sample	✓	EPSG:4326
<input type="checkbox"/>		sf	sf	archsites	✓	EPSG:26713
<input type="checkbox"/>		sf	sf	bugsites	✓	EPSG:26713
<input type="checkbox"/>		sf	sf	restricted	⚠	EPSG:26713
<input type="checkbox"/>		sf	sf	roads	✓	EPSG:26713
<input type="checkbox"/>		sf	sf	streams	✓	EPSG:26713
<input type="checkbox"/>		sf	sfdem	sfdem	✓	EPSG:26713

<< < 1 2 > >> Results 1 to 10 (out of 19 items)

Figure 5.14: *Layers page*

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

<input type="checkbox"/> Style Name	<input type="checkbox"/> Style Name
<input type="checkbox"/> burg	<input type="checkbox"/> burg
<input type="checkbox"/> giant_polygon	<input type="checkbox"/> capitals
<input type="checkbox"/> capitals	<input type="checkbox"/> cite_lakes
<input type="checkbox"/> simple_streams	<input type="checkbox"/> concat
<input type="checkbox"/> pophatch	<input type="checkbox"/> dem
<input type="checkbox"/> restricted	<input type="checkbox"/> flags
<input type="checkbox"/> tiger_roads	<input type="checkbox"/> giant_polygon
<input type="checkbox"/> poly_landmarks	<input type="checkbox"/> grass
<input type="checkbox"/> green	<input type="checkbox"/> green
<input type="checkbox"/> rain	<input type="checkbox"/> line

Figure 5.15: On the left an unsorted column; on the right a sorted column.

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

Results 1 to 1 (out of 1 matches from 7 items)

<input type="checkbox"/> Workspace Name
<input type="checkbox"/> topp

Results 1 to 1 (out of 1 matches from 7 items)

Figure 5.16: Search results for the query “top”.

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


### 5.4.1 Workspaces

This section is for viewing and configuring workspaces. Analogous to a namespace, a workspace is a container which is used to organize 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. (Ex: topp:states) Two different layers having the same name can exist as long as they exist in different workspaces. (Ex: sf:states, topp:states).

## Workspaces

Manage GeoServer workspaces

 Add new workspace

 Remove selected workspace(s)

 Results 1 to 7 (out of 7 items)

 Search

<input type="checkbox"/> Workspace Name
<input type="checkbox"/> sf
<input type="checkbox"/> topp
<input type="checkbox"/> it.geosolutions
<input type="checkbox"/> sde
<input type="checkbox"/> nunc
<input type="checkbox"/> tiger
<input type="checkbox"/> cite


 Results 1 to 7 (out of 7 items)

Figure 5.17: Workspaces page

### Edit Workspace

In order to view details and edit a workspace, click on a workspace name.

## Workspace topp

Namespace URI

The namespace uri associated with this workspace

Figure 5.18: Workspace named “topp”

A workspace consists of a name and a Namespace URI (Uniform Resource Identifier). The workspace name has a maximum of ten characters and may not contain space. A URI is similar to URLs, except URIs need not 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.

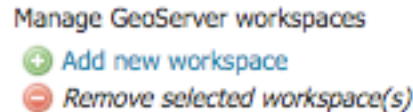


Figure 5.19: Buttons to add and remove

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







Figure 5.20: New Workspace page with example

In order to remove a workspace, click on 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 workspace(s)** button. You will be asked to confirm or cancel the deletion. Clicking **OK** will remove the workspace.

## 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 such as a table in a database, a single file (such as a shapefile), or a directory (such as Vector Product Format library). The store construct is used so 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.

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

In order to view and edit a store, click on a store name. The exact contents of this page will depend on the specific format chosen. (See the section on [Working with Data](#) for information about specific data formats.)

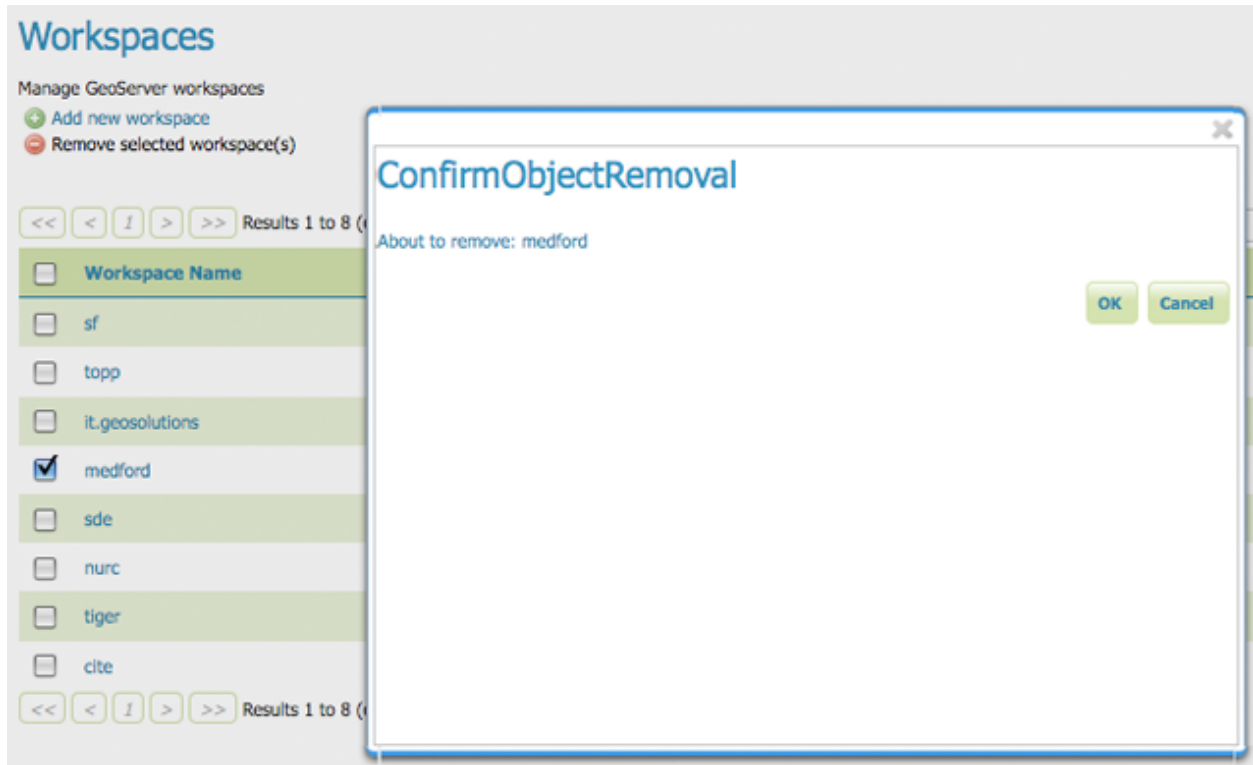


Figure 5.21: Workspace removal confirmation

## Stores

Manage the stores providing data to GeoServer

+ Add new Store

- Remove selected Stores

<< < 1 > >> Results 1 to 9 (out of 9 items)

Search

<input type="checkbox"/>	Type	Workspace	Store Name	Enabled?
<input type="checkbox"/>		nurc	arcGridSample	✓
<input type="checkbox"/>		nurc	img_sample2	✓
<input type="checkbox"/>		nurc	mosaic	✓
<input type="checkbox"/>		nurc	worldImageSample	✓
<input type="checkbox"/>		sf	sfdem	✓
<input type="checkbox"/>		sf	sf	✓
<input type="checkbox"/>		tiger	nyc	✓
<input type="checkbox"/>		topp	states_shapefile	✓
<input type="checkbox"/>		topp	taz_shapes	✓

<< < 1 > >> Results 1 to 9 (out of 9 items)

Figure 5.22: Stores View

In the example below we have the contents of the `nurc:ArcGridSample` store.

## Edit Raster Data Source

ArcGrid

Arc Grid Coverage Format

### Basic Store Info

#### Workspace

nurc

#### Data Source Name

arcGridSample

#### Description

☒ Enabled

### Connection Parameters

#### URL

file:coverages/arc\_sample/precip30min.asc

Figure 5.23: Editing a raster data store

While connection parameters will vary depending on data format, some the basic information is common across formats. The Workspace drop down 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** allows you to enable or disable 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. Please see the section on [Working with Data](#) for information on specific data formats.

## Stores

Manage the stores providing data to GeoServer









-  Add new Store
-  Remove selected Stores

Figure 5.24: Buttons to add and remove stores

## New Store chooser

### Vector Data Sources

-  Directory of spatial files - Takes a directory of spatial data files and exposes it as a data store
-  PostGIS NG - PostGIS Database
-  PostGIS NG (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 provides access to perform transactions on the server (when supported / allowed).

### 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

Figure 5.25: Choosing the data source for a new store

## Add Raster Data Source

ArcGrid

Arc Grid Coverage Format

### Basic Store Info

Workspace

cite

Data Source Name

Description

☒ Enabled

### Connection Parameters

URL

Figure 5.26: Configuration page for an ArcGrid raster data source



## Removing a Store

In order to remove a store, click on the store's corresponding check box. Multiple stores can be checked for batch removal.

**Stores**

Manage the stores providing data to GeoServer

Add new Store

Remove selected Stores

<< < 1 > >> Results 1 to 9 (out of 9 items) Search

<input type="checkbox"/>	Type	Workspace	Store Name	Enabled?
<input type="checkbox"/>		nurc	arcGridSample	
<input type="checkbox"/>		nurc	img_sample2	
<input type="checkbox"/>		nurc	mosaic	
<input checked="" type="checkbox"/>		nurc	worldImageSample	
<input type="checkbox"/>		sf	sfdem	
<input type="checkbox"/>		sf	sf	
<input checked="" type="checkbox"/>		tiger	nyc	
<input type="checkbox"/>		topp	states_shapefile	
<input type="checkbox"/>		topp	taz_shapes	

<< < 1 > >> Results 1 to 9 (out of 9 items)

Figure 5.27: Stores checked 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 the map. All layers have a source of data, called a Store.

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, i.e., the layer within the store within the workspace. The View page also displays the layer's status and native SRS.

### Layer Types

Layers are organized into two types of data, raster and vector. The difference between the two formats rests 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 earth surface features. Each cell has a distinct value, and all cells with the same value represent a specific feature.

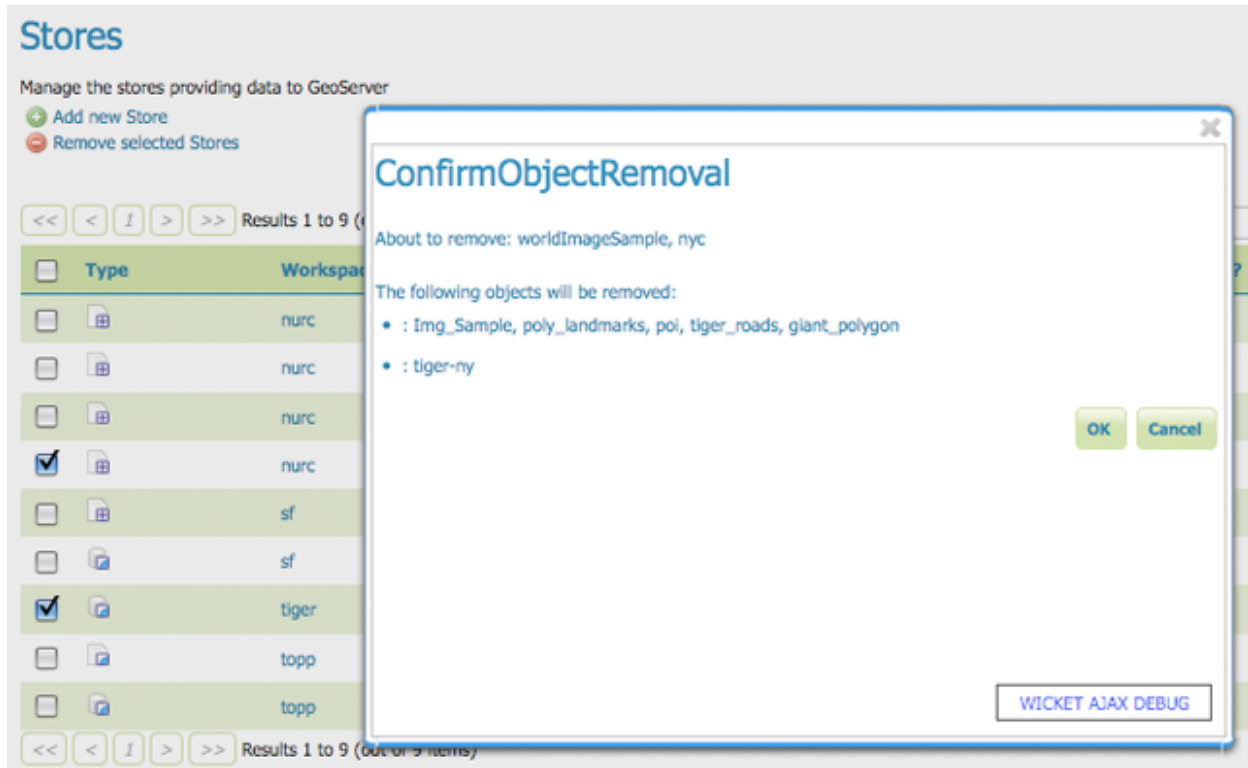


Figure 5.28: Confirm deletion of stores

## Layers

Manage the layers being published by GeoServer

- [Add a new resource](#)
- [Remove selected resources](#)



<< < 1 2 > >> Results 1 to 10 (out of 19 items)

Search

	Type	Workspace	Store	Layer Name	Enabled?	Native SRS
<input type="checkbox"/>		nurc	arcGridSample	Arc_Sample	✓	EPSG:4326
<input type="checkbox"/>		nurc	img_sample2	Pk50095	✓	EPSG:32633
<input type="checkbox"/>		nurc	mosaic	mosaic	✓	EPSG:4326
<input type="checkbox"/>		nurc	worldImageSample	Img_Sample	✓	EPSG:4326
<input type="checkbox"/>		sf	sf	archsites	✓	EPSG:26713
<input type="checkbox"/>		sf	sf	bugsites	✓	EPSG:26713
<input type="checkbox"/>		sf	sf	restricted	⚠	EPSG:26713
<input type="checkbox"/>		sf	sf	roads	✓	EPSG:26713
<input type="checkbox"/>		sf	sf	streams	✓	EPSG:26713
<input type="checkbox"/>		sf	sfdem	sfdem	✓	EPSG:26713

<< < 1 2 > >> Results 1 to 10 (out of 19 items)

Figure 5.29: Layers View

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.

## nurc:Arc\_Sample

Configure the resource and publishing information for the current layer

**Data** Publishing

---

### Basic Resource Info

**Name**

**Title**

**Abstract**

### Keywords

**Current Keywords**

WCS  
arcGridSample  
arcGridSample\_Coverage

Remove selected

**New Keyword**

Add

Figure 5.30: *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:** A human-readable description to briefly identify the layer to clients. (Required)

**Abstract:** Provides a descriptive narrative with more information about the layer.

**Keywords:** List of short words associated with the layer to aid in 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.

### Metadata links

Type	Format	URL	
FGDC	text/plain		Remove
<input type="button" value="Add link"/>			

Figure 5.31: Adding a metadata link in FGDC format

## 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 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 during both WMS and WFS requests

### Coordinate Reference Systems

#### Native SRS

EPSG:26713 [EPSG:NAD27 / UTM zone 13N...](#)

#### Declared SRS

EPSG:26713  [EPSG:NAD27 / UTM zone 13N...](#)

#### SRS handling

Force declared

Figure 5.32: Adding a metadata link in FGDC format

**Native SRS:** Refers to the projection the layer is stored in. Clicking on 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 is 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.

Bounding Boxes			
Native Bounding Box			
Min X	Min Y	Max X	Max Y
589,851.438	4,914,490.883	608,346.46	4,926,501.898
Compute from data			
Lat/Lon Bounding Box			
Min X	Min Y	Max X	Max Y
-103.873	44.377	-103.638	44.488
Compute from native bounds			

Figure 5.33: *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 2 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.

The **NotNullable** 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 **NotNullable** 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 allows for configuration of HTTP and WCS settings.

### Feature Type Details

Property	Type	Nullable	Min/Max Occurrences
the_geom	Point	true	0/1
cat	Long	true	0/1
str1	String	true	0/1

Figure 5.34: Feature Types Details for sf:archsites

## nurc:Arc\_Sample

Configure the resource and publishing information for the current layer

Data

Publishing

### Basic Settings

Name

☒ Enabled

### HTTP Settings

☐ Response Cache Headers

Cache Time (seconds)

### WCS Settings

#### Request SRS

Current Request SRS List

New Request SRS

#### Response SRS

Current Response SRS List

New Request SRS

Figure 5.35: Editing Publishing Data

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

**WMS Settings:** Sets the WMS specific publishing parameters.

## WMS Settings

### Default Style

polygon



### Additional Styles

Available Styles		Selected Styles
burg capitals cite_lakes colors dem distance giant_polygon grass green line	<div>➔</div> <div>➞</div>	

### Default Rendering Buffer

5

### Default WMS Path

- *Default style:* The 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:* the 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:* A URL to the data provider's website.
- *Logo URL:* A URL to an image that serves as a logo for the data provider.

### WMS Attribution

Attribution Text

Attribution Link

Logo URL

Logo Content Type

Logo Image Width

Logo Image Height

[Auto-detect image size and type](#)

Figure 5.36: WMS Attribution

- *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 allow users to know 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.

**WFS Settings:** For the layer, sets the maximum number of features 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 for Google Geo search crawler, to index from this particular layer. See [What is a Geo Sitemap?](#) for more information.

**KML Format Settings:** Allows for limiting 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. It takes slightly extra time to build the index upon first request.
- *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, here referred to as resource. The red minus button allows you to remove selected layers.

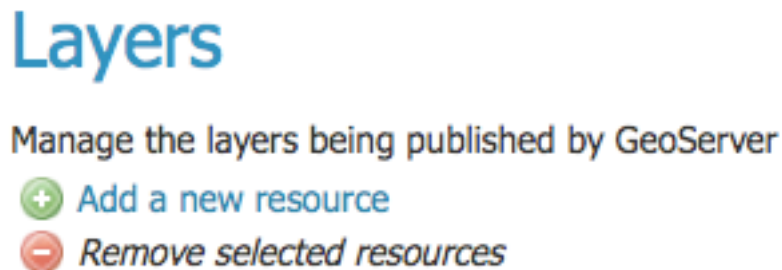


Figure 5.37: Buttons to Add or Remove a Layer

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

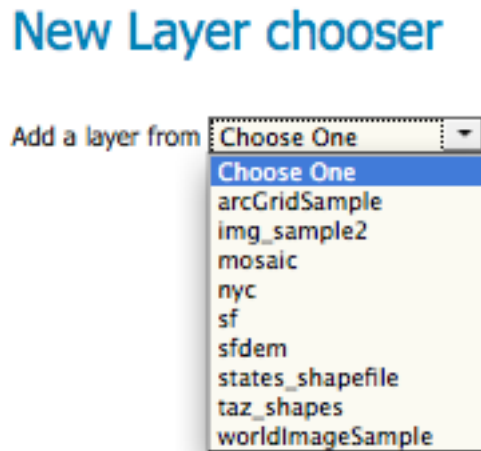


Figure 5.38: List of all currently enabled stores

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.

Upon selection of a layer name, you're redirected to a layer edit page. [Edit Layer Data](#)

## New Layer chooser

Add a layer from

Here is a list of resources contained in the 'nyc' store. Click on the layer you wish to configure

Results 0 to 0 (out of 0 items)

Layer with namespace and prefix	Published
<a href="#">giant_polygon</a>	<input checked="" type="checkbox"/>
<a href="#">poi</a>	<input checked="" type="checkbox"/>
<a href="#">poly_landmarks</a>	<input checked="" type="checkbox"/>
<a href="#">tiger_roads</a>	<input checked="" type="checkbox"/>

Results 0 to 0 (out of 0 items)

Figure 5.39: View of all layers

In order to delete a layer, click on the check box on the left side of each layer row. As shown below, multiple layers can be checked 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 enabling the checkbox in the header row.

Once layer(s) are checked, the **Remove selected resources** link is activated. Upon clicking on 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 group of layers that 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 groups act just like standard layers as far as WMS is concerned.

#### Edit Layer Group

To bring up the layer group edit page, click on a layer group name. The initial fields allow for the configuration of the name, bounds, and projection of the layer group. To automatically set bounding box, select the **Generate Bounds** button, other put in your own custom numbers. 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.

At the bottom of the page is a table listing the layers contained within the current layer group. When a layer group is processed, the layers are rendered in the order provided, so that the layer at the bottom of list will be rendered last, and thus will show on top of the other layers.

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 layer from the layer group, select the layer's button in the **Remove** column. You will not be prompted to confirm or cancel this deletion.

You can view layers group in the [Layer Preview](#) section of the web admin.


## Layers





















Manage the layers being published by GeoServer

 Add a new resource

 Remove selected resources

<< < 1 2 > >> Results 1 to 10 (out of 18 items)

 Search

<input type="checkbox"/>	Type	Workspace	Store	Layer Name	Enabled?	Native SRS
<input type="checkbox"/>		nurc	img_sample2	Pk50095		EPSG:32633
<input type="checkbox"/>		nurc	mosaic	mosaic		EPSG:4326
<input checked="" type="checkbox"/>		nurc	worldImageSample	Img_Sample		EPSG:4326
<input type="checkbox"/>		sf	sf	archsites		EPSG:26713
<input type="checkbox"/>		sf	sf	bugsites		EPSG:26713
<input checked="" type="checkbox"/>		sf	sf	restricted		EPSG:26713
<input type="checkbox"/>		sf	sf	roads		EPSG:26713
<input checked="" type="checkbox"/>		sf	sf	streams		EPSG:26713
<input type="checkbox"/>		sf	sfdem	sfdem		EPSG:26713
<input type="checkbox"/>		tiger	nyc	giant_polygon		EPSG:4326

<< < 1 2 > >> Results 1 to 10 (out of 18 items)

Figure 5.40: Layers *nurc:Img\_Sample*, *sf:restricted*, *sf:streams* selected for deletion

<< < 1 > >> Results 1 to 1:












<input checked="" type="checkbox"/>	Type	Workspace
<input checked="" type="checkbox"/>		nurc
<input checked="" type="checkbox"/>		nurc
<input checked="" type="checkbox"/>		nurc
<input checked="" type="checkbox"/>		sf
<input checked="" type="checkbox"/>		sf
<input checked="" type="checkbox"/>		sf
<input checked="" type="checkbox"/>		sf
<input checked="" type="checkbox"/>		sf
<input checked="" type="checkbox"/>		sf
<input checked="" type="checkbox"/>		tiger

Figure 5.41: All layers selected to be deleted


## Layer Groups

Define and manage layer groupings

 Add new layer group

 Remove selected layer group(s)

<< < 1 > >> Results 1 to 3 (out of 3 items)

 Search

<input type="checkbox"/>	Layer Group
<input type="checkbox"/>	spearfish
<input type="checkbox"/>	tasmania
<input type="checkbox"/>	tiger-ny

<< < 1 > >> Results 1 to 3 (out of 3 items)

Figure 5.42: Layer Groups page

## Layer group

Edit the contents of a layer groups

Name

spearfish





Bounds

Min X	Min Y	Max X	Max Y
589,425.934	4,913,959.225	609,518.672	4,928,082.95

EPSC:26713  EPSG:NAD27 / UTM zone 13N...

 Add Layer...

Layers

Layer	Style	Remove	Position
sfdem	dem		↓
streams	simple_streams		↑ ↓
roads	simple_roads		↑ ↓
restricted	restricted		↑ ↓
archsites	point		↑ ↓
bugsites	capitals		↑

<< < 1 > >> Results 1 to 6 (out of 6 items)

Figure 5.43: Layer Groups Edit page

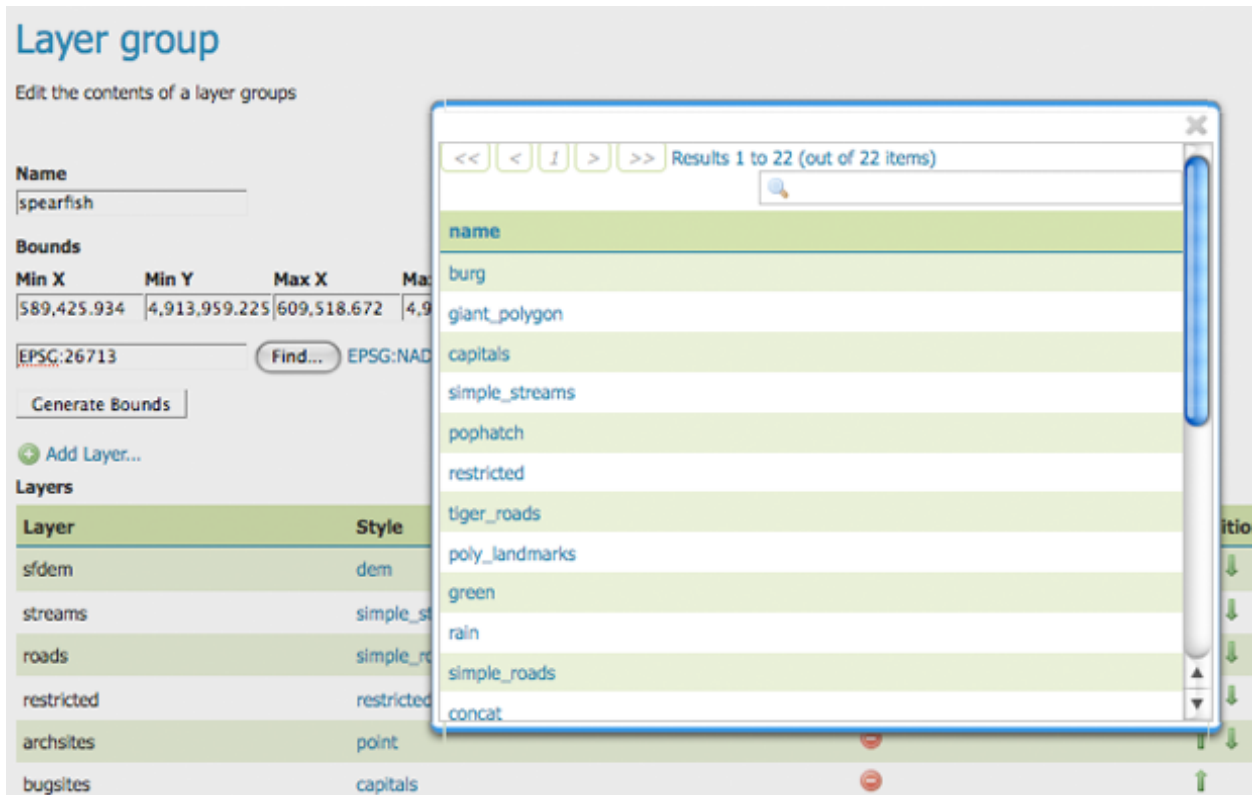


Figure 5.44: Style editing for a layer within a layer group

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

A layer can be added to the list by pressing the **Add Layer...** button at the top of the layer table. From the resulting list of layers, select the layer to be added by clicking on the layer name. This latest layer will be appended to the bottom of the layer list.

### 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.

### Remove a layer group

In order to remove a layer group, click on the check box next to the layer group. Multiple layer groups can be selected for match 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.

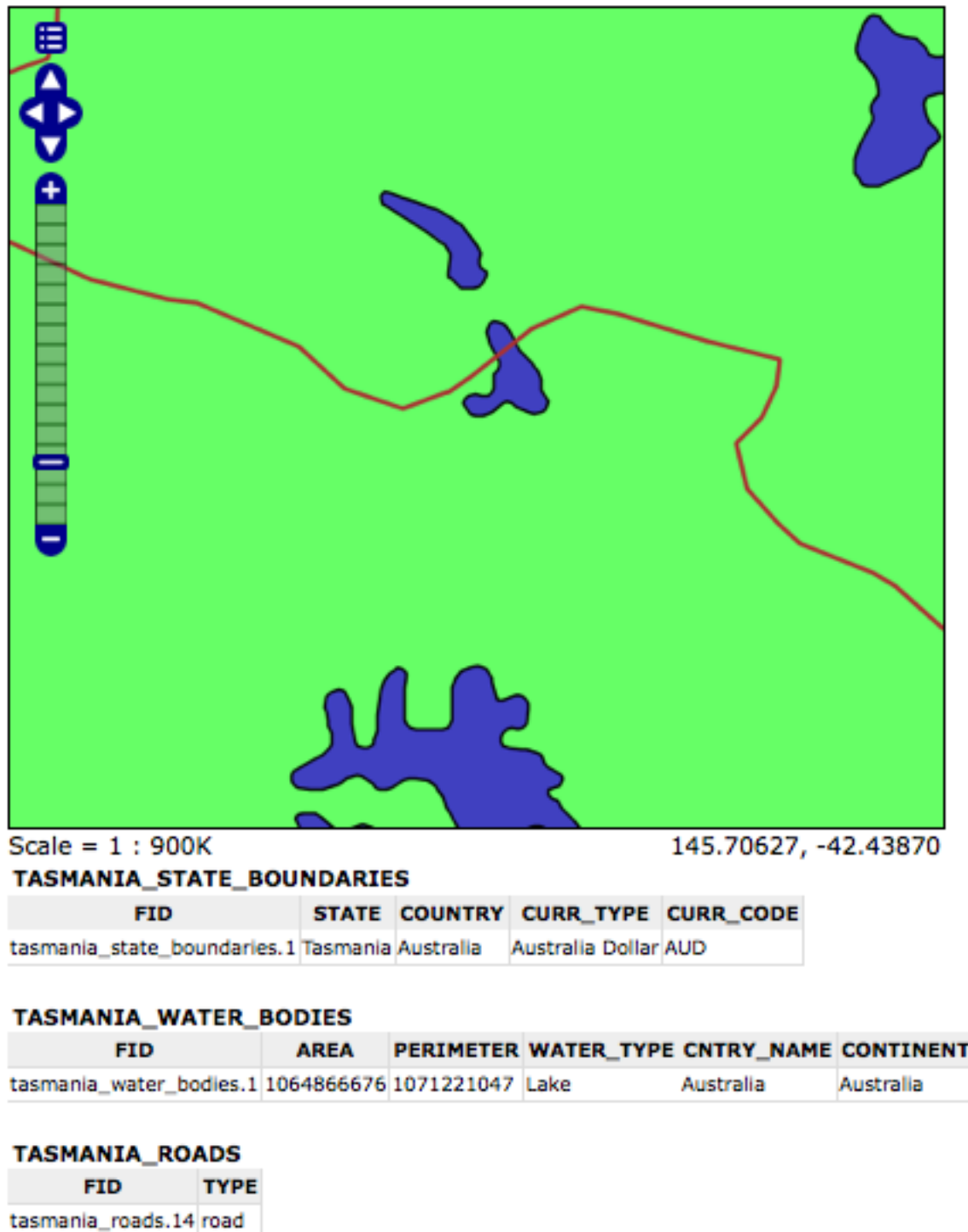


Figure 5.45: Openlayers preview of the layer group "tasmania"

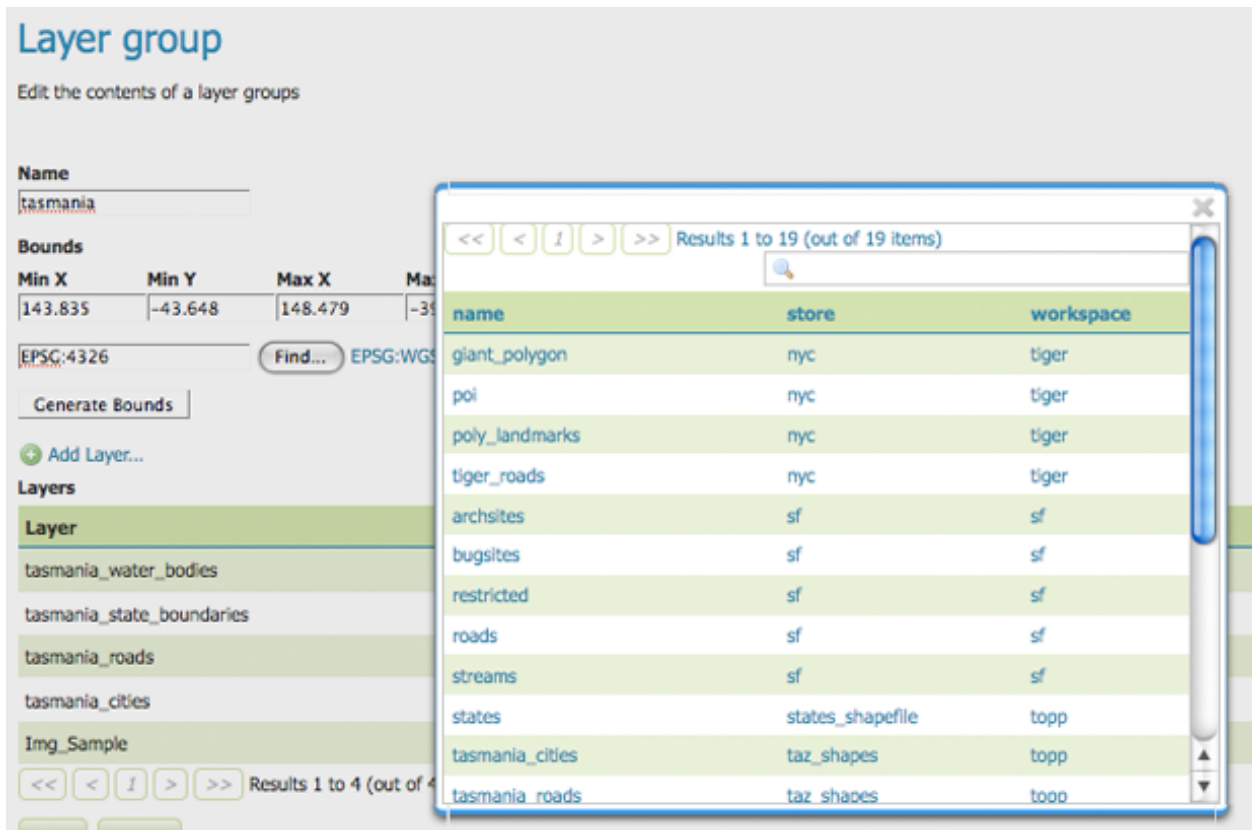


Figure 5.46: Dialog for adding a layer to a layer group

## Layer Groups

Define and manage layer groupings

Add new layer group

Remove selected layer group(s)

Figure 5.47: Buttons to add or remove a layer group

## New Layer Group

Add a new layer grouping

Name

layerABC

Submit

Cancel

Figure 5.48: New layer group dialog

## Layer group

Edit the contents of a layer groups

**Name**

**Bounds**

Min X	Min Y	Max X	Max Y
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

...

**Layers**

Layer	Style	Remove	Position
<div>&lt;&lt; &lt; &gt; &gt;&gt; Results 0 to 0 (out of 0 items)</div>			

Figure 5.49: New layer group configuration page

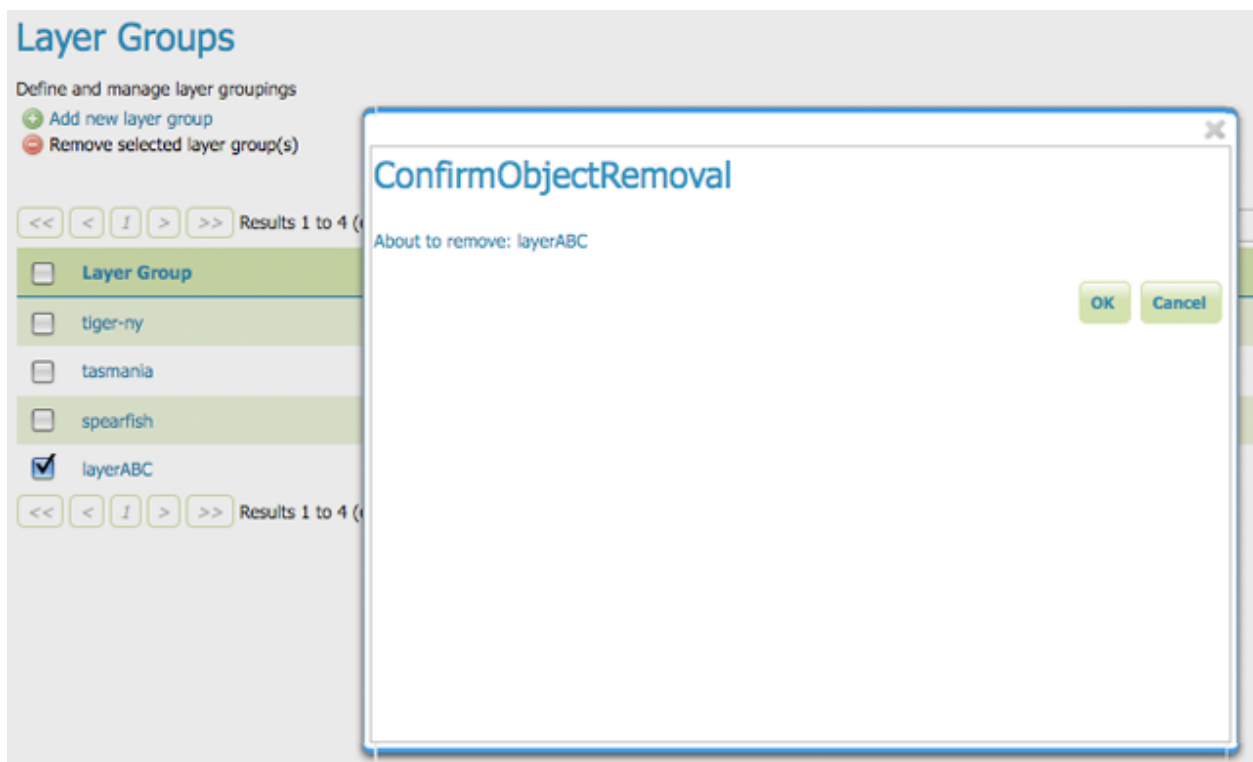


Figure 5.50: Removing a layer group



### 5.4.5 Styles

Styles are the method of rendering 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 this page, you can register or create a new style, edit an existing style, or delete remove a style.

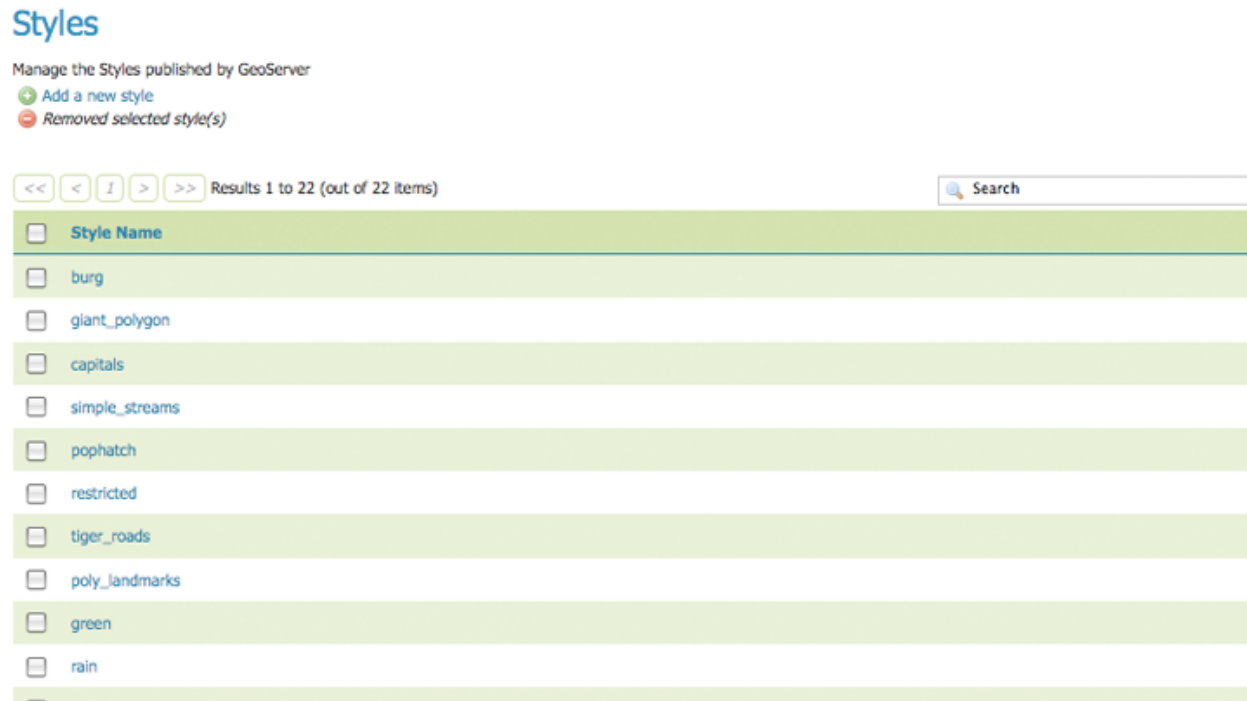


Figure 5.51: *Styles page*

### 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 alternate mode is an unformatted text editor. Check the **Toggle Editor** to switch between modes.

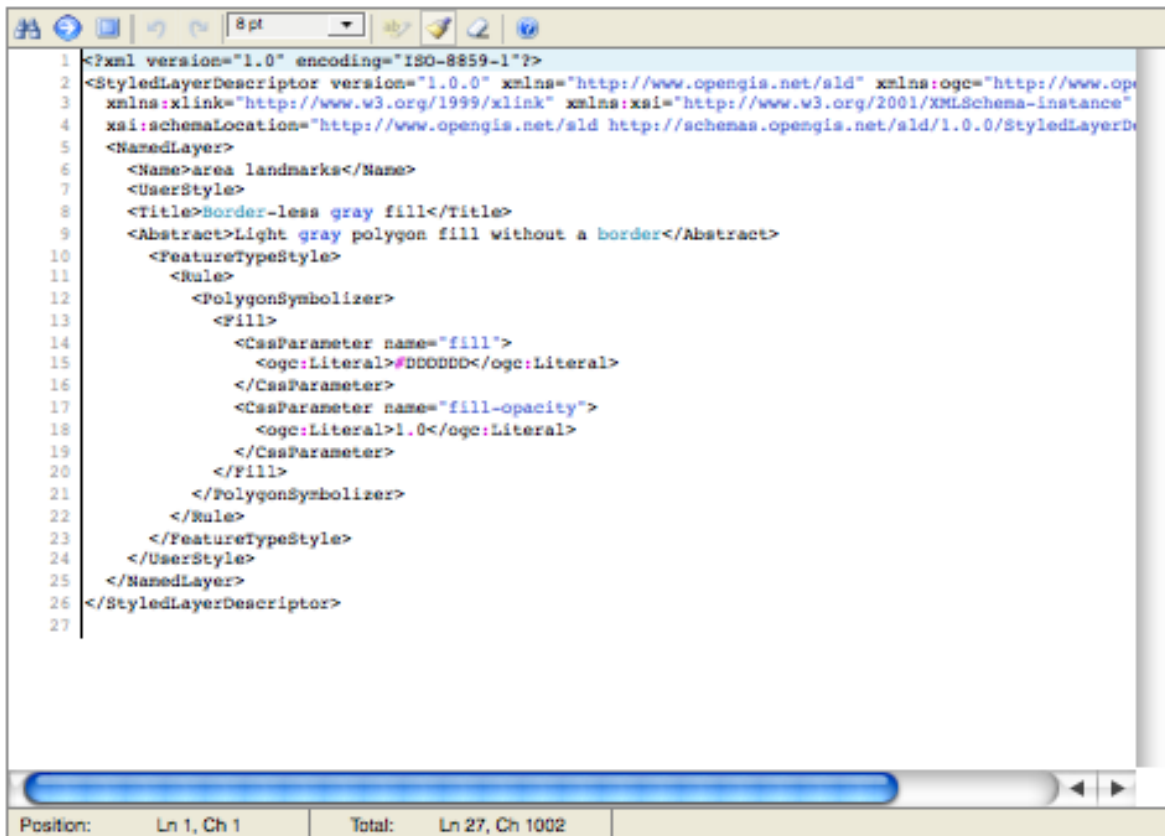
The rich editor is designed for text formatting, search and replace, line numbering, and real-time syntax highlighting. You can also switch view to full-screen mode for a larger editing area.

## Style Editor

Edit the current Styled Layer Description style. The editor can provide syntax highlight and be brought to full screen. Click on the "validate" |

Name

SLD Text



```

1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <StyledLayerDescriptor version="1.0.0" xmlns="http://www.opengis.net/sld" xmlns:ogc="http://www.op
3   xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://www.opengis.net/sld http://schemas.opengis.net/sld/1.0.0/StyledLayerD
5   <NamedLayer>
6     <Name>area_landmarks</Name>
7     <UserStyle>
8       <Title>Border-less gray fill</Title>
9       <Abstract>Light gray polygon fill without a border</Abstract>
10      <FeatureTypeStyle>
11        <Rule>
12          <PolygonSymbolizer>
13            <Fill>
14              <CssParameter name="fill">
15                <ogc:Literal>#DDDDDD</ogc:Literal>
16              </CssParameter>
17              <CssParameter name="fill-opacity">
18                <ogc:Literal>1.0</ogc:Literal>
19              </CssParameter>
20            </Fill>
21          </PolygonSymbolizer>
22        </Rule>
23      </FeatureTypeStyle>
24    </UserStyle>
25  </NamedLayer>
26 </StyledLayerDescriptor>
27

```

Position: Ln 1, Ch 1      Total: Ln 27, Ch 1002



Toggle editor



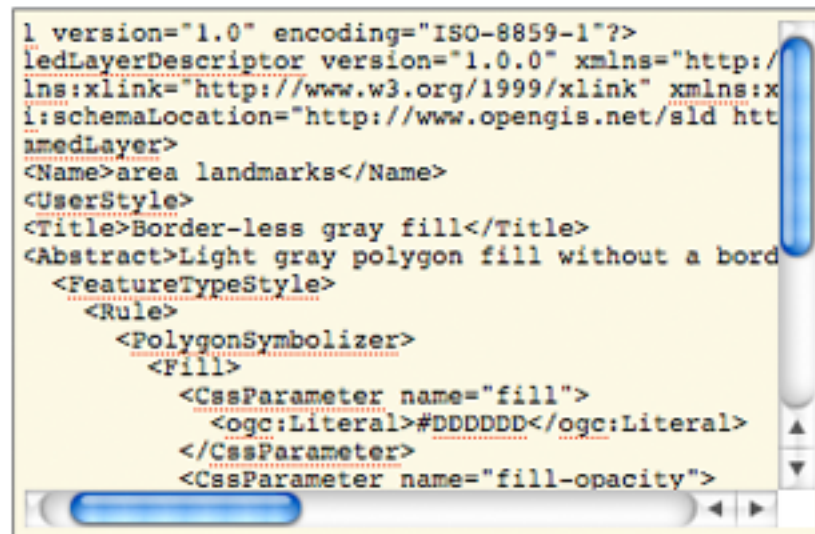

Figure 5.52: Rich text editor

## Style Editor

Edit the current Styled Layer Description style. The editor can provide syntax highlight and be brought to full the SLD schema.

Name

SLD Text



```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StyledLayerDescriptor version="1.0.0" xmlns="http://www.opengis.net/sld" 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://www.opengis.net/sld/schema/sld.xsd">
  <NamedLayer>
    <Name>area landmarks</Name>
    <UserStyle>
      <Title>Border-less gray fill</Title>
      <Abstract>Light gray polygon fill without a border</Abstract>
      <FeatureTypeStyle>
        <Rule>
          <PolygonSymbolizer>
            <Fill>
              <CssParameter name="fill">
                <ogc:Literal>#DDDDDD</ogc:Literal>
              </CssParameter>
              <CssParameter name="fill-opacity">
                <ogc:Literal>0.5</ogc:Literal>
              </CssParameter>
            </Fill>
          </PolygonSymbolizer>
        </Rule>
      </FeatureTypeStyle>
    </UserStyle>
  </NamedLayer>
</StyledLayerDescriptor>
```











Toggle editor

Validate

Submit

Cancel

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, press the **Validate** button. A message box will confirm whether the style has validation errors.

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

No validation errors.

Style Editor

Figure 5.54: No validation errors

org.xml.sax.SAXParseException: The element type "NamedLayer" must be terminated by the matching end-tag "</NamedLayer>".

Style Editor

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.

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 style will be listed.



Figure 5.56: Buttons to add or remove a style



Figure 5.57: Uploading an SLD file from your local computer

### Remove a Style

In order to remove a style, click on the check box next to the style. Multiple layer groups can be checked 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.

## 5.5 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.

### 5.5.1 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.

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.

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

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.

### 5.5.2 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.

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.

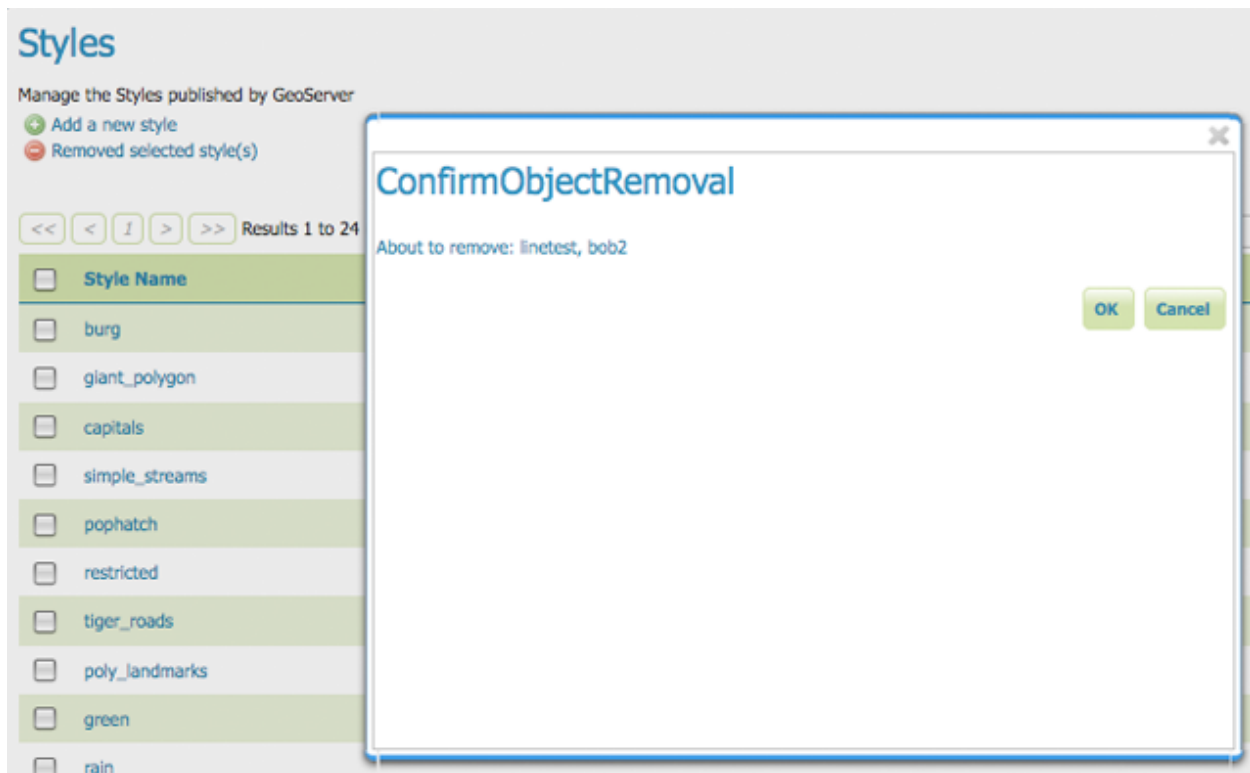


Figure 5.58: Confirmation prompt for removing styles

## GeoServer Demos

Collection of GeoServer demo applications and tools

- [Demo requests](#) Example requests for GeoServer (using the TestServlet).
- [SRS List](#) List of all SRS known to GeoServer

Figure 5.59: Demos page

## Demo requests

Example requests for GeoServer (using the TestServlet). Select a request from the drop down list, and then hit 'Change'. This will display the request Hit submit to send the request to GeoServer.

The screenshot shows the GeoServer TestServlet interface. A dropdown menu is open, displaying a list of demo requests. The menu is titled 'Request' and has a 'Choose One' button. The list of requests includes:

- WCS\_describeCoverage.xml
- WCS\_getCapabilities.xml
- WCS\_getCoverage.xml
- WFS\_describeFeatureType-1.0.xml
- WFS\_describeFeatureType-1.1.xml
- WFS\_getCapabilities-1.0.xml
- WFS\_getCapabilities-1.1.xml
- WFS\_getFeature-1.0.xml
- WFS\_getFeature-1.1.xml
- WFS\_getFeatureBBOX-1.0.xml
- WFS\_getFeatureBBOX-1.1.xml
- WFS\_getFeatureBBOX.url
- WFS\_getFeatureBetween-1.0.xml
- WFS\_getFeatureBetween-1.1.xml
- WFS\_getFeatureBetween.url
- WFS\_getFeatureBetweenCQL.url
- WFS\_getFeatureFid.url
- WFS\_getFeatureFidFilter.url
- WFS\_getFeatureIntersects-1.0.xml

Below the dropdown menu, there are input fields for 'URL', 'Body', 'User Name', and 'Password'. A 'Submit' button is located at the bottom left of the form.

Figure 5.60: *Selecting demo requests*

## Demo requests

Example requests for GeoServer (using the TestServlet). Select a request from the drop down list, and then hit 'Change'. This will display the request url (and body if an xml request). Hit submit to send the request to GeoServer.

Request:

URL:

Body:

```
<!-- A sample describe request. The schema is generated automatically by -->
<!-- GeoServer. You can modify the schema with the web interface to hide -->
<!-- and/or require certain attributes. -->
<!--
If you change the "<TypeName>" tag below to the name of another
dataset, you can see the GML Schema for that layer.
This will have all the column names and types.
The getCapabilities demo will tell you the names of all the layers!
-->
<DescribeFeatureType
  version="1.1.0"
  service="WFS"
  xmlns="http://www.opengis.net/wfs"
  xmlns:topp="http://www.openplans.org/topp"
  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">
  <TypeName>topp:states</TypeName>
</DescribeFeatureType>
```

User Name:  Password:

Figure 5.61: WFS 1.1 DescribeFeatureType sample request

Demo requests

Example requests for GeoServer (using the TestServlet). Select a request from the drop down list, and then hit 'Change'. This will display the request url (and body if an xml request). Hit submit to send the request to GeoServer.

Request:

URL:

Body:

```
<!--
If you change the "<TypeName>" tag below to the name of another
dataset, you can see the GML Schema for that layer.
This will have all the column names and types.
The getCapabilities demo will tell you the names of all the layers!
-->
<DescribeFeatureType
  version="1.1.0"
  service="WFS"
  xmlns="http://www.opengis.net/wfs"
  xmlns:topp="http://www.openplans.org/topp"
  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">
  <TypeName>topp:states</TypeName>
</DescribeFeatureType>
```

User Name:  Password:

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema elementFormDefault="qualified" targetNamespace="http://www.openplans.org/topp">
  <xsd:import namespace="http://www.opengis.net/gml" schemaLocation="http://localhost:8090/geoserver_latest/schemas/gml/3.1.1/base/gml.xsd"/>
  <xsd:complexType name="statesType">
    <xsd:complexContent>
      <xsd:extension base="gml:AbstractFeatureType">
        <xsd:sequence>
          <xsd:element maxOccurs="1" minOccurs="0" name="the_geom" nillable="true" type="gml:MultiSurfacePropertyType"/>
          <xsd:element maxOccurs="1" minOccurs="0" name="STATE_NAME" nillable="true" type="xsd:string"/>
          <xsd:element maxOccurs="1" minOccurs="0" name="STATE_FIPS" nillable="true" type="xsd:string"/>
          <xsd:element maxOccurs="1" minOccurs="0" name="SUB_REGION" nillable="true" type="xsd:string"/>
          <xsd:element maxOccurs="1" minOccurs="0" name="STATE_ABBR" nillable="true" type="xsd:string"/>
          <xsd:element maxOccurs="1" minOccurs="0" name="LAND_KM" nillable="true" type="xsd:double"/>
          <xsd:element maxOccurs="1" minOccurs="0" name="WATER_KM" nillable="true" type="xsd:double"/>
          <xsd:element maxOccurs="1" minOccurs="0" name="PERSONS" nillable="true" type="xsd:double"/>
          <xsd:element maxOccurs="1" minOccurs="0" name="FAMILIES" nillable="true" type="xsd:double"/>
          <xsd:element maxOccurs="1" minOccurs="0" name="HOUSHOLD" nillable="true" type="xsd:double"/>
          <xsd:element maxOccurs="1" minOccurs="0" name="MALE" nillable="true" type="xsd:double"/>
          <xsd:element maxOccurs="1" minOccurs="0" name="FEMALE" nillable="true" type="xsd:double"/>
          <xsd:element maxOccurs="1" minOccurs="0" name="WORKERS" nillable="true" type="xsd:double"/>
          <xsd:element maxOccurs="1" minOccurs="0" name="DRVALONE" nillable="true" type="xsd:double"/>
          <xsd:element maxOccurs="1" minOccurs="0" name="CARPOOL" nillable="true" type="xsd:double"/>
          <xsd:element maxOccurs="1" minOccurs="0" name="PUBTRANS" nillable="true" type="xsd:double"/>
          <xsd:element maxOccurs="1" minOccurs="0" name="EMPLOYED" nillable="true" type="xsd:double"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:schema>
```

Figure 5.62: XML response from a WFS 1.1 DescribeFeatureType sample request



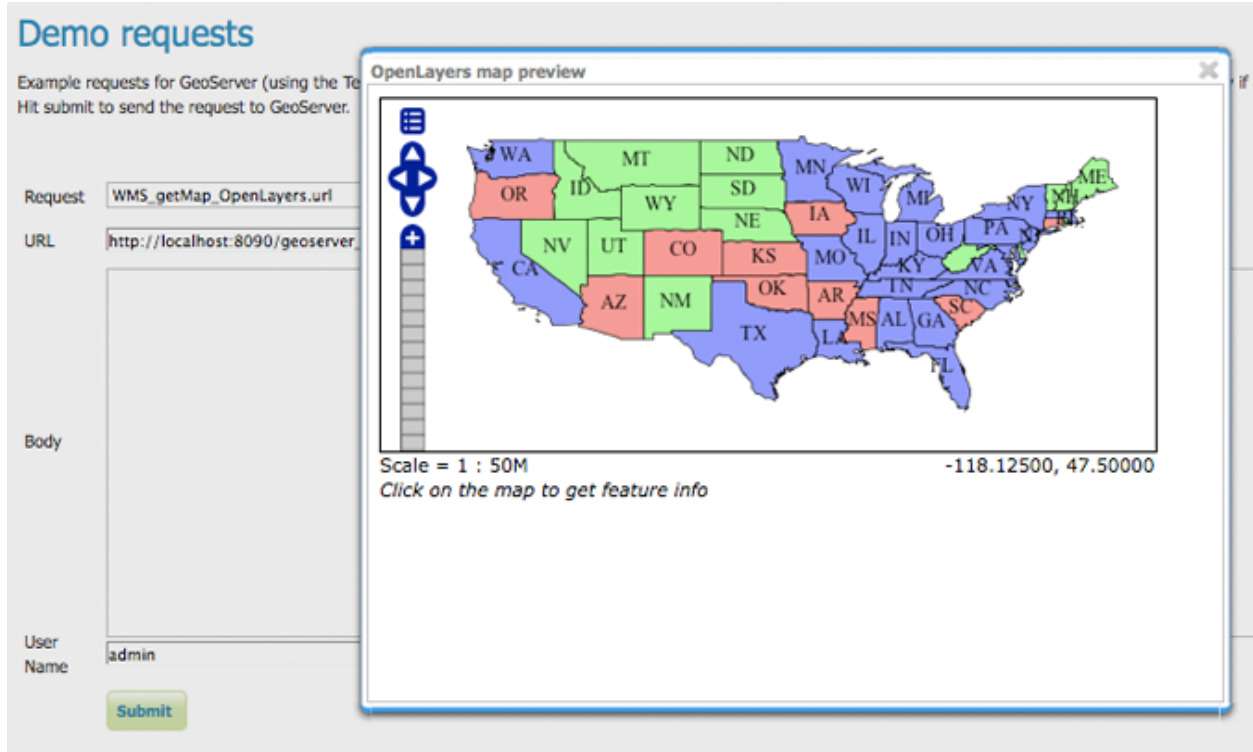


Figure 5.63: OpenLayers WMS GetMap request

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.

## 5.6 Layer Preview

This page provides layer views in various output formats. Note, a layer must be enabled in order to be previewed.

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

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 from Edit Layer Data panel.

## Demo requests

Example requests for GeoServer (using the TestServlet). Select a request from the drop down list, and then hit 'Change'. This will display the request url (and the request body) and then hit submit to send the request to GeoServer.

Request:   
URL:   
Body:   
User Name:

Results for FeatureType 'states':

```

-----
the_geom = [GEOMETRY (MultiPolygon) with 153 points]
STATE_NAME = Arizona
STATE_FIPS = 04
SUB_REGION = Mtn
STATE_ABBR = AZ
LAND_KM = 294333.462
WATER_KM = 942.772
PERSONS = 3665228.0
FAMILIES = 940106.0
HOUSHOLD = 1368843.0
MALE = 1810691.0
FEMALE = 1854537.0
WORKERS = 1358263.0
DRVALONE = 1178320.0
CARPOOL = 239083.0
PUBTRANS = 32856.0
EMPLOYED = 1603896.0
UNEMPLOY = 123902.0
SERVICE = 455896.0
MANUAL = 185109.0
P_MALE = 0.494
P_FEMALE = 0.506
SAMP_POP = 468178.0
-----

```

Figure 5.64: WMS *GetFeatureInfo* request

## SRS List

List of SRS known to GeoServer. You can choose the authority, filter based on the code and description, and gather details on each code

<< < 1 2 3 4 5 6 7 8 9 10 > >>

Results 1 to 25 (out of 3,911 items)

Search

Code	Description
2000	Anguilla 1957 / British West Indies Grid
2001	Antigua 1943 / British West Indies Grid
2002	Dominica 1945 / British West Indies Grid
2003	Grenada 1953 / British West Indies Grid
2004	Montserrat 1958 / British West Indies Grid
2005	St. Kitts 1955 / British West Indies Grid
2006	St. Lucia 1955 / British West Indies Grid
2007	St. Vincent 45 / British West Indies Grid
2008	NAD27(CGQ77) / SCoPQ zone 2
2009	NAD27(CGQ77) / SCoPQ zone 3
2010	NAD27(CGQ77) / SCoPQ zone 4
2011	NAD27(CGQ77) / SCoPQ zone 5
2012	NAD27(CGQ77) / SCoPQ zone 6
2013	NAD27(CGQ77) / SCoPQ zone 7
2014	NAD27(CGQ77) / SCoPQ zone 8

Figure 5.65: Listing of all Spatial Referencing Systems (SRS) known to GeoServer

## EPSG:2000

### Description

Anguilla 1957 / British West Indies Grid

### WKT

```
PROJCS["Anguilla 1957 / British West Indies Grid",
  GEOGCS["Anguilla 1957",
    DATUM["Anguilla 1957",
      SPHEROID["Clarke 1880 (RGS)", 6378249.145, 293.465, AUTHORITY["EPSG","7012"]],
      AUTHORITY["EPSG","6600"]],
    PRIMEM["Greenwich", 0.0, AUTHORITY["EPSG","8901"]],
    UNIT["degree", 0.017453292519943295],
    AXIS["Geodetic longitude", EAST],
    AXIS["Geodetic latitude", NORTH],
    AUTHORITY["EPSG","4600"]],
  PROJECTION["Transverse Mercator", AUTHORITY["EPSG","9807"]],
  PARAMETER["central_meridian", -62.0],
  PARAMETER["latitude_of_origin", 0.0],
  PARAMETER["scale_factor", 0.9995],
  PARAMETER["false_easting", 400000.0],
  PARAMETER["false_northing", 0.0],
  UNIT["m", 1.0],
  AXIS["Easting", EAST],
  AXIS["Northing", NORTH],
  AUTHORITY["EPSG","2000"]]
```

### Area of validity

xxx

Figure 5.66: Details for SRS EPSG:2000

## Layer Preview

List of all layers configured in GeoServer and provides previews in various formats for each.

<< < 1 > >> Results 1 to 19 (out of 19 items)












Type	Name	Title	Common Formats		All Formats
	nurc:Arc_Sample	A sample ArcGrid file	OpenLayers	KML	<input type="text" value="Select one"/>
	nurc:Pk50095	Pk50095 is a A raster file accompanied by a spatial data file	OpenLayers	KML	<input type="text" value="Select one"/>
	nurc:mosaic	Sample PNG mosaic	OpenLayers	KML	<input type="text" value="Select one"/>
	nurc:Img_Sample	North America sample imagery	OpenLayers	KML	<input type="text" value="Select one"/>
	sf:arcsites	Spearfish archeological sites	OpenLayers	KML GML	<input type="text" value="Select one"/>
	sf:bugsites	Spearfish bug locations	OpenLayers	KML GML	<input type="text" value="Select one"/>
	sf:restricted	Spearfish restricted areas	OpenLayers	KML GML	<input type="text" value="Select one"/>
	sf:roads	Spearfish roads	OpenLayers	KML GML	<input type="text" value="Select one"/>
	sf:streams	Spearfish streams	OpenLayers	KML GML	<input type="text" value="Select one"/>
	sf:sfdem	sfdem is a Tagged Image File Format with Geographic information	OpenLayers	KML	<input type="text" value="Select one"/>
	tiger:poi	Manhattan (NY) points of interest	OpenLayers	KML GML	<input type="text" value="Select one"/>

Figure 5.67: Layer's Preview Page


Type	Name	Title	Common Formats		All Formats
	nurc:Arc_Sample	A sample ArcGrid file	OpenLayers	KML	<input type="text" value="Select one"/>

Figure 5.68: Single Layer preview row

### 5.6.1 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” drop down 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 offer 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 contains 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. <a href="#">OpenLayers</a> 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.

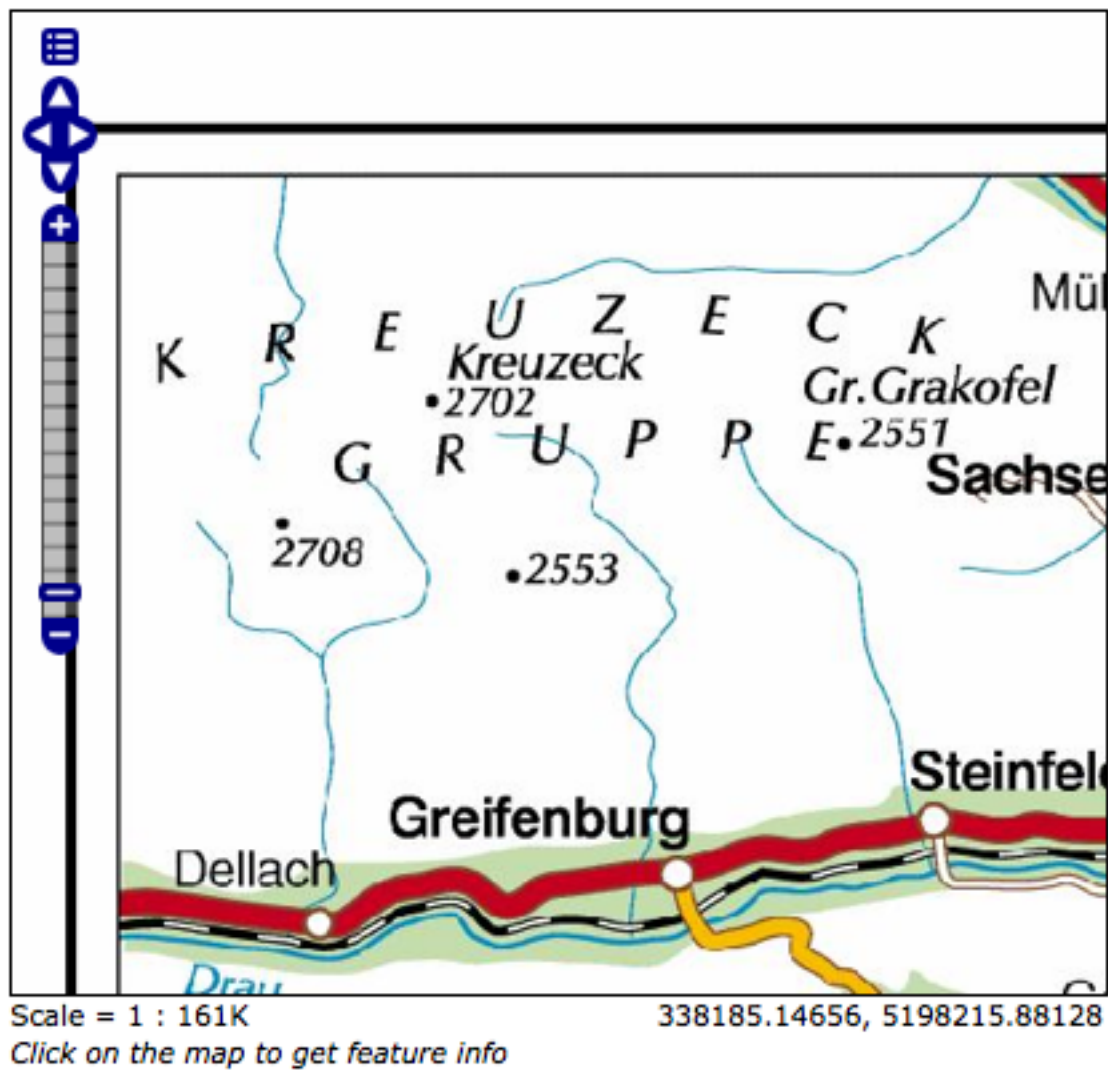


Figure 5.69: Sample Image Output-an OpenLayers preview of `nurc:Pk50095`

## Text Outputs

Format	Description
Atom-Pub	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. <a href="#">GeoRss</a> 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	<a href="#">JavaScript Object Notation</a> (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 <a href="#">Open Geospatial Consortium</a> (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" drop down 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.





---

# Working with Data

---

This section discusses the data sources that can GeoServer can read and access.

GeoServer allows the loading and serving of the following data formats by default:

- **Vector data formats**
  - Shapefiles (including directories of shapefiles)
  - PostGIS databases (with or without JNDI)
  - External WFS layers
  - Java Properties files
- **Raster data formats**
  - ArcGrid
  - GeoTIFF
  - Gtopo30
  - ImageMosaic
  - WorldImage
- **Other data formats**
  - External WMS layers

Other data sources require the use of GeoServer extensions, extra downloads that add functionality to GeoServer. These extensions are always available on the [GeoServer download page](#).

**Warning:** If an extension is required to load the data source, make sure to match the version of the extension to the version of the GeoServer instance!

## 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*.

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: <code>topp:states</code> ).
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 <code>file:C:\Data\shapefile.shp</code> ) or a path relative to the data directory (such as <code>file:data/shapefile.shp</code> ).
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 <code>.dbf</code> file.
memory mapped buffer	Enables the use of memory mapped I/O.

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 PostGIS

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

## New Vector Data Source

Shapefile

ESRI(tm) Shapefiles (\*.shp)

### Basic Store Info

#### Workspace

cite ▼

#### Data Source Name

#### Description

☒ Enabled

### Connection Parameters

#### URL

#### namespace

cite: <http://www.opengeospatial.net/cite> ▼

☐ create spatial index

#### charset

☐ memory mapped buffer

Save

Cancel

Figure 6.1: Adding a shapefile as a store

### 6.2.1 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*.

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

# New Vector Data Source

PostGIS NG  
PostGIS Database

## Basic Store Info

### Workspace

cite ▼

### Data Source Name

### Description

☒ Enabled

## Connection Parameters

### dbtype

postgisng

### host

localhost

### port

5432

### database

### schema

public

### user

### passwd

### namespace

cite: <http://www.opengeospatial.net/cite> ▼

## New Vector Data Source

PostGIS NG (JNDI)  
PostGIS Database (JNDI)

### Basic Store Info

#### Workspace

cite ▼

#### Data Source Name

#### Description

☒ Enabled

### Connection Parameters

#### dbtype

postgisng

#### jndiReferenceName

java:comp/env/jdbc/mydatabase

#### schema

#### namespace

cite: <http://www.opengeospatial.net/cite> ▼

Save

Cancel

Figure 6.3: Adding a PostGIS database (using JNDI)

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

When finished, click **Save**.

## 6.2.2 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.

## 6.2.3 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.

## 6.2.4 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' );
```

## 6.2.5 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.

## 6.2.6 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.

## 6.3 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.

### 6.3.1 Adding a directory

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

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

When finished, click **Save**.



## New Vector Data Source

Directory of spatial files

Takes a directory of spatial data files and exposes it as a data store

### Basic Store Info

#### Workspace

cite ▼

#### Data Source Name

#### Description

☒ Enabled

### Connection Parameters

#### URL

file:data/example.extension

#### namespace

cite: <http://www.opengeospatial.net/cite> ▼

Save

Cancel

Figure 6.4: Adding a directory of spatial files as a store

### 6.3.2 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.

## 6.4 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.

### 6.4.1 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
<b>Workspace</b>	Name of the workspace to contain the store. This will also be the prefix of all of the layer names created from the store.
<b>Data Source Name</b>	Name of the store as known to GeoServer.
<b>Description</b>	Description of the store.
<b>Enabled</b>	Enables the store. If disabled, no data from the external WFS will be served.
<b>GET_CAPABILITIES_URL</b>	URL to access the capabilities document of the remote WFS.
<b>PROTOCOL</b>	When checked, connects with POST, otherwise uses GET.
<b>USERNAME</b>	The user name to connect to the external WFS.
<b>PASSWORD</b>	The password associated with the above user name.
<b>ENCODING</b>	The character encoding of the XML requests sent to the server. Defaults to UTF-8.
<b>TIMEOUT</b>	Time (in milliseconds) before timing out. Default is 3000.
<b>BUFFER_SIZE</b>	Specifies a buffer size (in number of features). Default is 10 features.
<b>TRY_GZIP</b>	Specifies that the server should transfer data using compressed HTTP if supported by the server.
<b>LENIENT</b>	When checked, will try to render features that don't match the appropriate schema. Errors will be logged.
<b>MAXFEATURES</b>	Maximum amount of features to retrieve for each featuretype. Default is no limit.

When finished, click **Save**.

### 6.4.2 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.

### 6.4.3 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.

## New Vector Data Source

### Web Feature Server

The WFSDataStore represents a connection to a Web Feature Server. This connection provides access to the Features published by the server, and the ability to perform transactions on the server (when supported / allowed).

### Basic Store Info

#### Workspace

cite ▼

#### Data Source Name

#### Description

☒ Enabled

### Connection Parameters

#### WFSDataStoreFactory:GET\_CAPABILITIES\_URL

☐ WFSDataStoreFactory:PROTOCOL

#### WFSDataStoreFactory:USERNAME

#### WFSDataStoreFactory:PASSWORD

#### WFSDataStoreFactory:ENCODING

#### WFSDataStoreFactory:TIMEOUT

#### WFSDataStoreFactory:BUFFER\_SIZE

☐ WFSDataStoreFactory:TRY\_GZIP

☐ WFSDataStoreFactory:LENIENT

#### WFSDataStoreFactory:MAXFEATURES

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.

## 6.5 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.

### 6.5.1 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**.

#### Other Data Sources

 **WMS** - Cascades a remote Web Map Service

Figure 6.6: Adding an external WMS as a store

## New WMS Connection

Edit the connection to a remote WMS Connection

### Basic Store Info

**Workspace \***

cite ▼

**Data Source Name \***

**Description**

☒ Enabled

**Capabilities URL \***

Save

Cancel

Figure 6.7: *Configuring a new external WMS store*

<b>Option</b>	<b>Description</b>
<b>Workspace</b>	Name of the workspace to contain the store. This will also be the prefix of all of the layer names published from the store. <b>The workspace name on the remote WMS is not cascaded.</b>
<b>Data Source Name</b>	Name of the store as known to GeoServer.
<b>Description</b>	Description of the store.
<b>Enabled</b>	Enables the store. If disabled, no data from the remote WMS will be served.
<b>Capabilities URL</b>	The full URL to access the capabilities document of the remote WMS.

When finished, click **Save**.

## 6.5.2 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.

## 6.5.3 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.** Requires the optional [RESTful Configuration](#) extension. See the [REST Configuration API Reference](#) for more information about the GeoServer REST interface.

## 6.5.4 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.

## 6.6 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.

### 6.6.1 Adding a Properties data store

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

## Vector Data Sources



**Properties** - Allows access to Java Property files containing Feature information

Figure 6.8: *Properties in the list of vector data stores*

### 6.6.2 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 <code>/usr/local/geoserver/data/ex</code>

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.

### 6.6.3 Property file format

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

## New Vector Data Source

---

### Properties

Allows access to Java Property files containing Feature information

---

### Basic Store Info

#### Workspace

cite ▼

#### Data Source Name

#### Description

☒ Enabled

---

### Connection Parameters

#### directory

#### namespace

<http://www.opengeospatial.net/cite>

Save

Cancel

Figure 6.9: *Configuring a Properties data store*



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

```
_=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.

## 6.7 ArcGrid

ArcGrid is a coverage file format created by ESRI.

### 6.7.1 Adding an ArcGrid data store

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

## Raster Data Sources



Figure 6.10: *ArcGrid in the list of raster data stores*

## 6.7.2 Configuring a ArcGrid data store

Option	Description
Workspace	
Data Source Name	
Description	
Enabled	
URL	

## 6.8 GeoTIFF

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

### 6.8.1 Adding a GeoTIFF data store

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

### 6.8.2 Configuring a GeoTIFF data store

Option	Description
Workspace	
Data Source Name	
Description	
Enabled	
URL	

## 6.9 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>

### 6.9.1 Adding a GTOPO30 data store

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

### 6.9.2 Configuring a GTOPO30 data store

Option	Description
Workspace	
Data Source Name	
Description	
Enabled	
URL	

## Add Raster Data Source

Description

ArcGrid

Arc Grid Coverage Format

### Basic Store Info

#### Workspace

cite

#### Data Source Name

#### Description

☒ Enabled

### Connection Parameters

#### URL

file:data/example.extension

Save

Cancel

Figure 6.11: *Configuring an ArcGrid data store*

## Raster Data Sources



**GeoTIFF** - Tagged Image File Format with Geographic information

Figure 6.12: *GeoTIFF in the list of raster data stores*

## Add Raster Data Source

### Description

GeoTIFF

Tagged Image File Format with Geographic information

### Basic Store Info

#### Workspace

cite ▼

#### Data Source Name

#### Description

☒ Enabled

### Connection Parameters

#### URL

Save

Cancel

Figure 6.13: Configuring a GeoTIFF data store

## Raster Data Sources

 **Gtopo30** - Gtopo30 Coverage Format

Figure 6.14: GTOPO30 in the list of raster data stores

## Add Raster Data Source

Description

Gtopo30

Gtopo30 Coverage Format

### Basic Store Info

#### Workspace

cite ▼

#### Data Source Name

#### Description

☒ Enabled

### Connection Parameters

#### URL

Save

Cancel

Figure 6.15: *Configuring a GTOPO30 data store*

## 6.10 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 ImageMosaic is the tutorial: [Using the ImageMosaic plugin](#).

### 6.10.1 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 6.16: *ImageMosaic in the list of raster data stores*

### 6.10.2 Configuring an ImageMosaic data store

Option	Description
Workspace	
Data Source Name	
Description	
Enabled	
URL	

## 6.11 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.

### 6.11.1 Adding a WorldImage data store

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

# Add Raster Data Source

Description

ImageMosaic  
Image mosaicking plugin

## Basic Store Info

### Workspace

cite ▼

### Data Source Name

### Description

☒ Enabled

## Connection Parameters

### URL

Save

Cancel

Figure 6.17: *Configuring an ImageMosaic data store*

## Raster Data Sources

 **WorldImage** - A raster file accompanied by a spatial data file

Figure 6.18: *WorldImage in the list of raster data stores*

## Add Raster Data Source

Description

---

WorldImage

A raster file accompanied by a spatial data file

### Basic Store Info

---

#### Workspace

▼

#### Data Source Name

#### Description

☒ Enabled

### Connection Parameters

---

#### URL

**Save**

**Cancel**

Figure 6.19: *Configuring a WorldImage data store*



### 6.11.2 Configuring a WorldImage data store

Option	Description
Workspace	
Data Source Name	
Description	
Enabled	
URL	

## 6.12 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**.

### 6.12.1 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.

### 6.12.2 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

### 6.12.3 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
<code>jsde_sdk.jar</code>	Also known as <code>jsde##_sdk.jar</code> where ## is the version number, such as 92 for ArcSDE version 9.2
<code>jpe_sdk.jar</code>	Also known as <code>jpe##_sdk.jar</code> 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 installation 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.

#### 6.12.4 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](#).

### Vector Data Sources

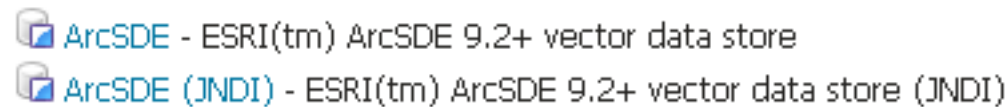


Figure 6.20: ArcSDE in the list of data sources

#### 6.12.5 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	Re-quired?	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 <a href="#">Database Connection Pooling</a> section for details.
pool.maxConnections	No	Connection pool configuration parameters. See the <a href="#">Database Connection Pooling</a> section for details.
pool.timeOut	No	Connection pool configuration parameters. See the <a href="#">Database Connection Pooling</a> section for details.

# New Vector Data Source

---

ArcSDE

ESRI(tm) ArcSDE 9.2+ vector data store

## Basic Store Info

---

### Workspace

▼

### Data Source Name

### Description

☒ Enabled

## Connection Parameters

---

### namespace

<http://www.opengeospatial.net/cite>

### Database type

### Server name or IP address

### Port

### Instance name

### User

### Password

### Initial number of connections

### Maximum number of connections

You may now add featurtypes as you would normally do, by navigating to the **New Layer** page, accessed from the [Layers](#) page in the *Web Administration Interface*.

### 6.12.6 Adding an ArcSDE vector data store with JNDI

### 6.12.7 Configuring an ArcSDE vector data store with JNDI

### 6.12.8 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*.

## Raster Data Sources



Figure 6.22: ArcSDE Raster in the list of data sources

### 6.12.9 Configuring an ArcSDE raster coveragestore

The next page contains configuration options for the ArcSDE instance. Fill out the form, then click **Save**.

Option	Re-quired?	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*.

## 6.13 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.

# Add Raster Data Source

Description

ArcSDE Raster

ArcSDE Raster Format

## Basic Store Info

### Workspace

cite ▼

### Data Source Name

### Description

☒ Enabled

## Connection Parameters

Same connection parameters as:

Choose One ▼

### Server

### Port

5151

### Database

### User Name

### Password

Choose One ▼

Refresh

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

### 6.13.1 Supported versions

Currently GML version 2 is supported.

### 6.13.2 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.

### 6.13.3 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.

#### Vector Data Sources



GML - Read only data store for validating gml 2.x data

Figure 6.24: GML in the list of vector data stores

### 6.13.4 Configuring a GML data store

## 6.14 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>

## New Vector Data Source

---

GML

Read only data store for validating gml 2.x data

### Basic Store Info

---

#### Workspace

cite ▼

#### Data Source Name

#### Description

☒ Enabled

### Connection Parameters

---

#### URL

Save

Cancel

Figure 6.25: *Configuring a GML data store*



### 6.14.1 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.

### 6.14.2 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.

#### Vector Data Sources

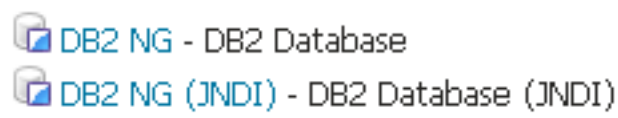


Figure 6.26: DB2 in the list of raster data stores

### 6.14.3 Configuring a DB2 data store

### 6.14.4 Configuring a DB2 data store with JNDI

### 6.14.5 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.

## New Vector Data Source

---

DB2 NG

DB2 Database

### Basic Store Info

---

#### Workspace

cite ▼

#### Data Source Name

#### Description

☒ Enabled

### Connection Parameters

---

#### dbtype

db2

#### host

localhost

#### port

#### database

#### schema

#### user

#### passwd

#### namespace

<http://www.opengeospatial.net/cite>

#### max connections

## 6.15 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.

### 6.15.1 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.

### 6.15.2 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 6.28: H2 in the list of vector data stores

### 6.15.3 Configuring an H2 data store

### 6.15.4 Configuring an H2 data store with JNDI

## 6.16 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.

# New Vector Data Source

H2

H2 Embedded Database

## Basic Store Info

### Workspace

cite ▼

### Data Source Name

### Description

☒ Enabled

## Connection Parameters

### dbtype

h2

### database

### namespace

<http://www.opengeospatial.net/cite>

### max connections

10

### min connections

1

### fetch size

1000

### Connection timeout

20

 Associations

### 6.16.1 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.

### 6.16.2 Adding a MySQL database

Once the extension is properly installed MySQL will show up as an option when creating a new data store.

## Vector Data Sources



Figure 6.30: MySQL in the list of data sources

### 6.16.3 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	Connection pool configuration parameters. See the <a href="#">Database Connection Pooling</a> section for details.
min connections	
validate connections	

## 6.17 Pregeneralized Features

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

### 6.17.1 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.

# New Vector Data Source

---

MySQL

MySQL Database

## Basic Store Info

---

### Workspace

cite ▼

### Data Source Name

### Description

☒ Enabled

## Connection Parameters

---

### dbtype

mysql

### host

localhost

### port

3306

### database

### user

### passwd

### max connections

10

### min connections

4

☐ validate connections

### 6.17.2 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.

## Vector Data Sources



Figure 6.32: Generalized Data Store in the list of vector data stores

### 6.17.3 Configuring a Pregeneralized Features data store

For a detailed description, look at the [Tutorial](#)

## 6.18 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 extensions of Oracle.

### 6.18.1 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.

### 6.18.2 Adding an Oracle datastore

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

## New Vector Data Source

Generalizing data store

Data store supporting generalized geometries

### Basic Store Info

#### Workspace

cite ▼

#### Data Source Name

#### Description

☒ Enabled

### Connection Parameters

#### RepositoryClassName

#### GeneralizationInfosProviderClassName

#### GeneralizationInfosProviderParam

#### namespace

<http://www.opengeospatial.net/cite>

Save

Cancel

Figure 6.33: Configuring a Pregeneralized Features data store



## Vector Data Sources

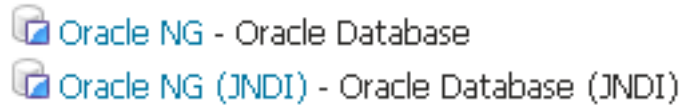


Figure 6.34: Oracle in the list of data sources

### 6.18.3 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.
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 oracle 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 the <a href="#">Database Connection Pooling</a> section for details.
Loose bbox	Controls how bounding box comparisons are made against geometries in the database. See the <a href="#">Using loose bounding box</a> section below.

#### Using loose bounding box

When the `loose bbox` option is set, only the bounding box of a geometry is used. This results in a significant performance gain. The downside is that some geometries may be considered inside of a bounding box when they are technically not.

If the primary use of the database is through [Web Map Service](#) this flag can be set safely since a loss of some accuracy is usually acceptable. However if using [Web Feature Service](#) and making use of BBOX filtering capabilities, this flag should not be set.

### 6.18.4 Configuring an Oracle database with JNDI

See [Setting up a JNDI connection pool with Tomcat](#) for a step by step guide on setting up an Oracle JNDI connection.

## 6.19 Microsoft SQL Server

**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.

# New Vector Data Source

Oracle NG  
Oracle Database

---

## Basic Store Info

### Workspace

▼

### Data Source Name

### Description

☒ Enabled

---

## Connection Parameters

### dbtype

### host

### port

### database

### schema

### user

### passwd

### namespace

<http://www.opengeospatial.net/cite>

### max connections

Microsoft's [SQL Server](#) is a relational database with spatial functionality.

### 6.19.1 Supported versions

The extension supports SQL Server 2008.

### 6.19.2 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 download page](#).
2. Download using the [Download SQL Server JDBC Driver 3.0](#) link.
3. Accept the license and download the appropriate archive for your operating system.
4. Extract the contents of the archive
5. Copy the file `sqljdbc4.jar` to the `WEB-INF/lib` directory of the GeoServer installation.
6. For GeoServer installed on Windows, copy `\x86\sqljdbc_auth.dll` and `\x86\sqljdbc_xa.dll` to `C:\Windows\System32`

### 6.19.3 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.

## Vector Data Sources



Figure 6.36: SQL Server in the list of vector data sources

### 6.19.4 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 <a href="#">note</a> below.
database	The name of the database to connect to.
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	Connection pool configuration parameters. See the <a href="#">Database Connection Pooling</a> section for details.
min connections	

#### 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
```

### 6.19.5 Adding a SQL Server database with JNDI

### 6.19.6 Configuring a SQL Server database with JNDI

## 6.20 VPF

**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](#).

### 6.20.1 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.

### 6.20.2 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.

# New Vector Data Source

Microsoft SQL Server

Microsoft SQL Server

## Basic Store Info

### Workspace

cite ▼

### Data Source Name

### Description

☒ Enabled

## Connection Parameters

### dbtype

sqlserver

### host

localhost

### port

### database

### schema

### user

### passwd

### namespace

<http://www.opengeospatial.net/cite>

### max connections

## Vector Data Sources

 **Vector Product Format Library** - Vector Product Format Library data store implementation.

Figure 6.38: VPF in the list of new data sources

### 6.20.3 Configuring a VPF data store

## 6.21 GDAL Image Formats

GeoServer can leverage the [ImageIO-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](http://www.gdal.org/frmt_dted.html)
- EHdr, ESRI .hdr Labelled: <[http://www.gdal.org/frmt\\_various.html#EHdr](http://www.gdal.org/frmt_various.html#EHdr)>
- ENVI, ENVI .hdr Labelled Raster: <[http://www.gdal.org/frmt\\_various.html#ENVI](http://www.gdal.org/frmt_various.html#ENVI)>
- HFA, Erdas Imagine (.img): <[http://www.gdal.org/frmt\\_hfa.html](http://www.gdal.org/frmt_hfa.html)>
- JP2MrSID, JPEG2000 (.jp2, .j2k): <[http://www.gdal.org/frmt\\_jp2mrsid.html](http://www.gdal.org/frmt_jp2mrsid.html)>
- MrSID, Multi-resolution Seamless Image Database: <[http://www.gdal.org/frmt\\_mrsid.html](http://www.gdal.org/frmt_mrsid.html)>
- NITF: <[http://www.gdal.org/frmt\\_nitf.html](http://www.gdal.org/frmt_nitf.html)>
- ECW, ERDAS Compressed Wavelets (.ecw): <[http://www.gdal.org/frmt\\_ecw.html](http://www.gdal.org/frmt_ecw.html)>
- JP2ECW, JPEG2000 (.jp2, .j2k): [http://www.gdal.org/frmt\\_jpeg2000.html](http://www.gdal.org/frmt_jpeg2000.html)
- AIG, Arc/Info Binary Grid: <[http://www.gdal.org/frmt\\_various.html#AIG](http://www.gdal.org/frmt_various.html#AIG)>
- JP2KAK, JPEG2000 (.jp2, .j2k): <[http://www.gdal.org/frmt\\_jp2kak.html](http://www.gdal.org/frmt_jp2kak.html)>

### 6.21.1 Installing GDAL

GDAL is not a standard GeoServer extension, as the GDAL library files are built into GeoServer by default. However, in order for GeoServer to leverage these libraries, the GDAL (binary) program itself must be installed through your host system's OS. Once this program is installed, GeoServer will be able to recognize GDAL data types. In order to install the GDAL Native libraries:

1. Navigate to the [imageio-ext document and files download page](#).
2. Select the most recent stable binary release.
3. Select "native libraries".
4. Download and extract/install the correct version for your OS.

**Note:** 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 be the folder where the SOs are extracted.

5. Select "libraries" from the last stable release root.

## New Vector Data Source

Vector Product Format Library

Vector Product Format Library data store implementation.

### Basic Store Info

#### Workspace

cite ▼

#### Data Source Name

#### Description

☒ Enabled

### Connection Parameters

#### URL

Save

Cancel

Figure 6.39: *Configuring a VPF data store*

6. Download and extract the gdal\_data-1.X.X archive.

**Note:** Make sure to set a GDAL\_DATA environment variable to the folder where you have extracted this file.

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

## Raster Data Sources



Figure 6.40: GDAL image formats in the list of raster data stores

### 6.21.2 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 worrisky 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.path.1=bin/wrapper/lib line.



## Add Raster Data Source

Description

DTED

DTED Coverage Format

### Basic Store Info

#### Workspace

cite ▼

#### Data Source Name

#### Description

☒ Enabled

### Connection Parameters

#### URL

Save

Cancel

Figure 6.41: *Configuring a DTED data store*

## Add Raster Data Source

Description

---

EHdr

EHdr Coverage Format

### Basic Store Info

---

#### Workspace

cite ▼

#### Data Source Name

#### Description

☒ Enabled

### Connection Parameters

---

#### URL

Figure 6.42: *Configuring a EHdr data store*

### 6.21.3 Adding support for ECW and Kakadu

### 6.21.4 Configuring a DTED data store

### 6.21.5 Configuring a EHdr data store

### 6.21.6 Configuring a ERDASImg data store

## Add Raster Data Source

Description

ERDASImg

Erdas Imagine Coverage Format

### Basic Store Info

#### Workspace

cite

#### Data Source Name

#### Description

☒ Enabled

### Connection Parameters

#### URL

file:data/example.extension

Save

Cancel

Figure 6.43: Configuring a ERDASImg data store

### 6.21.7 Configuring a JP2MrSID data store

### 6.21.8 Configuring a NITF data store

## 6.22 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.

### 6.22.1 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.

### 6.22.2 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.

### 6.22.3 Configuring an ImagePyramid data store

Option	Description
Workspace	
Data Source Name	
Description	
Enabled	
URL	

## 6.23 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.

### 6.23.1 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!

## Add Raster Data Source

Description

JP2MrSID

JP2K (MrSID) Coverage Format

### Basic Store Info

#### Workspace

cite ▼

#### Data Source Name

#### Description

☒ Enabled

### Connection Parameters

#### URL

Save

Cancel

Figure 6.44: Configuring a JP2MrSID data store

## Add Raster Data Source

Description

NITF

NITF Coverage Format

### Basic Store Info

#### Workspace

cite ▼

#### Data Source Name

#### Description

☒ Enabled

### Connection Parameters

#### URL

Save

Cancel

Figure 6.45: Configuring a NITF data store

## Raster Data Sources



ImagePyramid - Image pyramidal plugin

Figure 6.46: ImagePyramid in the list of raster data stores

## Add Raster Data Source

Description

ImagePyramid  
Image pyramidal plugin

### Basic Store Info

#### Workspace

cite ▼

#### Data Source Name

#### Description

☒ Enabled

### Connection Parameters

#### URL

Save

Cancel

Figure 6.47: *Configuring an ImagePyramid data store*

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

### 6.23.2 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.

## Raster Data Sources

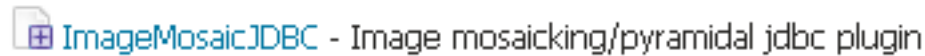


Figure 6.48: *Image Mosaic JDBC in the list of vector data stores*

### 6.23.3 Configuring an Image Mosaic JDBC data store

For a detailed description, look at the [Tutorial](#)

## 6.24 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.

### 6.24.1 Adding an Oracle Georaster data store

Read the geotools documentation for Oracle Georaster Support: <http://docs.codehaus.org/display/GEOTDOC/Oracle+Georaster>  
After creating the xml config file proceed to the section *Configuring GeoServer* in the [Image Mosaic JDBC Tutorial](#)

## 6.25 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.

### 6.25.1 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.codehaus.org/display/GEOTDOC/Customized+JDBC+Access>

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](#)



# Add Raster Data Source

## Description

ImageMosaicJDBC  
Image mosaicking/pyramidal jdbc plugin

## Basic Store Info

### Workspace

cite ▼

### Data Source Name

### Description

☒ Enabled

## Connection Parameters

### URL

Save

Cancel

Figure 6.49: Configuring an Image Mosaic JDBC data store

## 6.26 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.

### 6.26.1 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 <code>OutOfMemory</code> 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.

## 6.27 SQL views

The traditional way to use database backed data is to configure either a table or a database view as a new layer in GeoServer. Starting with GeoServer 2.1.0 the user can also create a new layer by specifying a raw SQL query, without the need to actually creating a view in the database. The SQL can also be parametrized, and parameter values passed in along with a WMS or WFS request.

### 6.27.1 Creating a plain SQL view

In order to create an SQL view the administrator can go into the “create new layer” page. Upon selection of a database backed store a list of tables and views available for publication will appear, but at the bottom of if a new link, “create SQL view”, will appear:

Selecting the link will open a new page where the SQL statement can be specified:

## New Layer chooser

Add layer from

Here is a list of resources contained in the store 'postgis'. Click on the layer you wish to configure

<div> <div>&lt;&lt;</div> <div>&lt;</div> <div>1</div> <div>&gt;</div> <div>&gt;&gt;</div> </div> <div>Results 0 to 0 (out of 0 items)</div> <div> <input type="text" value="Search"/> </div>		
Published	Layer name	
✓	parks	<a href="#">Publish again</a>
✓	pgstates	<a href="#">Publish again</a>
	bc_parks_2001	<a href="#">Publish</a>
	bc_roads	<a href="#">Publish</a>
	newvector	<a href="#">Publish</a>
	zips00	<a href="#">Publish</a>
<div> <div>&lt;&lt;</div> <div>&lt;</div> <div>1</div> <div>&gt;</div> <div>&gt;&gt;</div> </div> <div>Results 0 to 0 (out of 0 items)</div>		

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

### SQL statement

```
select gid, state_name, persons, the_geom from
pgstates where state_name ilike '%n%'
```

**Note:** The query can be any SQL statement that can be validly executed as part of a subquery in the FROM clauses, that is `select * from (<the sql view>) [as] vtable`. This is true for most SQL statements, but specific syntax might be needed to call onto a stored procedure depending on the database. Also, all the columns returned by the SQL statement must have a name, in some databases aliasing is required when calling function names

Once a valid SQL statement has been specified press the “refresh” link in the Attributes table to get a list of the feature type attributes:

#### Attributes

[Refresh](#)

Name	Type	SRID	Identifier
gid	Integer		<input checked="" type="checkbox"/>
state_name	String		<input type="checkbox"/>
persons	BigDecimal		<input type="checkbox"/>
the_geom	<input type="text" value="MultiPolygon"/>	<input type="text" value="4326"/>	<input type="checkbox"/>

GeoServer will do its best to figure out automatically the geometry type and the native srid, but they should always be double checked and eventually corrected. In particular having the right SRID (spatial reference id) is key to have spatial queries actually work. In many spatial databases the SRID is equal to the EPSG code for the specific spatial reference system, but that is not always true (e.g., Oracle has a number of non EPSG SRID codes).

If stable feature ids are desired for the view’s features one or more column providing a unique identification for the features should be checked in the “Identifier” column. Always make sure those attributes generate a actually unique key, or filtering and WFS clients will mishbehave.

Once the query and the attribute details are set press save and the usual new layer configuration page will show up. That page will have a link to a SQL view editor at the bottom of the “Data” tab:

### Feature Type Details

Property	Type	Nillable	Min/Max Occurences
state_name	String	true	0/1
persons	BigDecimal	true	0/1
the_geom	MultiPolygon	true	0/1

[Edit sql view](#)

Once create the SQL view based layer can be used as any other table backed layer.

### 6.27.2 Creating a parametric SQL view

**Warning:** As a rule of thumb use SQL parameter substitution only if the required functionality cannot be obtained with safer means, such as dynamic filtering (CQL filters) or SLD parameter substitution. Only use SQL parameters as a last resort, improperly validated parameters can open the door to [SQL injection attacks](#).

A parametric SQL view is based on a SQL query containing parameters whose values can be dynamically provided along WMS or WFS requests. A parameter is bound by % signs, can have a default value, and should always have a validation regular expression.

Here is an example of a SQL query with two parameters, `low` and `high`:

#### View Name

popstates

#### SQL statement

```
select gid, state_name, the_geom from pgstates where
persons between %low% and %high%
```

The parameters can be manually specified, but GeoServer can figure out the parameter names by itself when the “Guess parameters from SQL” link is clicked. The result will be a parameter table filled with the parameter names and some default validation expressions:

#### SQL view parameters

[Guess parameters from SQL](#) [Add new parameter](#) [Remove selected](#)

<input type="checkbox"/>	Name	Default value	Validation regular expression
<input type="checkbox"/>	high		^[\\w\\d\\s]+\$
<input type="checkbox"/>	low		^[\\w\\d\\s]+\$

In this case query cannot be executed without default values, as `select gid, state_name, the_geom from pgstates where persons between` and would be invalid SQL. Moreover, the two parameters are positive integer numbers, so the validation expression can be adjusted to allow only that kind of input:

#### SQL view parameters

[Guess parameters from SQL](#) [Add new parameter](#) [Remove selected](#)

<input type="checkbox"/>	Name	Default value	Validation regular expression
<input type="checkbox"/>	high	100000000	^[\\d]+\$
<input type="checkbox"/>	low	0	^[\\d]+\$

Once the default values have been set the “Attributes” refresh link can be used to double check the query, retrieve the attributes and eventually fix the geometry and identifier details. At this point the workflow proceeds just like with a non parametrized query.

Going to the WMS preview for the `popstates` layer should result in all the states being displayed. The SQL view parameters can now be specified by adding the `viewparams` parameter in the GetMap request. `viewparams` is structured as a set of key/value pairs separated by semicolons: `viewparams=p1:v1;p2:v2;...`

For example, to select all states having more than 20 million inhabitants the following params can be added to the normal GetMap request: `low:20000000`



In order to get all the states having between 2 and 5 millions inhabitants the following can be specified instead: `&viewparams=low:2000000;high:5000000`

### 6.27.3 Parameters and validation

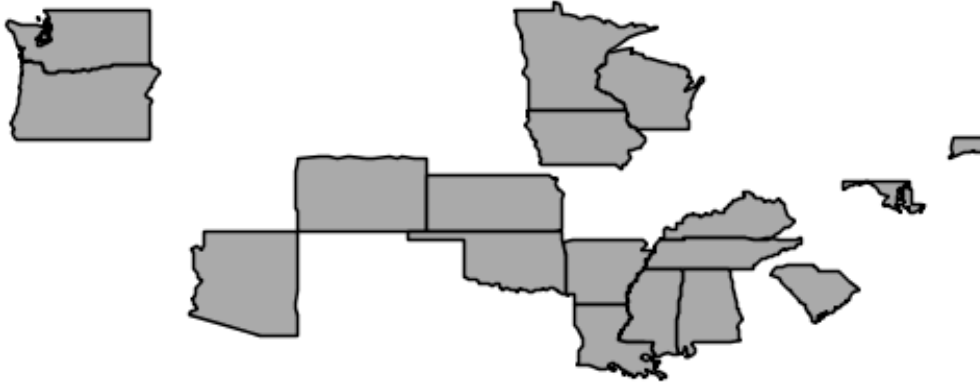
A SQL view parameter can be anything, the only rule to follow is that the set of attributes returned by the view and their types must never change. This in particular means it's possible to create views containing wide open parameters allowing to specify full SQL query portions.

For example, `select * from pgstates %where%`, along with an empty validation regular expression, would allow to specify the where clause of the query dynamically. However, that opens a serious risk for [SQL injection attacks](#) unless access to the server is allowed only to trusted parties.

In general it is advised to use SQL parameters with great care and cast a validation regular expression that only allows for the intended parameter values. The expression should be created to prevent attacks, but not necessarily to double check the value is the expected type.

For example:

- `^[\\d\\.\\+-eE]+$` will check that the parameter value is composed with valid elements for a floating point number, eventually in scientific notation, but will not check that the provided value is actually a valid floating point number



- `[^;']+` will check the parameter value does not contain quotes or semicolon, preventing common sql injection attacks, without actually imposing much on the parameter value structure

#### 6.27.4 Regular expressions references

Casting the proper validation regular expression is important in terms of security. Regular expressions are a wide topic that cannot be addressed in a short space. Here is a set of links on the internet to get more information about this topic:

- The regular expression engine used by GeoServer is the Java built-in one. The [Pattern class javadocs](#) contain the full specification of the allowed syntax.
- This <http://www.regular-expressions.info> site is fully dedicated to regular expressions, with tutorials and examples.
- This [applet](#) can be used to interactively test a regular expression online.

## 6.28 Application Schema Support

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.

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.

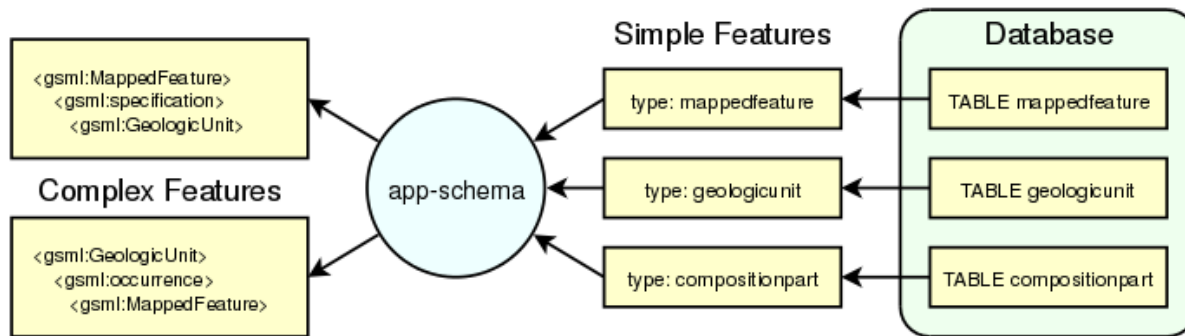


Figure 6.50: Three tables in a database are accessed using GeoServer simple feature support and converted into two complex feature types.

## 6.28.1 Contents

### Complex Features

To understand complex features, and why you would want use them, you first need to know a little about simple features.

### 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 properties, 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.

## 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 sub-types..

### 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 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://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`.

## 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]).

## 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,

## 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>
```

## 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>
```

## 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.

## Workspace layout

The GeoServer data directory contains a folder called `workspaces` with the following structure:

```
workspaces
- gsml
  - SomeDataStore
    - SomeFeatureType
      - featurertype.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).

## 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.

## 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 XPath expressions in the output XML.

## 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.

## 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>
```

## 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>
    <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.

**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** A CQL expression that is used to set the `gml:id` of the output feature type. This could be a column in a database, the automatically generated simple feature ID obtained with `getId()`, or some other expression.

**Note:** Every feature type must have one `idExpression` mapping to set its `gml:id`. This requirement is an implementation limitation (strictly, `gml:id` is optional in GML).

**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 expressions for vocabulary translations. Read more about it in *Vocabulary functions*.

**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. This mapping must come before the mapping for the enclosed elements. In this example, `gsml:positionalAccuracy` is a `gsml:CGI_ValuePropertyType` which contains a `gsml:CGI_Value`, which is abstract. In this case, `targetAttributeNode` must be used to set the type of the property type to a type that encloses a non-abstract element:

```
<AttributeMapping>
  <targetAttribute>gsml:positionalAccuracy</targetAttribute>
  <targetAttributeNode>gsml:CGI_TermValuePropertyType</targetAttributeNode>
</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 “striping” 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` often appears in a mapping with `targetAttribute` and no other elements.

**isMultiple (optional)** The `isMultiple` element states whether there might be multiple values for this attribute. Because the default value is `false` and it is omitted in this case, it is most usually seen as:

```
<isMultiple>true</isMultiple>
```

**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 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 as `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.

## CQL

- String literals are enclosed in single quotes, for example `'urn:ogc:def:nil:OGC:missing'`.
- The uDig manual contains information on CQL:
  - <http://udig.refrains.net/confluence/display/EN/Common+Query+Language>

## Database identifiers

When referring to database table/view names or column names, use:

- lowercase for PostGIS
- UPPERCASE for Oracle Spatial and ArcSDE

## 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 be 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.

## 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.

## Supported GML versions

- At this time, only GML 3.1.1 application schemas are supported.

- GML 3.1.1 is distributed with GeoServer and does not need to be downloaded nor supplied by the user.

## 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.

## 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?).

## 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.

## 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.

## 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).

## Secondary Namespaces

### 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/>
```

## 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". In order to allow GeoServer App-Schema to support secondary namespaces, please follow the steps outlined below:

Using the sampling namespace as an example.

### 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.

### Step 2:Create files

Create two files below in the "sa" folder:

1. namespace.xml
2. workspace.xml

### 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.

## Vocabulary functions

### Scope

This page 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](#).

**Versions supported** This functionality is supported from GeoTools version 2.6-M2 onwards.

### Useful functions

**Recode function** 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 function** 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 function** This is the new function Jody implemented, and 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.

## 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.

## 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.

## 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.

## 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>

```

## 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

```

## 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](#).

## 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.

## 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	

## 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>
  </parameters>
</DataStore>
```

```

    <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.

## 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.

## 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.

## 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.

## 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.

## 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 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 may 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](#).

## Feature Chaining

### 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
- Handling 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.

The current state of the User guide refers to setting up Geotools application schema configurations, and will be updated to reflect a Geoserver 2.0 configuration example shortly. For non-application schema configurations, please refer to [Data Access Integration](#).

**Versions supported** Feature chaining is implemented within the app-schemas module in GeoTools trunk (from 2.6.x on). Work on supporting this in Geoserver trunk is currently underway. It is not supported in GeoServer 1.6 community-schemas.

## 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>getID()</OCQL>
    <linkElement>gsml:CGI_TermValue</linkElement>
    <linkField>FEATURE_LINK[1]</linkField>
  </sourceExpression>
  <isMultiple>true</isMultiple>
</AttributeMapping>
<AttributeMapping>
  <targetAttribute>gsml:eventProcess</targetAttribute>
  <sourceExpression>
    <OCQL>getID()</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	GEO-LOGIC_UNIT_ID:String	ghmi-nage: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:value>
        </gsml:CGI_TermValue>
      </gsml:lower>
    </gsml:CGI_TermRange>
  </gsml:eventAge>
</gsml:GeologicEvent>
```

```
    <gsml:CGI_TermValue>
      <gsml:value codeSpace="urn:cgi:classfierScheme:ICS:StratChart:2008">Paleocene</gsml:value>
    </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:classfierScheme:GSV:GeologicalUnitCode'</value>
  </ClientProperty>
</AttributeMapping>
<AttributeMapping>
  <targetAttribute>gml:name[2]</targetAttribute>
  <sourceExpression>
    <OCQL>NAME</OCQL>
  </sourceExpression>
  <ClientProperty>
    <name>codeSpace</name>
    <value>'urn:cgi:classfierScheme:GSV:GeologicalUnitName'</value>
  </ClientProperty>
</AttributeMapping>
<AttributeMapping>
  <targetAttribute>gml:name[3]</targetAttribute>
  <sourceExpression>
```

```

    <OCQL>strTrim(getId())</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:

```

<gsm1:specification>
  <gsm1: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</gml:name>
    <gml:name codeSpace="urn:cgi:classifierScheme:GSV:MappedFeatureReference">gu.25678</gml:name>
  </gsm1:GeologicUnit>
</gsm1:specification>

```

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>gsm1:composition</targetAttribute>
  <sourceExpression>
    <OCQL>getID()</OCQL>
    <linkElement>gsm1:CompositionPart</linkElement>
    <linkField>FEATURE_LINK</linkField>
  </sourceExpression>
  <isMultiple>true</isMultiple>
</AttributeMapping>

```

- *OCQL*: `getID()` returns 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	ABBREVI- ATAION: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	COMPO- NENT_ROLE:String	PROPOR- TION:String	GEO- LOGIC_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.

## 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 feature-type.xml in its location.

## Nesting simple properties

You don't need to chain multi-valued simple properties and map them separately. The original configuration would still work.

## 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:PropertyName>
      <ogc:Literal>Olivine basalt, tuff, microgabbro, minor sedimentary rocks</ogc:Literal>
    </ogc:Function>
    <ogc:Literal>1</ogc:Literal>
  </ogc:PropertyIsEqualTo>
</ogc:Filter>
```

## 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>strTrim(getId())</OCQL>
    <linkElement>gsml:MappedFeature</linkElement>
    <linkField>gsml:specification</linkField>
  </sourceExpression>
  <isMultiple>true</isMultiple>
  <ClientProperty>
    <name>xlink:href</name>
    <value>strConcat('urn:cgi:feature:MappedFeature:', getId())</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"/>
</gsml:GeologicUnit>
```

**Note:**

- In the example above, we use `strConcat('urn:cgi:feature:MappedFeature:', getId())` as Client Property value. The function `getId()` would return the id value from the nested feature table (`gsml:MappedFeature`). You can use other column names from the nested feature data store.
- Lastly, 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.

## Troubleshooting

1. **Error message:** `"java.lang.RuntimeException: org.geotools.data.DataSourceException: Feature type ... not found. H`
  - Check that the nested feature type mapping file exists.
  - Check that the nested feature type name is consistent with the `linkElement` in the containing feature type.
2. **The nested features aren't shown.**
  - Check that the OCQL tag in the "container" type points to the right column in the data store.
  - If the nested type uses `getID()` as the OCQL source expression for the referenced field, ensure it's wrapped in String converting functions such as `strTrim()` or `strConcat()`.
3. **Error message:** `"java.lang.IllegalArgumentException: Don't know how to map ..."`

- Check that the linkField tag in the “container” type points to the right field on the nested type.

#### 4. Wrong nested features (too many) appeared inside the “container” features.

- If the relationship is many-to-many, make sure you are not using the same (denormalized) view for both sides of the nesting.

## 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.

## 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>
```

## 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.

## 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:OGC:1.0:missing')))
```

**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.

## 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.

## 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 just need to specify targetAttributeNode for inline mappings. 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>
```

Using feature chaining, we just chain it as usual:

```
<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>
```

## 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.

## Data Access Integration

This page assumes prior knowledge of [Application Schema Support](#) 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.

## How to connect to the input data access

Configure the data store connection in “`sourceDataStores`” tag as usual, but also specify the additional “`is-DataAccess`” 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>
  </FeatureTypeMapping>
</typeMappings>
...

```

## 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>

```

## 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:occurrence</targetAttribute>
  <sourceExpression>
    <inputAttribute>mo:commodityDescription</inputAttribute>
    <linkElement>gsml:MappedFeature</linkElement>
    <linkField>gml:name[2]</linkField>
  </sourceExpression>
  <isMultiple>true</isMultiple>
</AttributeMapping>

```

## 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/gsml:TermValue
    </ogc:Function>
    <ogc:Literal>Olivine basalt, tuff, microgabbro, minor sedimentary rocks</ogc:Literal>
  </ogc:PropertyIsEqualTo>
</ogc:Filter>
```

## Tutorial

This tutorial demonstrates how to configure two complex feature types using the app-schema plugin and data from two property files.

## 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`.

## Installation

- Install GeoServer as usual.
- Install the app-schema plugin (place the jar files in `WEB-INF/lib`).
- The tutorial configuration is a complete working GeoServer data directory. It includes all the schema (XSD) files required to use GeoSciML 2.0, the data files, and the app-schema configuration files. There are two ways you can get it:

1. Download **geoserver-app-schema-tutorial-config.zip** and unzip it into the folder that you will use as your data directory.
  2. Check it out from the [AuScope subversion repository](#).
- If the data directory differs from the default, edit `WEB-INF/web.xml` to set `GEOSERVER_DATA_DIR`. (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.

## 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.

## 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.

**Catalog** Both feature types use the same OASIS XML Catalog, given as a path relative to the mapping file:

```
<catalog>../../../../schemas/catalog.xml</catalog>
```

- The catalog contains the the XSD schemas for GeoSciML 2.0 its dependencies.
- Note that some dependencies are imported as relative filesystem paths, and so are not resolved through the OASIS Catalog, but are still present on the filesystem.
- GML 3.1.1 is also a dependency, but is not required because it is distributed with GeoServer.
- Use of a catalog is required because the implementation otherwise fails to honour relative imports.



**Target types** Both feature types are defined 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>getId()</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. For database sources, `getId()` will synthesise an id from the table or view name, a dot “.”, and the primary key of the table. If this is not desirable, any other 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 used 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).

## 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&typeName=gsml:GeologicUnit>
- <http://localhost:8080/geoserver/wfs?request=GetFeature&typeName=gsml:MappedFeature>

### gsml:GeologicUnit

- **The WFS response for `gsml:GeologicUnit`** contains two features corresponding to the two rows in `gsml_GeologicUnit.properties`. The response document has been manually pretty-printed, so contains more whitespace than the original GeoServer response, but is otherwise a complete WFS response.
- 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.

**Note:** The data in this tutorial is fictitious. Some of the text and numbers have been taken from real data, but have been modified to the extent that they have no real-world meaning.

---

# Filtering in GeoServer

---

Filtering allows to identify features that satisfy a specific set of conditions. This can be used to reduce the amount of data returned by WFS or to apply different symbolization on a thematic map.

## 7.1 GeoServer supported filter languages

Data filtering in GeoServer is based on the concepts found in the [OGC Filter Encoding Specification](#), which we suggest the reader to get familiar with.

In particular GeoServer accepts filters encoded in three different languages:

- [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
- [CQL, Catalog Query Language](#), a plain text language created for the OGC Catalog specification and adapted to be a general and easy to use filtering mechanism.
- [ECQL, Extended CQL](#), an extension to CQL that allows to express the same filters OGC Filter 1.1 can encode. A quick [CQL and ECQL](#) is also available in this guide that shows examples of both CQL and ECQL.

We suggest to look into the respective specifications for details.

## 7.2 Filter functions

The OGC Filter encoding specification contains a generic concept, the *filter function*.

A *filter function* is a function, with arguments, that can be called inside of a filter or, more generically, an expression, to perform specific calculations: as such it can be useful when building WFS filters or SLD style sheets. A filter function can be anything a trigonometric function, a string formatting one, a geometry buffer.

The filter specification does not mandate specific functions, so while the syntax to call a function is uniform, any server is free to provide whatever function it wants, so the actual invocation will work only against specific software.

Here are a couple of examples on function usage, the first is about WFS filtering, the second a way to use functions in SLD to get richer rendering.

## 7.2.1 WFS filtering example

Let's assume we have a WFS 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 multi point. This cannot be achieved with a fully portable filter, but it can be done using a GeoServer specific filter function named `geometryType`. Here is how:

```
<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>
```

## 7.2.2 SLD formatting example

We want to include elevation labels in a contour map. The label is stored as a floating point, and the resulting labelling will be something like "150.0" or "149.999999". We want to avoid that and get 150 instead. To achieve this result we can use the `numberFormat` filter function:

```
...
<TextSymbolizer>
  <Label>
    <ogc:Function name="numberFormat">
      <ogc:Literal>##</ogc:Literal>
      <ogc:PropertyName>ELEVATION</ogc:PropertyName>
    </ogc:Function>
  </Label>
  ...
</TextSymbolizer>
...
```

## 7.2.3 Performance implications

Using filter functions in SLD symbolizer expressions does not have significant overhead, unless the function is performing some very heavy computation.

However, using them in WFS or SLD filtering can take a very visible toll: this is usually because filter functions are not recognized by the native encoders, and thus the functions are not used inside the primary filters, and are performed 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, one, the bounding box filter, is actually used as a primary filter (e.g., encoded in SQL) whilst the geometry function part will be executed in memory on top of the results coming from the primary filter.

## 7.3 Filter Function Reference

This page contains a reference to filter functions that can be used in WFS filtering or in SLD expressions. If a function reported by the WFS capabilities is not available in this list it might either mean that the function cannot actually be used for the above purposes, or that it's new and has not been documented still. Ask for details on the user mailing list.

Unless otherwise specified none of the filter functions in this references is understood natively by the data stores, and as a result all expressions using them will be evaluated in memory.

### 7.3.1 Function argument type reference

Type	Description
Double	Floating point number, 8 bytes, IEEE 754. ranging from 4.94065645841246544e-324d to 1.79769313486231570e+308d
Float	Floating point number, 4 bytes, IEEE 754. ranging 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 Number	Integer number, ranging from -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807
String	Can be any type of number
Timestamp	A sequence of characters
	Date and time information

### 7.3.2 Comparison and control Functions

Name	Arguments	Description
between	num:Number, low:Number,“high”:Number	returns true if <code>low &lt;= num &lt;= high</code>
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 <code>x &gt;= y</code> . Parameters can be either numbers or strings (in the second case lexicographic ordering is used)
greaterThan	x:Object, y:Object	Returns true if <code>x &gt; y</code> . Parameters can be either numbers or strings (in the second case lexicographic ordering is used)
if_the_else	condition:Boolean x:Object, y:Object	Returns <code>x</code> if the condition is true, <code>y</code> otherwise
in10, in9, in8, in7, in6, in5, in4, in3, in2	candidate:Object, v1:Object, ..., v9:Object	Returns true if <code>candidate</code> is equal to one of the <code>v1, ..., v9</code> values. Use the appropriate function name depending on how many arguments you need to pass.
isLike	string:String, pattern:String	Returns true if the string matches the specified pattern. For the full syntax of the pattern specification see the <a href="#">Java Pattern class javadocs</a>
isNull	obj:Object	Returns true the passed parameter is <code>null</code> , false otherwise
lessThan	x:Object, y:Object	Returns true if <code>x &lt; y</code> . Parameters can be either numbers or strings (in the second case lexicographic ordering is used)
lessThanEqual	x:Object, y:Object	Returns true if <code>x &lt;= y</code> . Parameters can be either numbers or strings (in the second case lexicographic ordering is used)
not	bool:Boolean	Returns the negation of <code>bool</code>
notEqual	x:Object, y:Object	Returns true if <code>x</code> and <code>y</code> are equal, false otherwise

### 7.3.3 Feature functions

Name	Arguments	Description
id	feature:Feature	returns the identifier of the feature
PropertyExists	f:Feature, propertyName:String	Returns <code>true</code> if <code>f</code> has a property named <code>propertyName</code>

### 7.3.4 Geometric Functions

Most of the geometric function listed below refer to geometry relationship, to get more information about the meaning of each spatial relationship consult the [OGC Simple Feature Specification for SQL](#)

Name	Arguments	Description
Area	geometry:Geometry	The area of the specified geometry
boundary	geometry:Geometry	Returns the boundary of a geometry
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
bufferWithSegments	geometry:Geometry, distance:Double, segments:Integer	Returns the buffered area around
centroid	geometry:Geometry	Returns the centroid of the geometry
contains	a:Geometry, b:Geometry	Returns true if the geometry <code>a</code> contains <code>b</code>
convexHull	geometry:Geometry	Returns the convex hull of the geometry
crosses	a:Geometry, b:Geometry	Returns true if <code>a</code> crosses <code>b</code>
difference	a:Geometry, b:Geometry	Returns all the points that sit in <code>a</code> but not in <code>b</code>



dimension	a:Geometry	Returns the dimension of the specified geometry
disjoint	a:Geometry, b:Geometry	Returns true if the two geometries do not intersect
distance	a:Geometry, b:Geometry	Returns the euclidean distance between the two geometries
endPoint	line:LineString	Returns the end point of the line
envelope	geometry:Geometry	Returns the polygon representing the envelope of the geometry
equalsExact	a:Geometry, b:Geometry	Returns true if the two geometries are exactly equal
equalsExactTolerance	a:Geometry, b:Geometry, tol:Double	Returns true if the two geometries are equal within the specified tolerance
exteriorRing	poly:Polygon	Returns the exterior ring of the polygon
geometryType	geometry:Geometry	Returns the type of the geometry
geomFromWKT	wkt:String	Returns the Geometry represented by the WKT string
geomLength	geometry:Geometry	Returns the length/perimeter of the geometry
getGeometryN	collection:GeometryCollection, n:Integer	Returns the n-th geometry inside the collection
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 interior or on the boundary of the geometry
interiorRingN	poly:Polygon, n:Integer	Returns the n-th interior ring of the polygon
intersection	a:Geometry, b:Geometry	Returns the intersection between the two geometries
intersects	a:Geometry, b:Geometry	Returns true if a intersects b
isClosed	line:LineString	Returns true if line forms a closed loop
isEmpty	geometry:Geometry	Returns true if the geometry does not contain any points
isometric	geometry:Geometry, extrusion:Double	Returns a multi-polygon containing the isometric extrusion of the geometry
isRing	line:LineString	Returns true if the line is actually a ring
isSimple	line:LineString	Returns true if the geometry self intersects
isValid	geometry:Geometry	Returns true if the geometry is topologically valid
isWithinDistance	a:Geometry, b:Geometry, distance:Double	Returns true if the distance between the two geometries is less than or equal to the specified distance
numGeometries	collection:GeometryCollection	Returns the number of geometries in the collection
numInteriorRing	poly:Polygon	Returns the number of interior rings in the polygon
numPoint	geometry:Geometry	Returns the number of points (vertices) in the geometry
offset	geometry:Geometry, offsetX:Double, offsetY:Double	Offsets all points in a geometry by the specified offsets
overlaps	a:Geometry, b:Geometry	Returns true if a overlaps with b
pointN	geometry:Geometry, n:Integer	Returns the n-th point inside the geometry
relate	a:Geometry, b:Geometry	Returns the DE-9IM intersection model string
relatePattern	a:Geometry, b:Geometry, pattern:String	Returns true if the DE-9IM intersection model matches the specified pattern
startPoint	line:LineString	Returns the starting point of the line
symDifference	a:Geometry, b:Geometry	Returns the symmetrical difference of the two geometries
touches	a:Geometry, b:Geometry	Returns true if a touches b according to the DE-9IM model
toWKT	geometry:Geometry	Returns the WKT representation of the geometry
union	a:Geometry, b:Geometry	Returns the union of a and b (the area covered by both)
vertices	geom:Geometry	Returns a multi-point made with the vertices of the geometry
within	a:Geometry, b:Geometry	Returns true if a is fully contained inside b



### 7.3.5 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 expressed in radians, in the range of 0.0 through $\pi$
asin	angle:Double	Returns the arc sine of an angle expressed in radians, in the range of $-\pi / 2$ through $\pi / 2$
atan	angle:Double	Returns the arc tangent of an angle, in the range of $-\pi/2$ through $\pi/2$
atan2	x:Double, y:Double	Converts rectangular coordinates (x, y) to polar (r, theta).
ceil	x: Double	Returns the smallest (closest to negative infinity) double value that is greater than or equal to the argument 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 the number is zero, false otherwise
exp	x: Double	Returns Euler's number raised to the power of x
floor	x: Double	Returns the largest (closest to positive infinity) value that is less than or equal to the argument and is equal to a mathematical integer
IEEERemainder	x: Double, y:Double	Computes the remainder operation on two arguments as prescribed by the IEEE 754 standard
int2bool	x: Integer	Returns true if the number is zero, false otherwise
int2ddouble	x: Integer	Converts the number to Double
log	x: Integer	Returns the natural logarithm (base e) of x
max,	x1: Double,	Returns the maximum between x1, ..., x4
max_3,	x2:Double,	
max_4	x3:Double,	
	x4:Double	
min,	x1: Double,	Returns the minimum between x1, ..., x4
min_3,	x2:Double,	
min_4	x3:Double,	
	x4:Double	
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 the argument. 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 <code>(int)floor(a + 0.5)</code>
round-Double	x:Double	Returns the closest Long to the argument
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

### 7.3.6 String functions

Name	Arguments	Description
strCapitalize (since 2.0.2)	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 <code>string</code> ends with <code>suffix</code>
strEqual-Ignore-Case	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 <a href="#">Java Pattern class javadocs</a>
strReplace	string:String, pattern:String	Returns true if the string matches the specified regular expression. For the full syntax of the pattern specification see the <a href="#">Java Pattern class javadocs</a>
strStartsWith	string:String, prefix:String	Returns true if <code>string</code> starts with <code>prefix</code>
strSub-string	string:String, begin:Integer, end:Integer	Returns a new string that is a substring of this string. The substring begins at the specified <code>begin</code> and extends to the character at index <code>endIndex - 1</code> (indexes are zero-based).
strSub-stringStart	string:String, begin:Integer	Returns a new string that is a substring of this string. The substring begins at the specified <code>begin</code> 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

### 7.3.7 Parsing and formatting functions

Name	Arguments	Description
date- Format	date:Timestamp, format:String	Formats the specified date according to the provided format. The format syntax can be found in the <a href="#">Java SimpleDateFormat javadocs</a>
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 <a href="#">Java SimpleDateFormat javadocs</a>
num- ber- Format	number:Double, format:String	Formats the number according to the specified format. The format syntax can be found in the <a href="#">Java DecimalFormat javadocs</a>
parse- Boolean	boolean:String	Parses a string into a boolean. The empty string, f, 0.0 and 0 are considered false, everything else is considered true.
parse- Dou- ble	number:String	Parses a string into a double. The number can be expressed in normal or scientific form.
par- seInt	number:String	Parses a string into an integer.
parse- Long	number:String	Parses a string into a long integer



---

# Styling

---

This section discusses the styling of geospatial data served through GeoServer.

## 8.1 Introduction to SLD

Geospatial data has no intrinsic visual component. In order to see data, it must be styled. This means to specify color, thickness, and other visible attributes. In GeoServer, this 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, though it can be intimidating. This page will give a basic introduction to what one can do with SLD and how GeoServer handles it.

**Note:** Since GeoServer uses SLD exclusively for styling, the terms “SLD” and “style” will often be used interchangeably.

### 8.1.1 Types of styling

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.

More advanced styling is possible than just color and thickness. 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. Styles can be based on attributes in the data, so that certain features are styled differently. Text labels on features are possible as well. Features can be styled based on zoom level, with the size of the feature determining how it is displayed. The possibilities are vast.

### 8.1.2 Style metadata

### 8.1.3 GeoServer and SLD

Every layer (featuretype) registered with GeoServer needs to have at least one style associated with it. GeoServer comes bundled with a few basic styles, and any number of new styles can be added. It is

possible to change any layer's associated style at any time in the [Layers](#) page of the [Web Administration Interface](#). When adding a layer and a style to GeoServer at the same time, the style should be added first, so that the new layer can be associated with the style immediately. You can add a style in the [Styles](#) menu of the [Web Administration Interface](#).

## 8.1.4 Definitions

### Symbolizer

### Rule

### FeatureTypeStyle

## 8.1.5 A basic style

This SLD takes a layer that contains points, and styles 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>
```

Don't let the lengthy nature of this simple example intimidate; only a few lines are really important to understand. **Line 14** states that we are using a "PointSymbolizer", a style for point data. **Line 17** states that we are using a "well known name", a circle, to style the points. There are many well known names for shapes such as "square", "star", "triangle", etc. **Lines 18-20** states to fill the shape 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 Name/Title/Abstract.



Many more examples can be found in the [SLD Cookbook](#).

**Note:** You will find that some tags have prefixes, such as `ogc:` in front of them. The reason for this is because they are **XML namespaces**. In the tag on **lines 2-7**, there are two XML namespaces, one called `xmlns`, and one called `xmlns:ogc`. Tags corresponding to the first namespace do not need a prefix, but tags corresponding to the second require a prefix of `ogc:`. It should be pointed out that the name of the namespaces are not important: The first namespace could be `xmlns:sld` (as it often is) and then all of the tags in this example would require an `sld:` prefix. The important part is that the namespaces need to match the tags.

### 8.1.6 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. When adding a style into GeoServer, it is automatically checked for validation with the OGC SLD specification (although that may be bypassed), but it will not be checked for errors. It is very easy to have syntax errors creep into a valid SLD. Most of the time this will result in a map displaying no features (a blank map), but sometimes errors will prevent the map from even loading at all.

The easiest way to fix errors in an SLD is to try to isolate the error. If the SLD is long and incorporates many different rules and filters, try temporarily removing some of them to see if the errors go away.

To minimize errors when creating the SLD, it is recommended to use a text editor that is designed to work with XML. Editors designed for XML can make finding and removing errors much easier by providing syntax highlighting and (sometimes) built-in error checking.

## 8.2 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	<a href="#">sld_cookbook_point.zip</a>
Line	<a href="#">sld_cookbook_line.zip</a>
Polygon	<a href="#">sld_cookbook_polygon.zip</a>
Raster	<a href="#">sld_cookbook_raster.zip</a>

### 8.2.1 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 8.1: *Simple point*

## Code

View and download the full “Simple point” 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         </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.

## 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>

```



Figure 8.2: *Simple point with stroke*

```

16     </Graphic>
17   </PointSymbolizer>
18 </Rule>
19 </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 8.3: *Rotated square*

## Code

View and download the full “Rotated square” SLD

```

1   <FeatureTypeStyle>
2     <Rule>

```

```
3      <PointSymbolizer>
4        <Graphic>
5          <Mark>
6            <WellKnownName>square</WellKnownName>
7            <Fill>
8              <CssParameter name="fill">#009900</CssParameter>
9            </Fill>
10           </Mark>
11           <Size>12</Size>
12           <Rotation>45</Rotation>
13         </Graphic>
14       </PointSymbolizer>
15     </Rule>
16   </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).

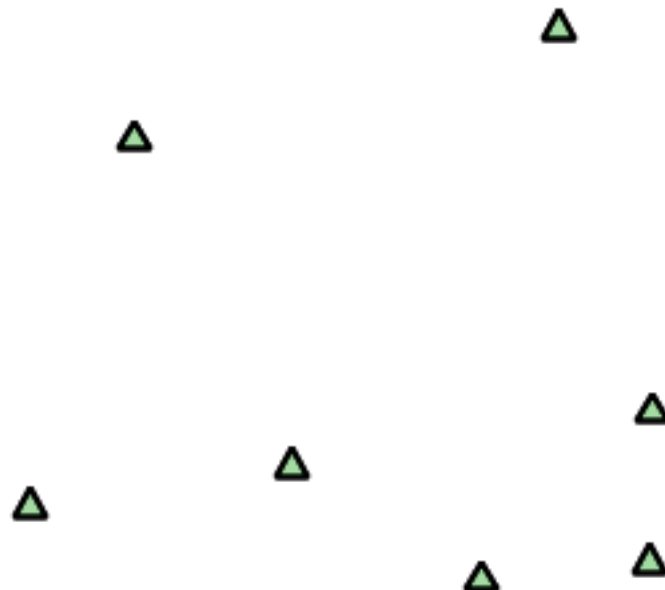


Figure 8.4: *Transparent triangle*

## 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.

## 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" />

```



Figure 8.5: *Point as graphic*



```

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 8.6: 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.

## 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>

```

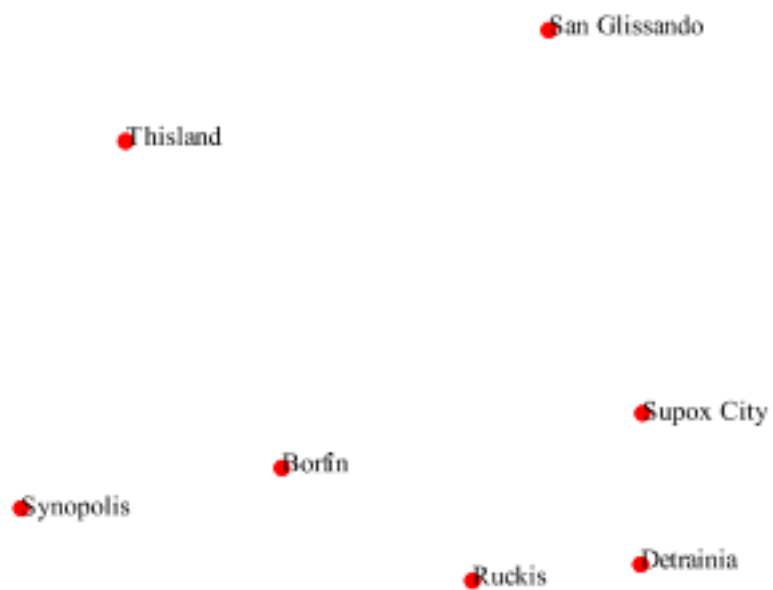


Figure 8.7: *Point with default label*

```

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.

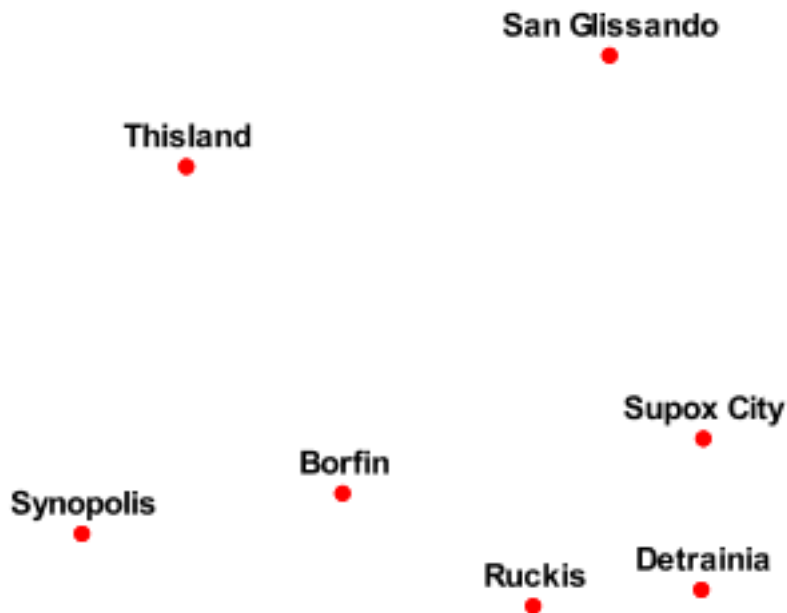


Figure 8.8: *Point with styled label*

## 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.

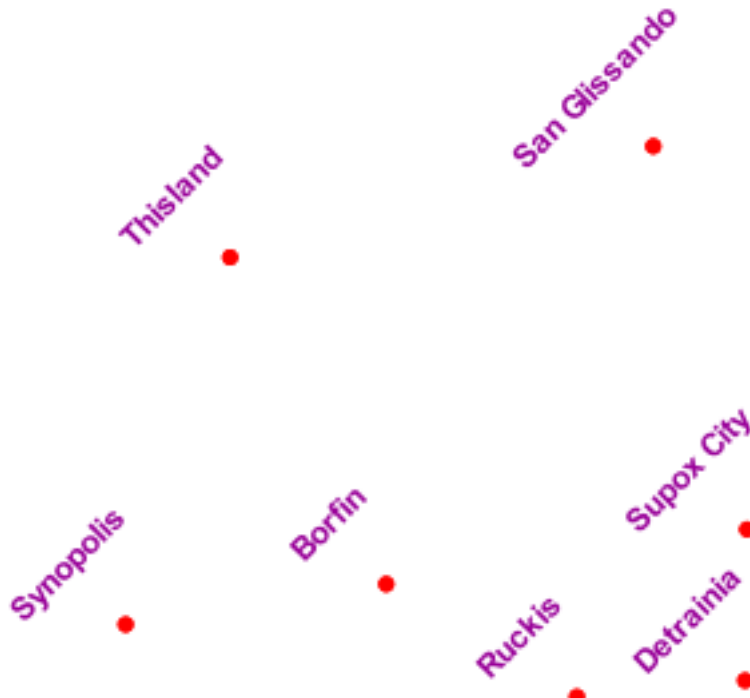


Figure 8.9: *Point with rotated label*

## Code

View and download the full “Point with rotated 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>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 (#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

View and download the full “Attribute-based point” SLD

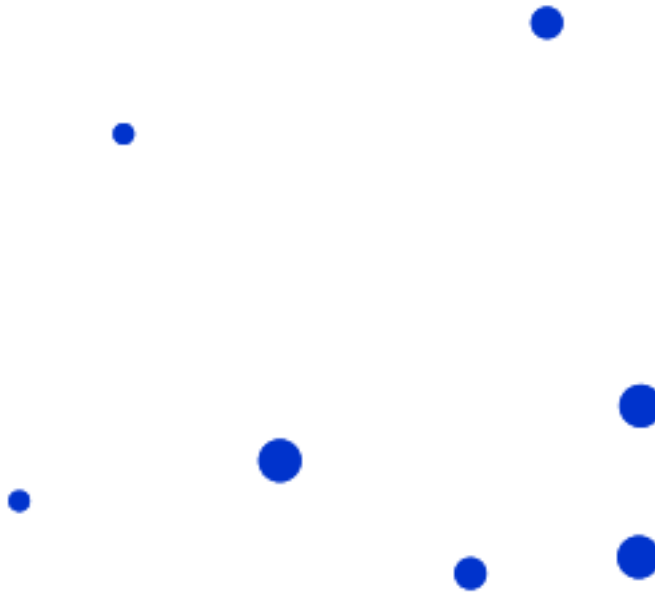


Figure 8.10: *Attribute-based point*

```
1 <FeatureTypeStyle>
2   <Rule>
3     <Name>SmallPop</Name>
4     <Title>1 to 50000</Title>
5     <ogc:Filter>
6       <ogc:PropertyIsLessThan>
7         <ogc:PropertyName>pop</ogc:PropertyName>
8         <ogc:Literal>50000</ogc:Literal>
9       </ogc:PropertyIsLessThan>
10    </ogc:Filter>
11    <PointSymbolizer>
12      <Graphic>
13        <Mark>
14          <WellKnownName>circle</WellKnownName>
15          <Fill>
16            <CssParameter name="fill">#0033CC</CssParameter>
17          </Fill>
18        </Mark>
19        <Size>8</Size>
20      </Graphic>
21    </PointSymbolizer>
22  </Rule>
23  <Rule>
24    <Name>MediumPop</Name>
25    <Title>50000 to 100000</Title>
26    <ogc:Filter>
27      <ogc:And>
28        <ogc:PropertyIsGreaterThanOrEqualTo>
29          <ogc:PropertyName>pop</ogc:PropertyName>
30          <ogc:Literal>50000</ogc:Literal>
31        </ogc:PropertyIsGreaterThanOrEqualTo>
32        <ogc:PropertyIsLessThan>
33          <ogc:PropertyName>pop</ogc:PropertyName>
34          <ogc:Literal>100000</ogc:Literal>
35        </ogc:PropertyIsLessThan>
36      </ogc:And>
37    </ogc:Filter>
38    <PointSymbolizer>
39      <Graphic>
40        <Mark>
41          <WellKnownName>circle</WellKnownName>
42          <Fill>
43            <CssParameter name="fill">#0033CC</CssParameter>
44          </Fill>
45        </Mark>
46        <Size>12</Size>
47      </Graphic>
48    </PointSymbolizer>
49  </Rule>
50  <Rule>
51    <Name>LargePop</Name>
52    <Title>Greater than 100000</Title>
53    <ogc:Filter>
54      <ogc:PropertyIsGreaterThanOrEqualTo>
55        <ogc:PropertyName>pop</ogc:PropertyName>
56        <ogc:Literal>100000</ogc:Literal>
57      </ogc:PropertyIsGreaterThanOrEqualTo>
```



```

58     </ogc:Filter>
59     <PointSymbolizer>
60       <Graphic>
61         <Mark>
62           <WellKnownName>circle</WellKnownName>
63           <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.

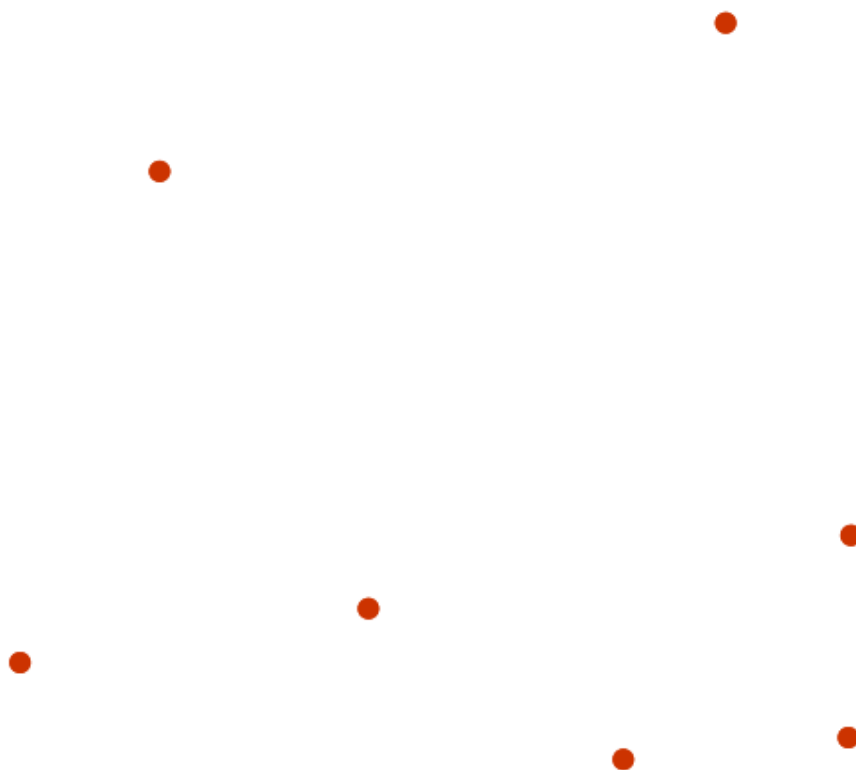


Figure 8.11: *Zoom-based point: Zoomed in*

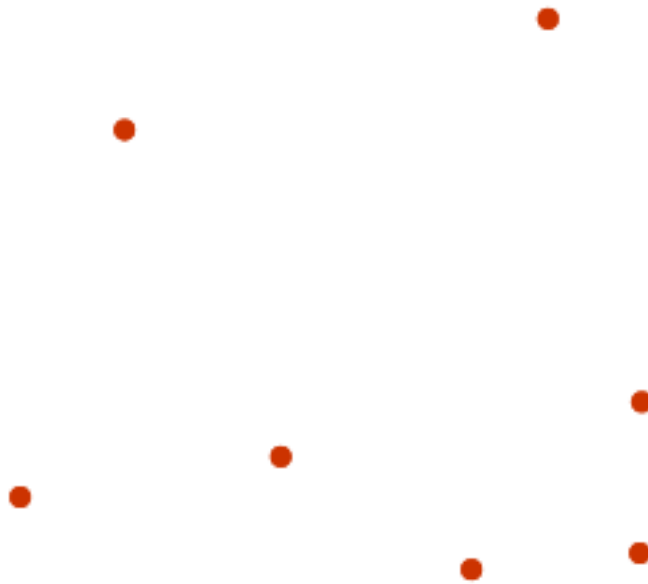


Figure 8.12: *Zoom-based point: Partially zoomed*



Figure 8.13: *Zoom-based point: Zoomed out*

## Code

View and download the full “Zoom-based point” SLD

```

1  <FeatureTypeStyle>
2    <Rule>
3      <Name>Large</Name>
4      <MaxScaleDenominator>160000000</MaxScaleDenominator>
5      <PointSymbolizer>
6        <Graphic>
7          <Mark>
8            <WellKnownName>circle</WellKnownName>
9            <Fill>
10             <CssParameter name="fill">#CC3300</CssParameter>
11           </Fill>
12          </Mark>
13          <Size>12</Size>
14        </Graphic>
15      </PointSymbolizer>
16    </Rule>
17    <Rule>
18      <Name>Medium</Name>
19      <MinScaleDenominator>160000000</MinScaleDenominator>
20      <MaxScaleDenominator>320000000</MaxScaleDenominator>
21      <PointSymbolizer>
22        <Graphic>
23          <Mark>
24            <WellKnownName>circle</WellKnownName>
25            <Fill>
26              <CssParameter name="fill">#CC3300</CssParameter>
27            </Fill>
28          </Mark>
29          <Size>8</Size>
30        </Graphic>
31      </PointSymbolizer>
32    </Rule>
33    <Rule>
34      <Name>Small</Name>
35      <MinScaleDenominator>320000000</MinScaleDenominator>
36      <PointSymbolizer>
37        <Graphic>
38          <Mark>
39            <WellKnownName>circle</WellKnownName>
40            <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.

### 8.2.2 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.

<b>fid</b> (Feature ID)	<b>name</b> (Road name)	<b>type</b> (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>
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 spec-

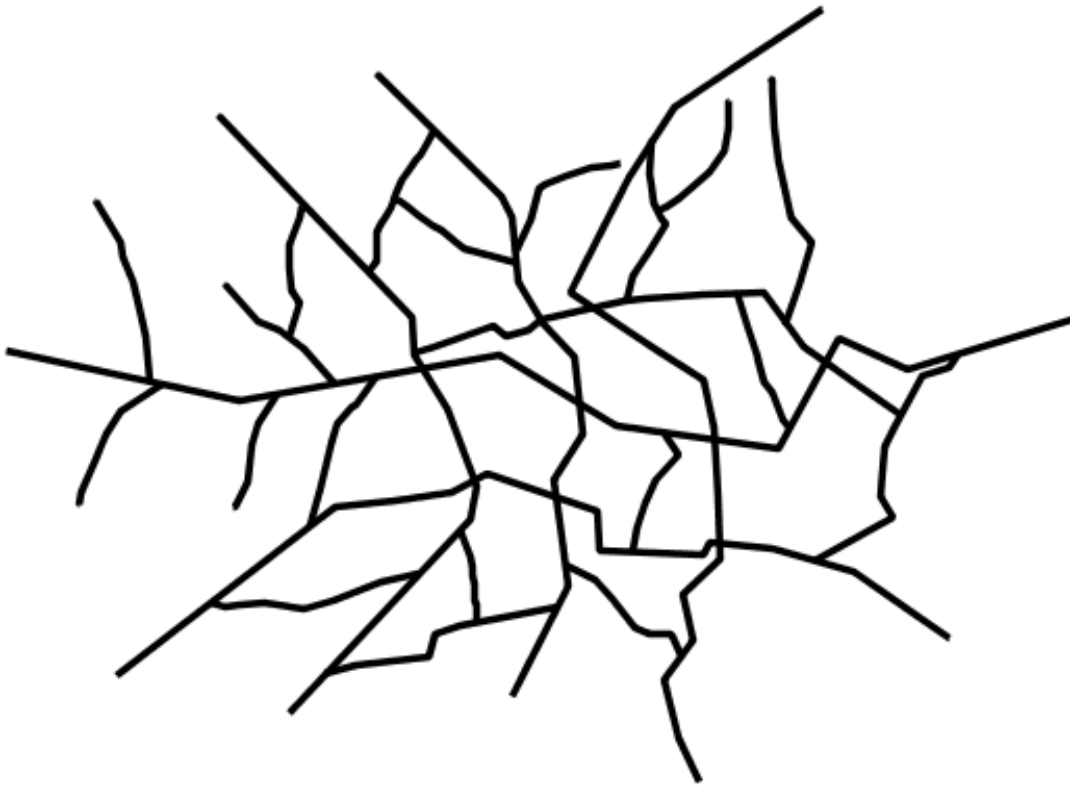


Figure 8.14: *Simple line*



ified.) 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 draws lines with a blue fill of 3 pixels and a gray stroke of 1 pixel.

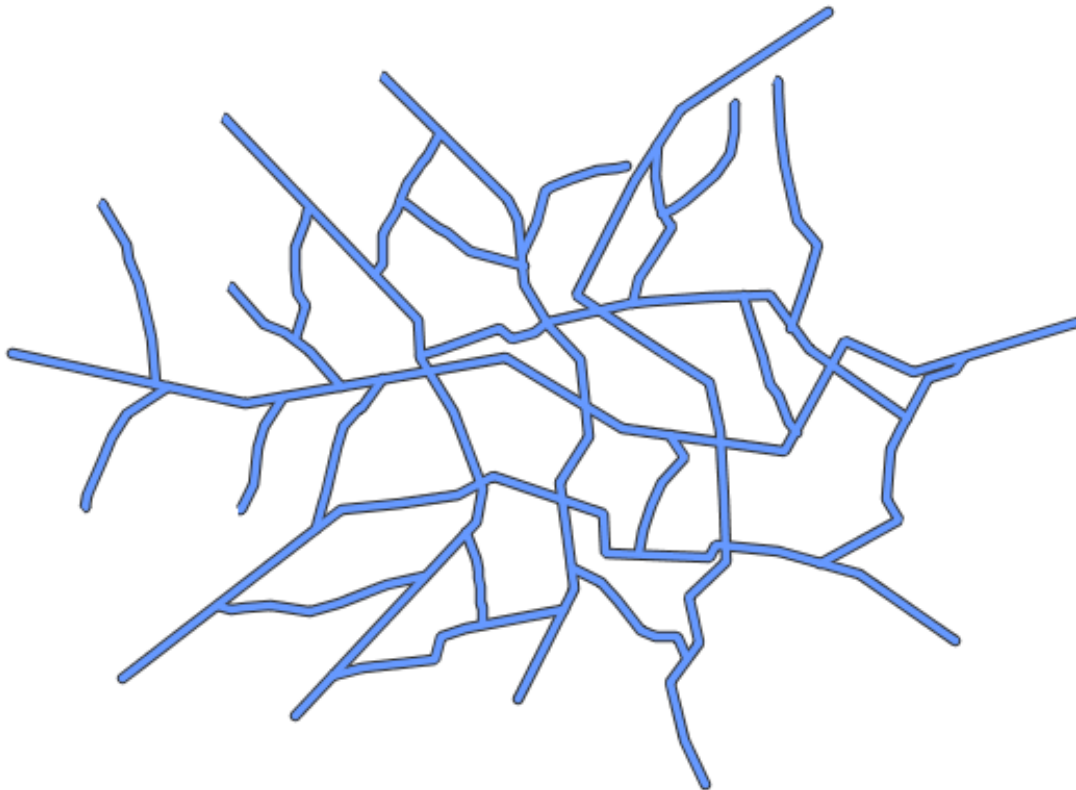


Figure 8.15: *Line with border*

### 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>

```

```
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. In this style, all of the gray lines are drawn first via the first `<FeatureTypeStyle>`, followed by all of the blue lines in a second `<FeatureTypeStyle>`. GeoServer will render every `<FeatureTypeStyle>` in the order that they are presented in the SLD. This not only ensures that the blue lines won’t be obscured by the gray lines, but 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 **line 7** renders the edges of the line to be rounded instead of flat. (When working with lines that have borders, using the `stroke-linecap` parameter ensures that the ends of the lines will have a properly-drawn border.)

**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.

## Code

View and download the full “Dashed line” SLD

```
1  <FeatureTypeStyle>
2    <Rule>
3      <LineSymbolizer>
```

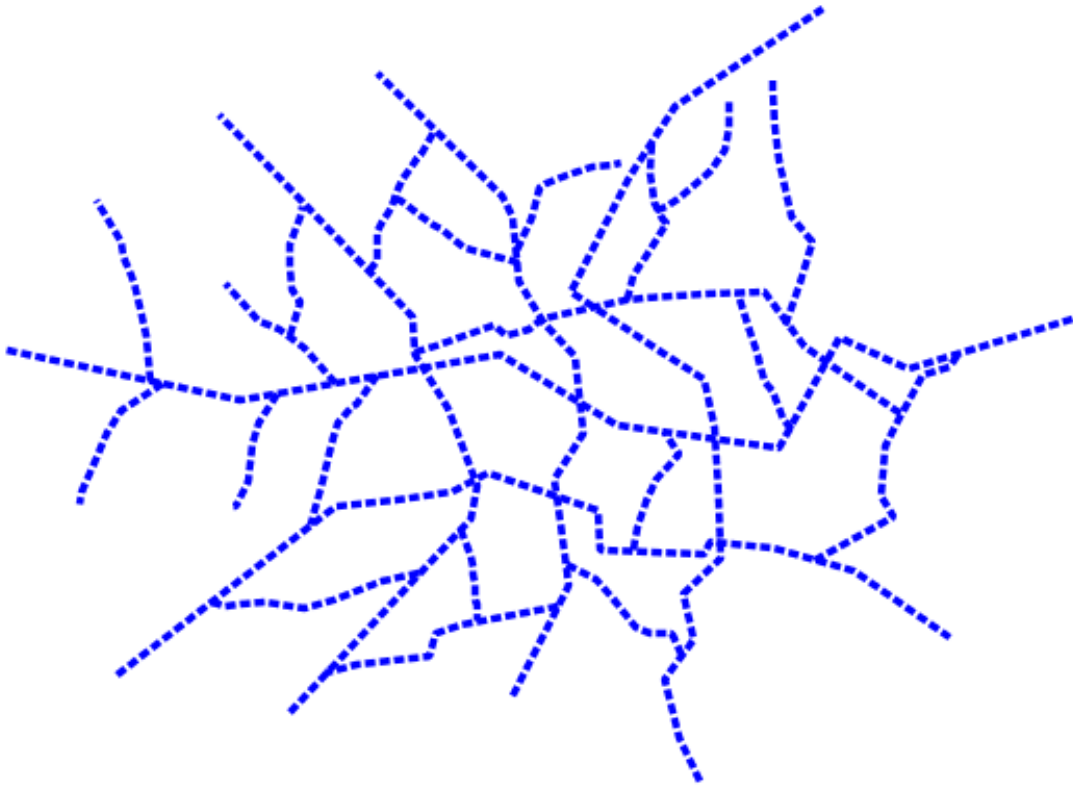


Figure 8.16: *Dashed line*

```
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.

## 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    </Rule>
10   <Rule>
11     <LineSymbolizer>
12       <Stroke>
13         <GraphicStroke>
14           <Graphic>
15             <Mark>
16               <WellKnownName>shape://vertline</WellKnownName>
17             <Stroke>
18               <CssParameter name="stroke">#333333</CssParameter>
19               <CssParameter name="stroke-width">1</CssParameter>
20             </Stroke>
21           </Mark>
22           <Size>12</Size>
23         </Graphic>
24       </GraphicStroke>
25     </Stroke>
26   </LineSymbolizer>
```

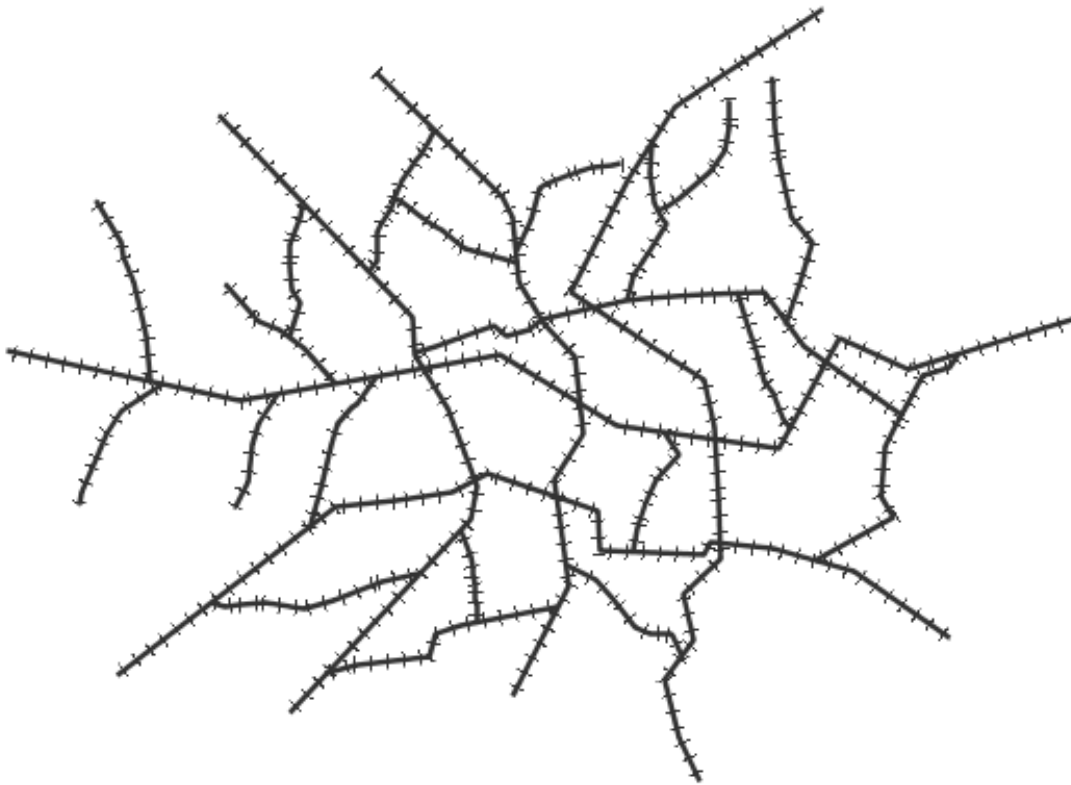


Figure 8.17: *Railroad (hatching)*

```

27     </Rule>
28 </FeatureTypeStyle>

```

## Details

In this example, there are two rules, each containing a `<LineSymbolizer>`. (Each `<LineSymbolizer>` must exist in its own rule.) The first rule, on **lines 2-8**, 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 rule, on **lines 10-27**. **Line 16\*** specifies that the rule draw a vertical line hatch ("`shape://vertline`") perpendicular to the line geometry. **\*\*Lines 18-19** set the hatch color to dark gray (`#333333`) and width to 1 pixel. Finally, **line 22** specifies both the length of the hatch and the distance between each hatch to both be 12 pixels.

### 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 8.18: *Line with default label*

## Code

View and download the full “Line with default label” SLD

```

1      <FeatureTypeStyle>
2          <Rule>
3              <LineSymbolizer>
4                  <Stroke>
5                      <CssParameter name="stroke">#FF0000</CssParameter>
6                  </Stroke>
7              </LineSymbolizer>
8              <TextSymbolizer>
9                  <Label>
10                     <ogc:PropertyName>name</ogc:PropertyName>
11                 </Label>
12                 <Fill>
13                     <CssParameter name="fill">#000000</CssParameter>
14                 </Fill>
15             </TextSymbolizer>
16         </Rule>
17     </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

```

1      <FeatureTypeStyle>
2          <Rule>
3              <LineSymbolizer>
4                  <Stroke>
5                      <CssParameter name="stroke">#FF0000</CssParameter>
6                  </Stroke>
7              </LineSymbolizer>
8              <TextSymbolizer>
9                  <Label>

```



Figure 8.19: *Label following line*



```

10      <ogc:PropertyName>name</ogc:PropertyName>
11    </Label>
12    <Fill>
13      <CssParameter name="fill">#000000</CssParameter>
14    </Fill>
15    <VendorOption name="followLine">true</VendorOption>
16    <LabelPlacement>
17      <LinePlacement />
18    </LabelPlacement>
19  </TextSymbolizer>
20 </Rule>
21 </FeatureTypeStyle>

```

## Details

As the [Line with default label](#) example showed, the default label behavior isn't very 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 [Line with default label](#) example with the exception of **lines 15-18**. **Line 15** sets the option to have the label follow the line, while **lines 16-18** 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.

## Code

View and download the full “Optimized label” SLD

```

1  <FeatureTypeStyle>
2    <Rule>
3      <LineSymbolizer>
4        <Stroke>
5          <CssParameter name="stroke">#FF0000</CssParameter>
6        </Stroke>
7      </LineSymbolizer>
8      <TextSymbolizer>
9        <Label>
10         <ogc:PropertyName>name</ogc:PropertyName>
11       </Label>
12       <Fill>
13         <CssParameter name="fill">#000000</CssParameter>
14       </Fill>
15       <VendorOption name="followLine">true</VendorOption>
16       <VendorOption name="maxAngleDelta">90</VendorOption>

```

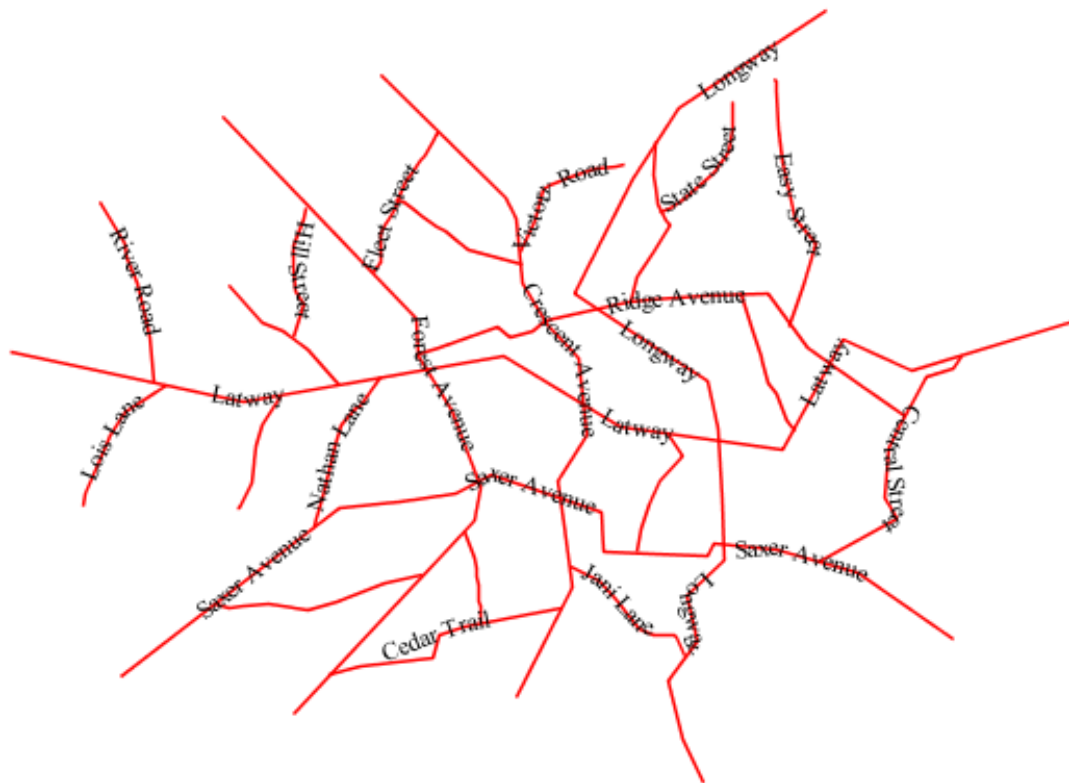


Figure 8.20: *Optimized label*

```

17     <VendorOption name="maxDisplacement">400</VendorOption>
18     <VendorOption name="repeat">150</VendorOption>
19     <LabelPlacement>
20       <LinePlacement />
21     </LabelPlacement>
22   </TextSymbolizer>
23 </Rule>
24 </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 16-18**. **Line 16** 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 17** 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 18** 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.

## 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>
6         </Stroke>
7       </LineSymbolizer>
8       <TextSymbolizer>
9         <Label>
10          <ogc:PropertyName>name</ogc:PropertyName>
11        </Label>
12        <Fill>
13          <CssParameter name="fill">#000000</CssParameter>
14        </Fill>
15        <Font>
16          <CssParameter name="font-family">Arial</CssParameter>
17          <CssParameter name="font-size">10</CssParameter>
18          <CssParameter name="font-style">normal</CssParameter>

```



Figure 8.21: *Optimized and styled label*

```

19         <CssParameter name="font-weight">bold</CssParameter>
20     </Font>
21     <VendorOption name="followLine">true</VendorOption>
22     <VendorOption name="maxAngleDelta">90</VendorOption>
23     <VendorOption name="maxDisplacement">400</VendorOption>
24     <VendorOption name="repeat">150</VendorOption>
25     <LabelPlacement>
26         <LinePlacement />
27     </LabelPlacement>
28 </TextSymbolizer>
29 </Rule>
30 </FeatureTypeStyle>

```

## Details

This example is similar to the [Optimized label placement](#). The only difference is in the font information, which is contained in **lines 15-20**. **Line 16** sets the font family to be “Arial”, **line 17** sets the font size to 10, **line 18** sets the font style to “normal” (as opposed to “italic” or “oblique”), and **line 19** 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

```

1  <FeatureTypeStyle>
2    <Rule>
3      <Name>local-road</Name>
4      <ogc:Filter>
5        <ogc:PropertyIsEqualTo>
6          <ogc:PropertyName>type</ogc:PropertyName>
7          <ogc:Literal>local-road</ogc:Literal>
8        </ogc:PropertyIsEqualTo>
9      </ogc:Filter>
10     <LineSymbolizer>
11       <Stroke>
12         <CssParameter name="stroke">#009933</CssParameter>
13         <CssParameter name="stroke-width">2</CssParameter>
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>

```

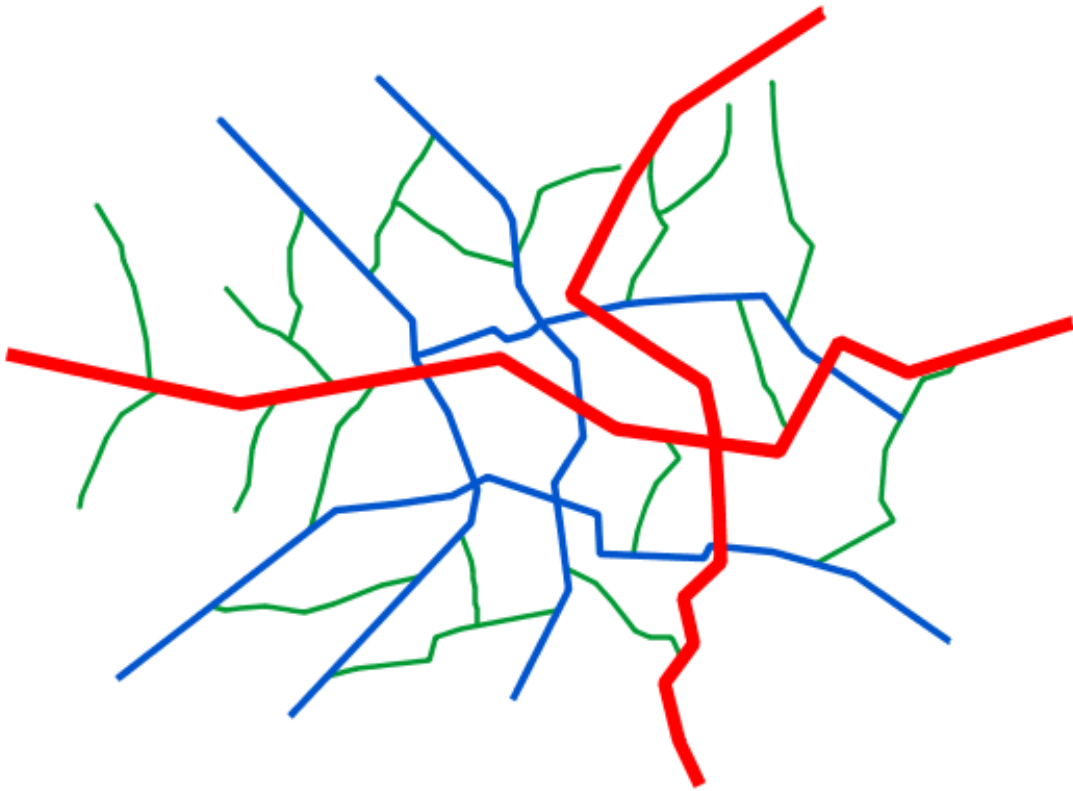


Figure 8.22: *Attribute-based line*

```

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 free-ways: “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.



Figure 8.23: Zoom-based line: Zoomed in

### Code

View and download the full “Zoom-based line” SLD

```
1  <FeatureTypeStyle>
2    <Rule>
3      <Name>Large</Name>
4      <MaxScaleDenominator>180000000</MaxScaleDenominator>
5      <LineSymbolizer>
```



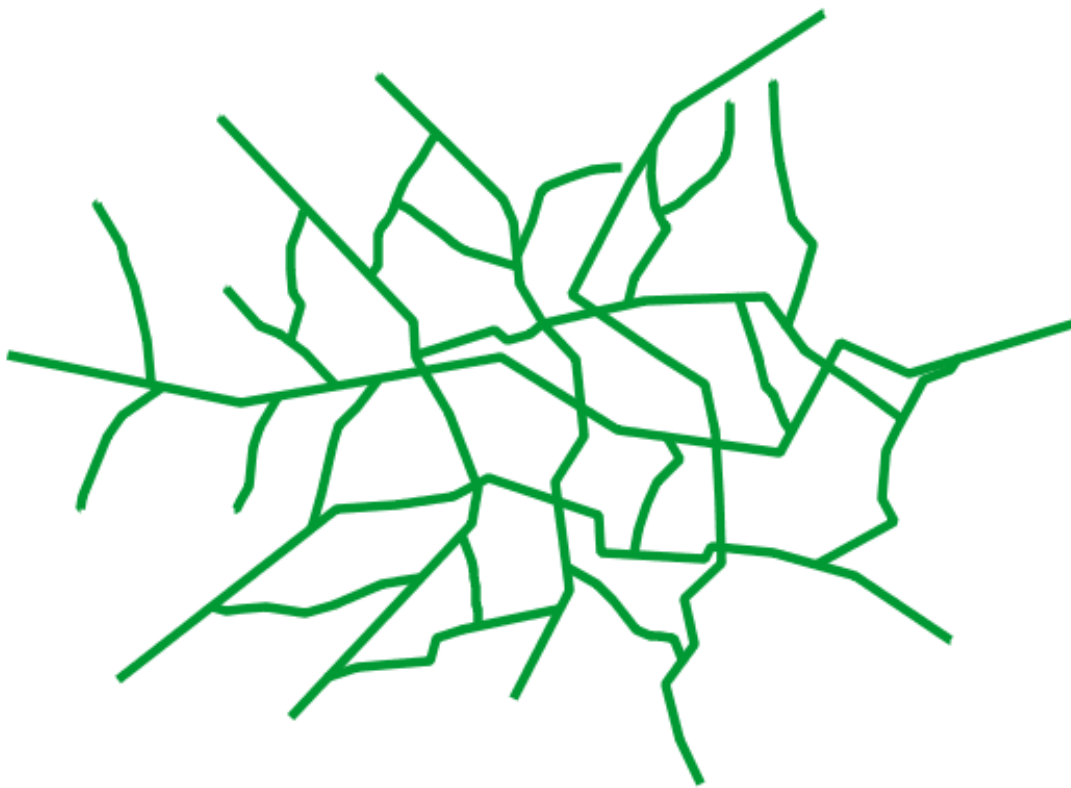


Figure 8.24: *Zoom-based line: Partially zoomed*

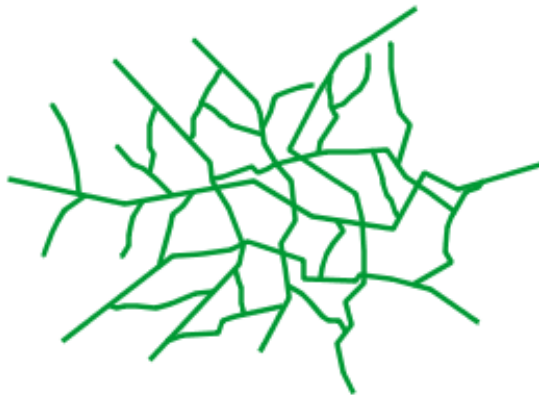


Figure 8.25: *Zoom-based line: Zoomed out*

```

6      <Stroke>
7        <CssParameter name="stroke">#009933</CssParameter>
8        <CssParameter name="stroke-width">6</CssParameter>
9      </Stroke>
10     </LineSymbolizer>
11 </Rule>
12 <Rule>
13   <Name>Medium</Name>
14   <MinScaleDenominator>180000000</MinScaleDenominator>
15   <MaxScaleDenominator>360000000</MaxScaleDenominator>
16   <LineSymbolizer>
17     <Stroke>
18       <CssParameter name="stroke">#009933</CssParameter>
19       <CssParameter name="stroke-width">4</CssParameter>
20     </Stroke>
21   </LineSymbolizer>
22 </Rule>
23 <Rule>
24   <Name>Small</Name>
25   <MinScaleDenominator>360000000</MinScaleDenominator>
26   <LineSymbolizer>
27     <Stroke>
28       <CssParameter name="stroke">#009933</CssParameter>
29       <CssParameter name="stroke-width">2</CssParameter>
30     </Stroke>
31   </LineSymbolizer>
32 </Rule>
33 </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.

## 8.2.3 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.

## 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>
```

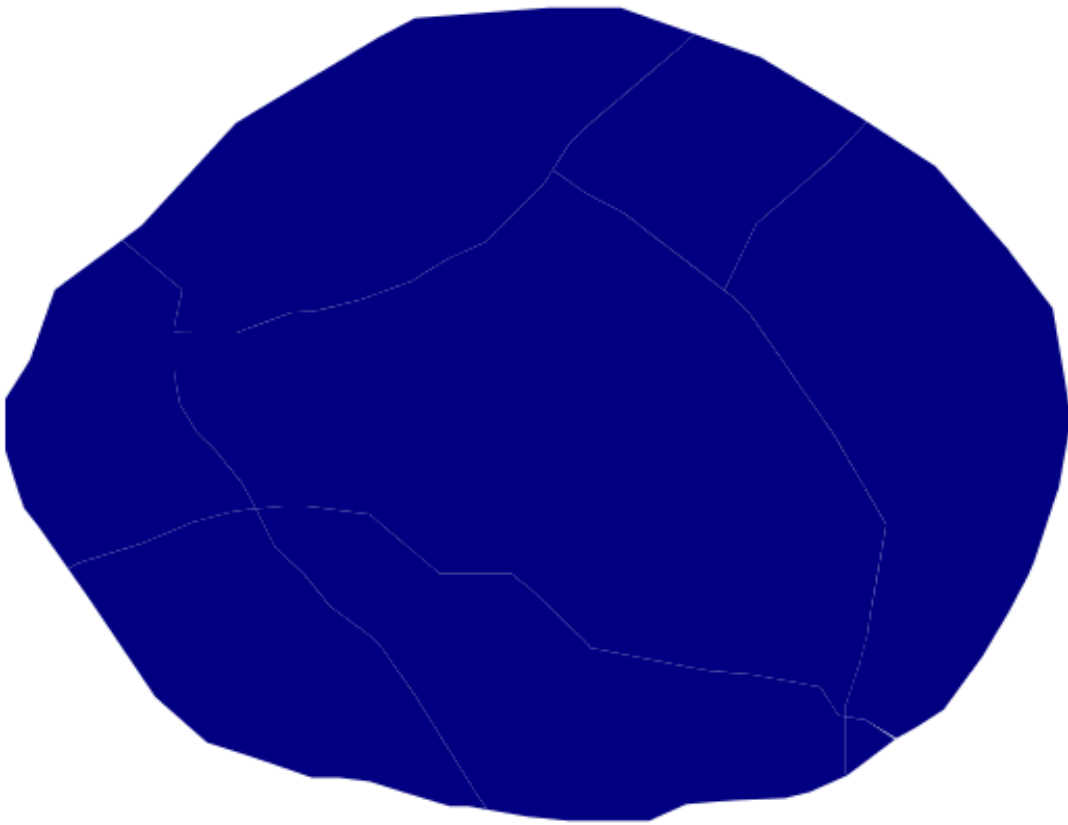


Figure 8.26: *Simple polygon*

```
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.

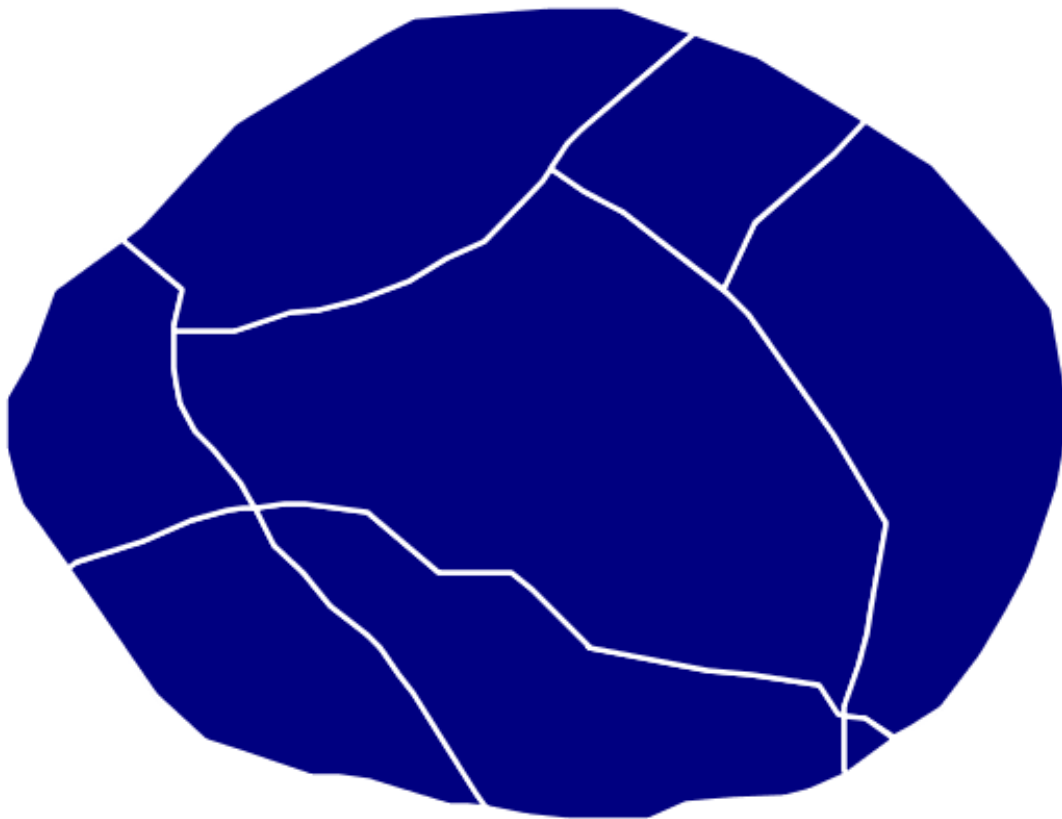


Figure 8.27: *Simple polygon with stroke*

## 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%.

## 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

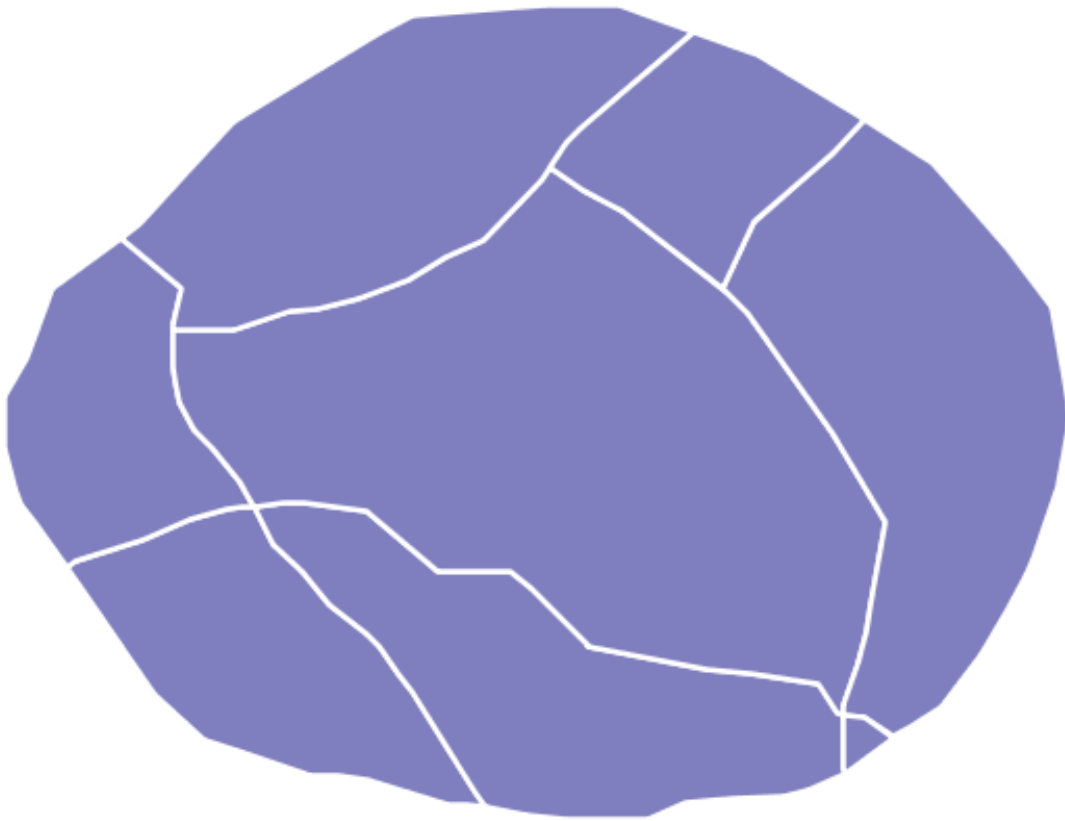


Figure 8.28: *Transparent polygon*



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.

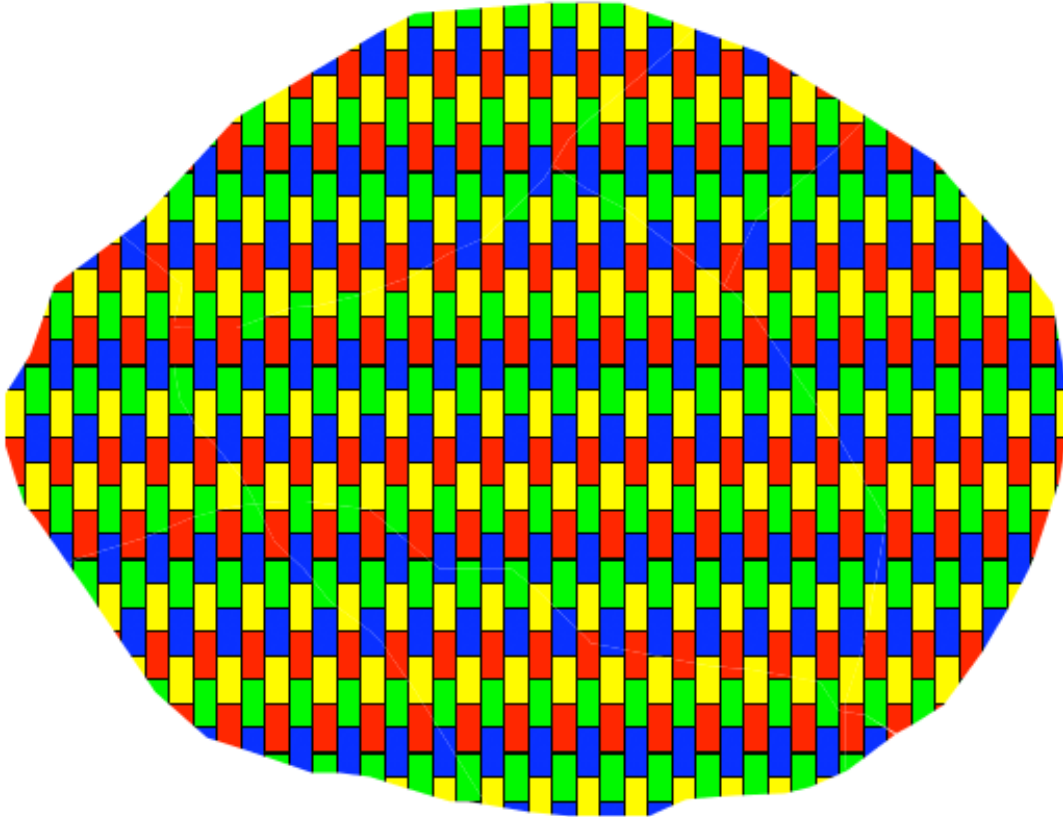


Figure 8.29: *Graphic fill*

### 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 8.30: *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>
```

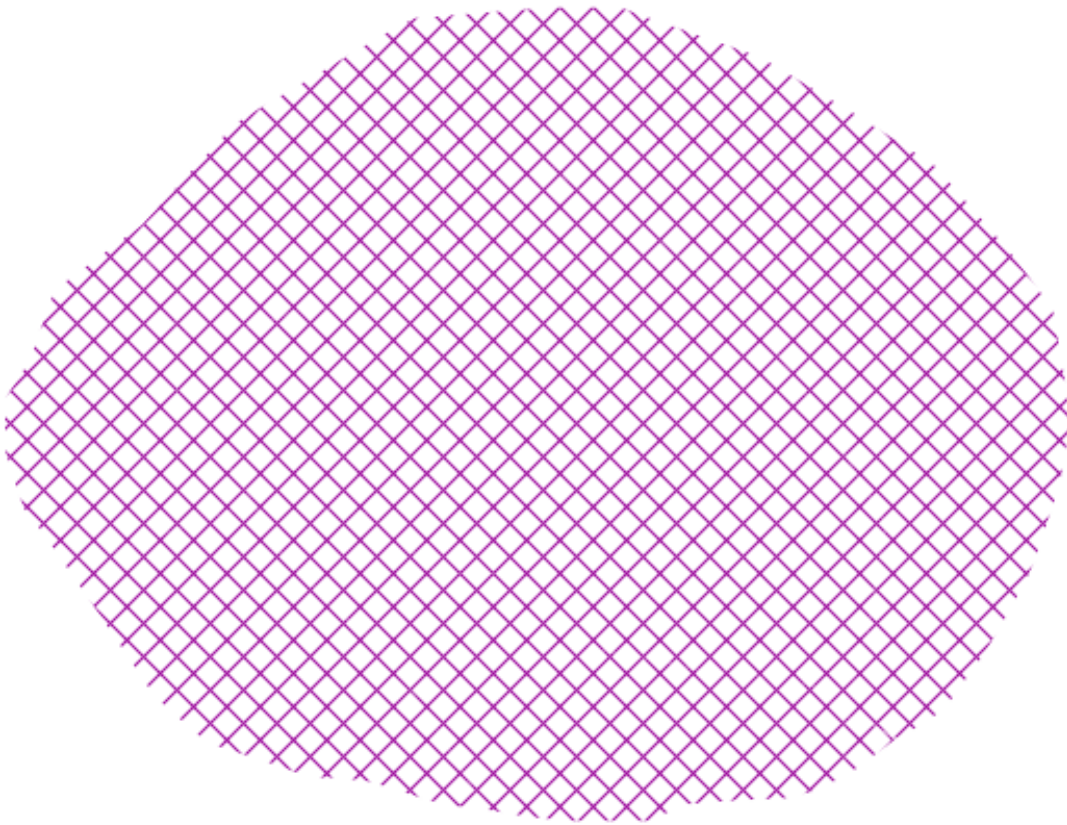


Figure 8.31: *Hatching fill*

```
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>
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>
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>
```



Figure 8.32: *Polygon with default label*

## 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 8.33: *Label halo*

## Code

View and download the full “Label halo” 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       <Halo>
17         <Radius>3</Radius>
18         <Fill>
19           <CssParameter name="fill">#FFFFFF</CssParameter>
20         </Fill>
21       </Halo>
22     </TextSymbolizer>
23   </Rule>
24 </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.

## 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>

```



Figure 8.34: *Polygon with styled label*



```

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.

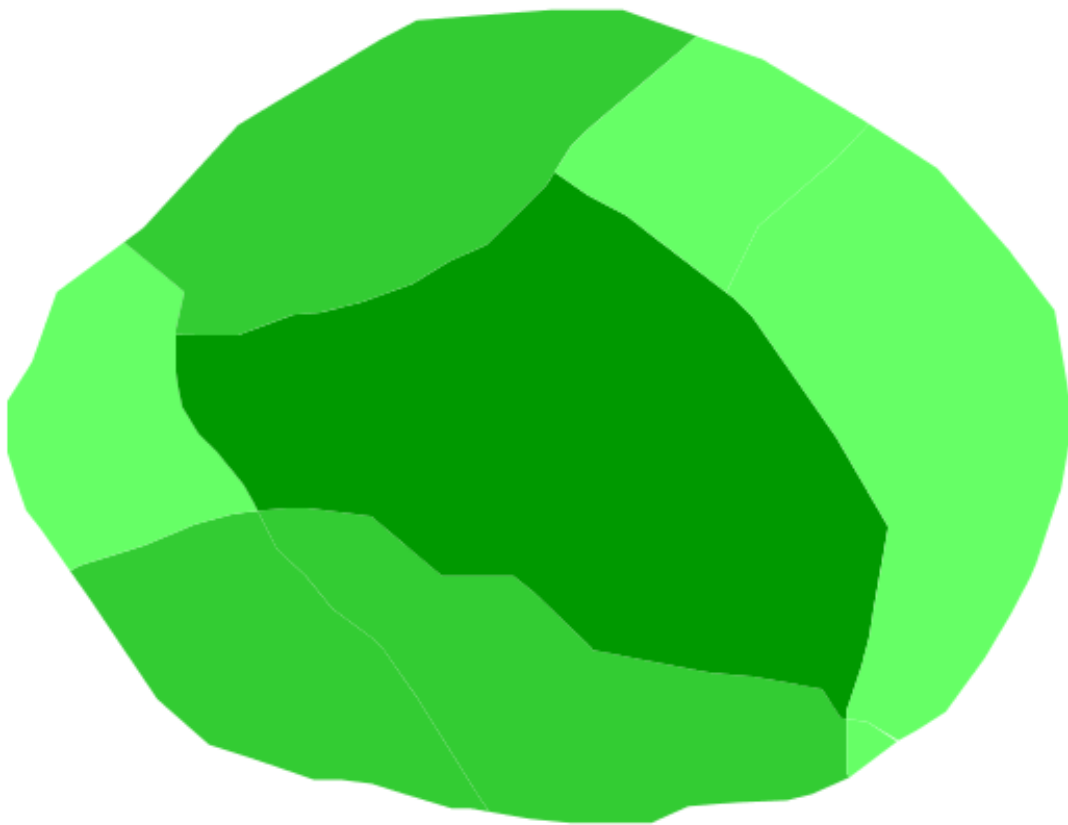


Figure 8.35: *Attribute-based polygon*

## Code

View and download the full “Attribute-based polygon” SLD

```

1  <FeatureTypeStyle>
2    <Rule>
3      <Name>SmallPop</Name>
4      <Title>Less Than 200,000</Title>
5      <ogc:Filter>
6        <ogc:PropertyIsLessThan>
7          <ogc:PropertyName>pop</ogc:PropertyName>
8          <ogc:Literal>200000</ogc:Literal>
9        </ogc:PropertyIsLessThan>
10     </ogc:Filter>
11     <PolygonSymbolizer>
12       <Fill>
13         <CssParameter name="fill">#66FF66</CssParameter>
14       </Fill>
15     </PolygonSymbolizer>
16   </Rule>
17   <Rule>
18     <Name>MediumPop</Name>
19     <Title>200,000 to 500,000</Title>
20     <ogc:Filter>
21       <ogc:And>
22         <ogc:PropertyIsGreaterThanOrEqualTo>
23           <ogc:PropertyName>pop</ogc:PropertyName>
24           <ogc:Literal>200000</ogc:Literal>
25         </ogc:PropertyIsGreaterThanOrEqualTo>
26         <ogc:PropertyIsLessThan>
27           <ogc:PropertyName>pop</ogc:PropertyName>
28           <ogc:Literal>500000</ogc:Literal>
29         </ogc:PropertyIsLessThan>
30       </ogc:And>
31     </ogc:Filter>
32     <PolygonSymbolizer>
33       <Fill>
34         <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.

## Code

View and download the full “Zoom-based polygon” SLD

```
1      <FeatureTypeStyle>
2      <Rule>
3          <Name>Large</Name>
4          <MaxScaleDenominator>10000000</MaxScaleDenominator>
5          <PolygonSymbolizer>
6              <Fill>
7                  <CssParameter name="fill">#0000CC</CssParameter>
8              </Fill>
9              <Stroke>
10                 <CssParameter name="stroke">#000000</CssParameter>
11                 <CssParameter name="stroke-width">7</CssParameter>
12             </Stroke>
13         </PolygonSymbolizer>
14         <TextSymbolizer>
```



Figure 8.36: *Zoom-based polygon: Zoomed in*

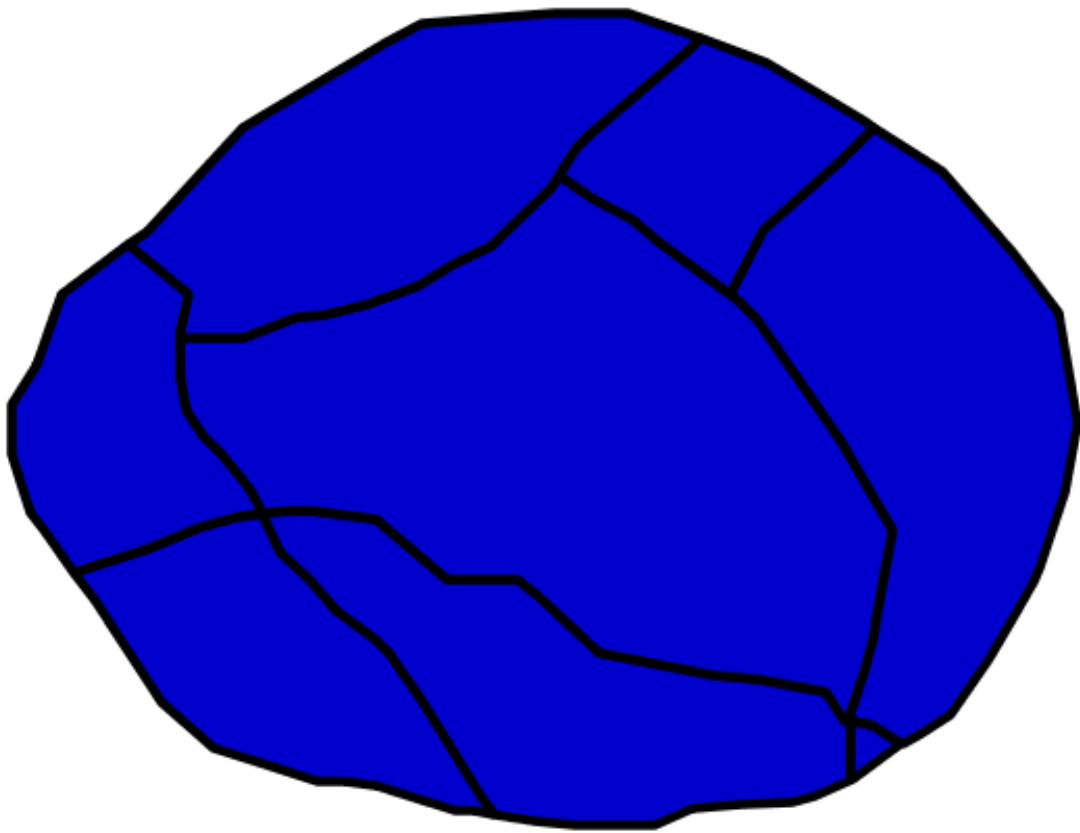


Figure 8.37: *Zoom-based polygon: Partially zoomed*

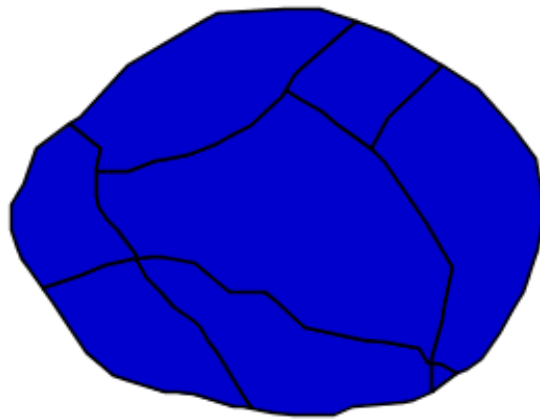


Figure 8.38: *Zoom-based polygon: Zoomed out*

```
15     <Label>
16       <ogc:PropertyName>name</ogc:PropertyName>
17     </Label>
18     <Font>
19       <CssParameter name="font-family">Arial</CssParameter>
20       <CssParameter name="font-size">14</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>0.5</AnchorPointY>
29         </AnchorPoint>
30       </PointPlacement>
31     </LabelPlacement>
32     <Fill>
33       <CssParameter name="fill">#FFFFFF</CssParameter>
34     </Fill>
35   </TextSymbolizer>
36 </Rule>
37 <Rule>
38   <Name>Medium</Name>
39   <MinScaleDenominator>10000000</MinScaleDenominator>
40   <MaxScaleDenominator>20000000</MaxScaleDenominator>
41   <PolygonSymbolizer>
42     <Fill>
43       <CssParameter name="fill">#0000CC</CssParameter>
44     </Fill>
45     <Stroke>
46       <CssParameter name="stroke">#000000</CssParameter>
47       <CssParameter name="stroke-width">4</CssParameter>
48     </Stroke>
49   </PolygonSymbolizer>
50 </Rule>
51 <Rule>
52   <Name>Small</Name>
53   <MinScaleDenominator>20000000</MinScaleDenominator>
54   <PolygonSymbolizer>
55     <Fill>
56       <CssParameter name="fill">#0000CC</CssParameter>
57     </Fill>
58     <Stroke>
59       <CssParameter name="stroke">#000000</CssParameter>
60       <CssParameter name="stroke-width">1</CssParameter>
61     </Stroke>
62   </PolygonSymbolizer>
63 </Rule>
64 </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 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.

## 8.2.4 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:



Figure 8.39: *Raster file as rendered in grayscale*

## Download the raster shapefile

### Two-color gradient

This example shows a two-color style with green at lower elevations and brown at higher elevations.

## 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>
```

Figure 8.40: *Two-color gradient*

## 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.

## 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" />

```

Figure 8.41: *Transparent gradient*

```
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.

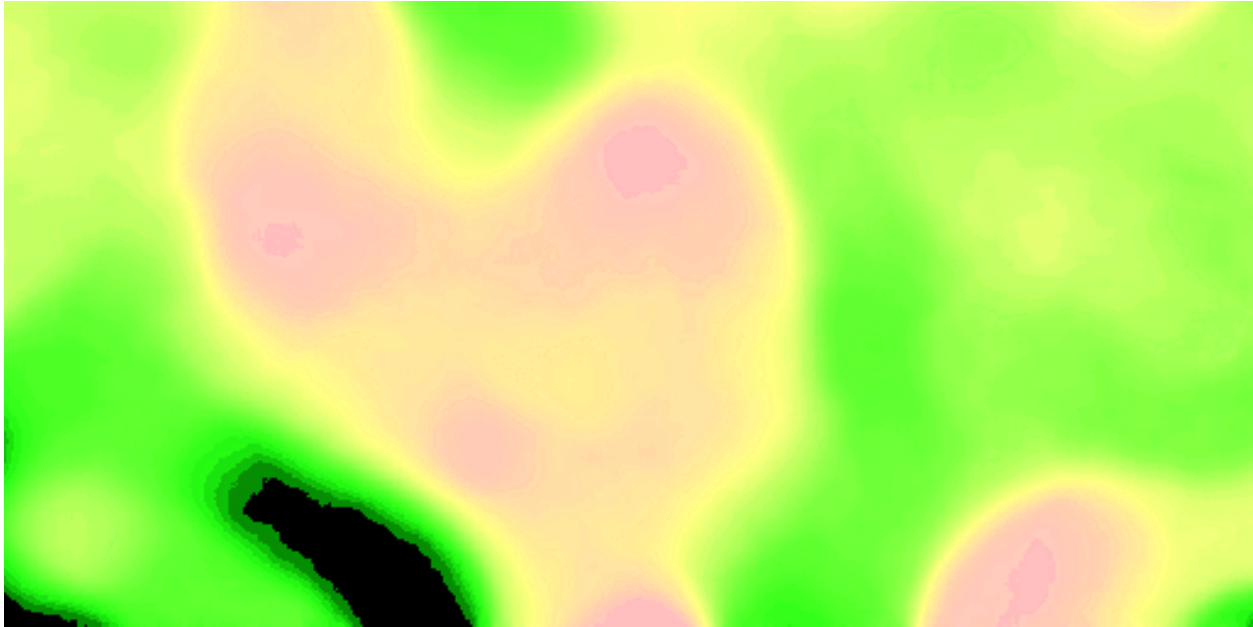
## Brightness and contrast

This example normalizes the color output and then increases the brightness by a factor of 2.

## Code

View and download the full “Brightness and contrast” SLD

```
1 <FeatureTypeStyle>
2   <Rule>
3     <RasterSymbolizer>
4       <ContrastEnhancement>
```

Figure 8.42: *Brightness and contrast*

```

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>

```

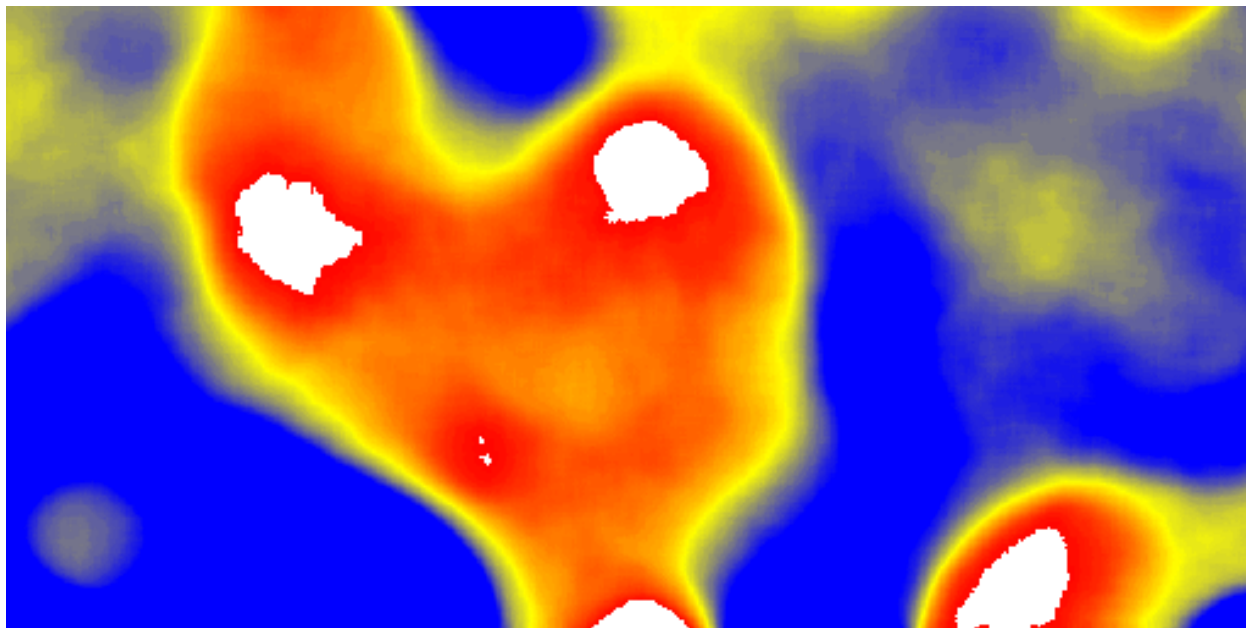
## Details

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 8.43: *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.

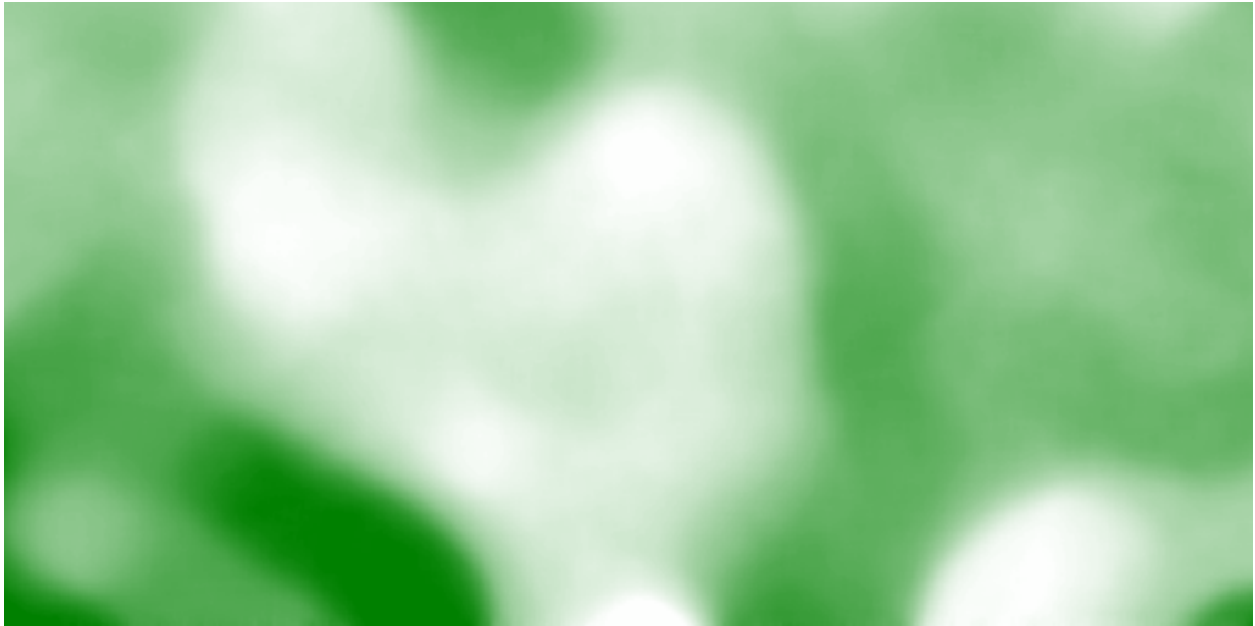


Figure 8.44: *Alpha channel*

## Code

View and download the full “Alpha channel” SLD

```

1      <FeatureTypeStyle>
2      <Rule>
3      <RasterSymbolizer>
4      <ColorMap>
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.



Figure 8.45: *Discrete colors*

### 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" />
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 an eight color gradient.

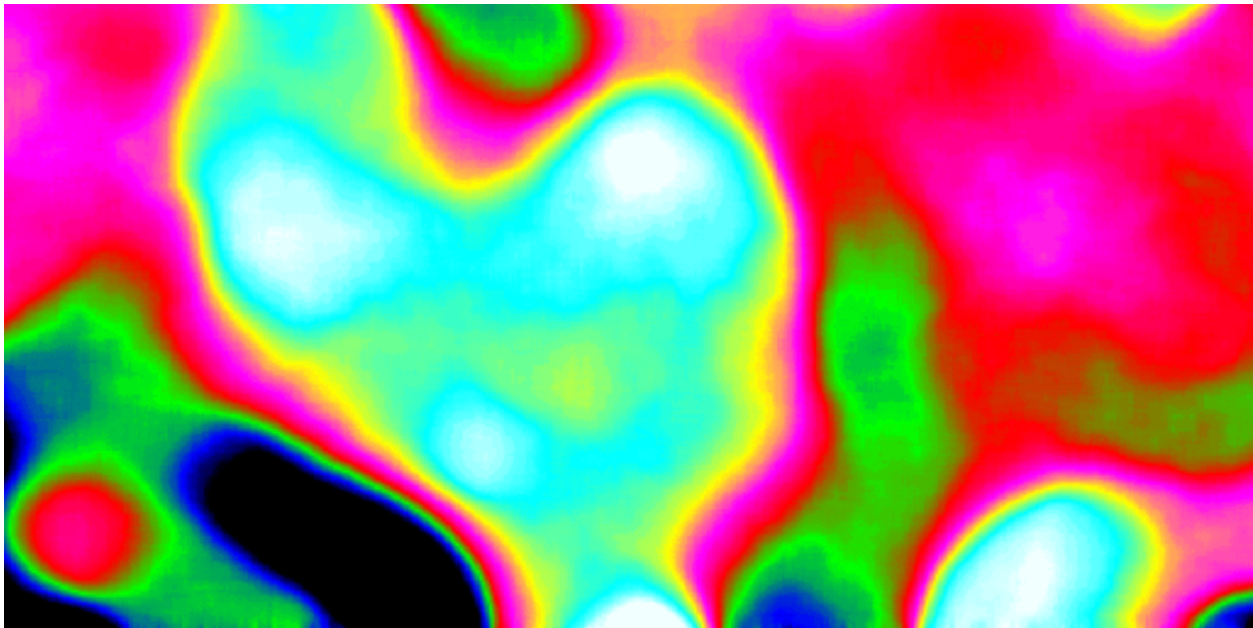


Figure 8.46: *Many color gradient*

## 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

There is no limit on the amount of entries that can be contained in a `<ColorMap>` (**lines 4-13**). This example has eight entries:

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

## 8.3 SLD Reference

A symbolizer specifies how data should be visualized. There are 5 types of symbolizers: `PointSymbolizer`, which is used to portray **point** data; `LineSymbolizer`, which is used to portray **line** data; `PolygonSymbolizer`, which is used to portray **polygon** data; `RasterSymbolizer`, which is used to portray **raster** data; and `TextSymbolizer`, which is used to portray **text labels**.

**Warning:** Intro for filters and scale.

### 8.3.1 PointSymbolizer

The `PointSymbolizer` styles **points**, Points are elements that contain only position information.

#### Syntax

The outermost element is the `<Graphic>` tag. This determines the type of visualization. There are five possible tags to include inside the `<Graphic>` tag:

Tag	Required?	Description
<ExternalGraphic>	No (when using <Mark>)	Specifies an image file to use as the symbolizer.
<Mark>	No (when using <ExternalGraphic>)	Specifies a common shape to use as the symbolizer.
<Opacity>	No	Determines the opacity (transparency) of symbolizers. Values range from 0 (completely transparent) to 1 (completely opaque). Default is 1.
<Size>	Yes	Determines the size of the symbolizer in pixels. When used with an image file, this will specify the height of the image, with the width scaled accordingly.
<Rotation>	No	Determines the rotation of the graphic in degrees. The rotation increases in the clockwise direction. Negative values indicate counter-clockwise rotation. Default is 0.

Within the <ExternalGraphic> tag, there are also additional tags:

Tag	Required?	Description
<OnlineResource>	Yes	The location of the image file. Can be either a URL or a local path relative to the SLD.
<Format>	Yes	The MIME type of the image format. Most standard web image formats are supported.

Within the <Mark> tag, there are also additional tags:

Tag	Required?	Description
<WellKnownName>	Yes	The name of the common shape. Options are circle, square, triangle, star, cross, or x. Default is square.
<Fill>	No (when using <Stroke>)	Specifies how the symbolizer should be filled. Options are a <CssParameter name="fill"> specifying a color in the form #RRGGBB, or <GraphicFill> for a repeated graphic.
<Stroke>	No (when using <Fill>)	Specifies how the symbolizer should be drawn on its border. Options are a <CssParameter name="fill"> specifying a color in the form #RRGGBB or <GraphicStroke> for a repeated graphic.

## Example

Consider the following symbolizer taken from the Simple Point example in the [Points](#) section in the [SLD Cookbook](#).

```

1      <PointSymbolizer>
2          <Graphic>
3              <Mark>
4                  <WellKnownName>circle</WellKnownName>
5                  <Fill>
6                      <CssParameter name="fill">#FF0000</CssParameter>
7                  </Fill>
8              </Mark>
9              <Size>6</Size>
10         </Graphic>
11     </PointSymbolizer>
```

The symbolizer contains a <Graphic> tag, which is required. Inside this tag is the <Mark> tag and <Size> tag, which are the minimum required tags inside <Graphic> (when not using the <ExternalGraphic>

tag). The `<Mark>` tag contains the `<WellKnownName>` tag and a `<Fill>` tag. No other tags are required. In summary, this example specifies the following:

1. Data 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

Further examples can be found in the [Points](#) section of the *SLD Cookbook*.

## 8.3.2 LineSymbolizer

The LineSymbolizer styles **lines**. Lines are one-dimensional geometry elements that contain position and length. Lines can be comprised of multiple line segments.

### Syntax

The outermost tag is the `<Stroke>` tag. This tag is required, and determines the visualization of the line. There are three possible tags that can be included inside the `<Stroke>` tag.

Tag	Required?	Description
<code>&lt;GraphicFill&gt;</code>	No	Renders the pixels of the line with a repeated pattern.
<code>&lt;GraphicStroke&gt;</code>	No	Renders the line with a repeated linear graphic.
<code>&lt;CssParameter&gt;</code>	No	Determines the stroke styling parameters.

When using the `<GraphicStroke>` and `<GraphicFill>` tags, it is required to insert the `<Graphic>` tag inside them. The syntax for this tag is identical to that mentioned in the [PointSymbolizer](#) section above.

Within the `<CssParameter>` tag, there are also additional parameters that go inside the actual tag:

Parameter	Required?	Description
<code>name="stroke"</code>	No	Specifies the solid color given to the line, in the form #RRGGBB. Default is black (#000000).
<code>name="stroke-width"</code>	No	Specifies the width of the line in pixels. Default is 1.
<code>name="stroke-opacity"</code>	No	Specifies the opacity (transparency) of the line. possible values are between 0 (completely transparent) and 1 (completely opaque). Default is 1.
<code>name="stroke-linejoin"</code>	No	Determines how lines are rendered at intersections of line segments. Possible values are <code>mitre</code> (sharp corner), <code>round</code> (rounded corner), and <code>bevel</code> (diagonal corner). Default is <code>mitre</code> .
<code>name="stroke-linecap"</code>	No	Determines how lines are rendered at ends of line segments. Possible values are <code>butt</code> (sharp square edge), <code>round</code> (rounded edge), and <code>square</code> (slightly elongated square edge). Default is <code>butt</code> .
<code>name="stroke-dasharray"</code>	No	Encodes a dash pattern as a series of numbers separated by spaces. Odd-indexed numbers (first, third, etc) determine the length in pixels 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.
<code>name="stroke-dashoffset"</code>	No	Specifies the distance in pixels into the <code>dasharray</code> pattern at which to start drawing. Default is 0.

**Warning:** Maybe a screenshot of the different linecaps etc?

## Example

### 8.3.3 PolygonSymbolizer

The LineSymbolizer styles **polygons**. Lines are two-dimensional geometry elements. They can contain styling information about their border (stroke) and their fill.

## Syntax

A <PolygonSymbolizer> can have two outermost tags:

Tag	Required?	Description
<Fill>	No (when using <Stroke>)	Determines the styling for the fill of the polygon.
<Stroke>	No (when using <Fill>)	Determines the styling for the stroke of the polygon.

The details for the <Stroke> tag are identical to that mentioned in the [LineSymbolizer](#) section above.

Within the <Fill> tag, there are additional tags:

Tag	Required?	Description
<GraphicFill>	No	Renders the fill of the polygon with a repeated pattern.
<CssParameter>	No	Determines the fill styling parameters.

When using the <GraphicFill> tag, it is required to insert the <Graphic> tag inside it. The syntax for this tag is identical to that mentioned in the [PointSymbolizer](#) section above.

Within the <CssParameter> tag, there are also additional parameters that go inside the actual tag:

Parameter	Required?	Description
name="fill"	No	Specifies the fill color for the polygon, in the form #RRGGBB. Default is grey (#808080).
name="fill-opacity"	No	Specifies the opacity (transparency) of the fill of the polygon. Possible values are between 0 (completely transparent) and 1 (completely opaque). Default is 1.

## Example

Consider the following symbolizer taken from the Simple Point example in 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> tag. Inside this tag is a <CssParameter> that specifies a fill color for the polygon to be #000080, or a muted blue.

Further examples can be found in the [Polygons](#) section of the [SLD Cookbook](#).

### 8.3.4 Raster Symbolizer

#### Introduction

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 lots of different attributes, 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>`.

#### Elements and Syntax

The following elements are available to be used as parameters inside `<RasterSymbolizer>`.

- `<Opacity>`
- `<ColorMap>`
- `<ChannelSelection>`
- `<ContrastEnhancement>`
- `<ShadedRelief>`
- `<OverlapBehavior>`
- `<ImageOutline>`

Notice that not all the above are actually implemented in the current version of the GeoServer.

#### Opacity

This element sets the transparency level for the entire dataset. As is standard, the values range from zero (0) to one (1), with zero being totally transparent, and one being not transparent at all. The syntax for `<Opacity>` is very simple:

```
<Opacity>0.5</Opacity>
```

where, in this case, the raster would be displayed at 50% opacity.

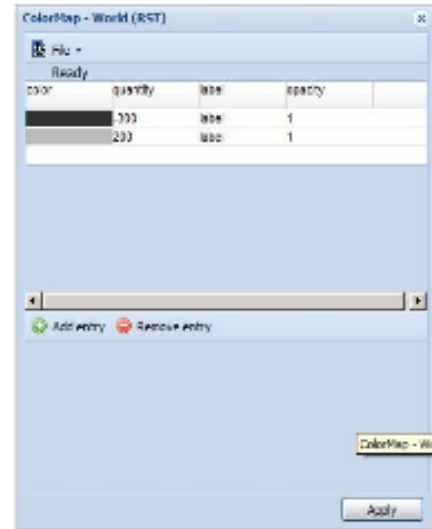
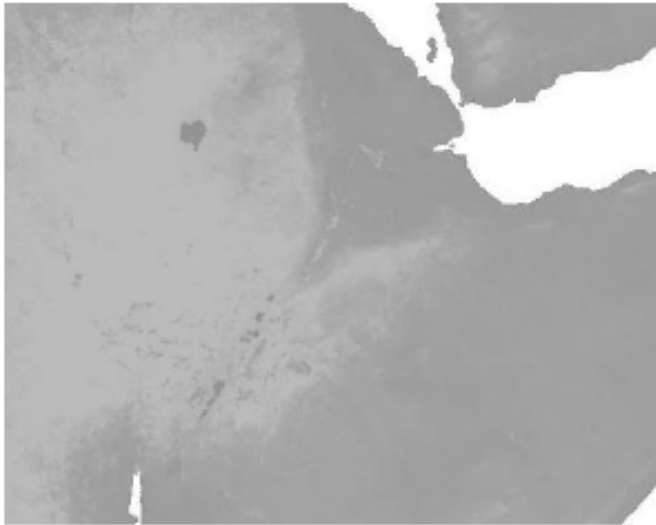
#### ColorMap

The `<ColorMap>` element sets rules for color gradients based on the quantity attribute. This quantity refers to the magnitude of the value of a data point. At its simplest, one could create two color map entries (the element called `<ColorMapEntry>`, one with a color for the “bottom” of the dataset, and another with a color for the “top” of the dataset. The colors in between will be automatically interpolated with the quantity values in between, making creating color gradients easy. One can also fine tune the color map by adding additional entries, which is handy if the dataset has more discrete values rather than a gradient. In that case, one could add an entry for each value to be set to a different color. In all cases, the color is denoted in standard hexadecimal RGB format (`#RRGGBB`). In addition to color and quantity, `ColorMapEntry` elements

can also have opacity and label, the former which could be used instead of the global value mentioned previously, and the latter which could be used for legends.

For example a simple ColorMap can be:

```
<ColorMap>
  <ColorMapEntry color="#323232" quantity="-300" label="label1" opacity="1"/>
  <ColorMapEntry color="#BBBBBB" quantity="200" label="label2" opacity="1"/>
</ColorMap>
```



This example would create a color gradient from #323232 color to #BBBBBB color using quantity values -300 to 200:

```
<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>
```

This example would create a color gradient from #FFCC32 color through #BBBBBB color running through #3645CC color and #CC3636 color. Here, though, #FFCC32 color would be transparent (simulating an alpha channel). Notice that default opacity, when not specified, is 1, which means opaque.

Two attributes can be created in ColorMap root node like 'type' and 'extended'.

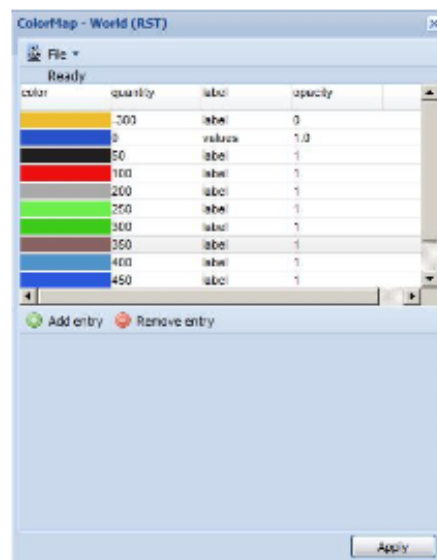
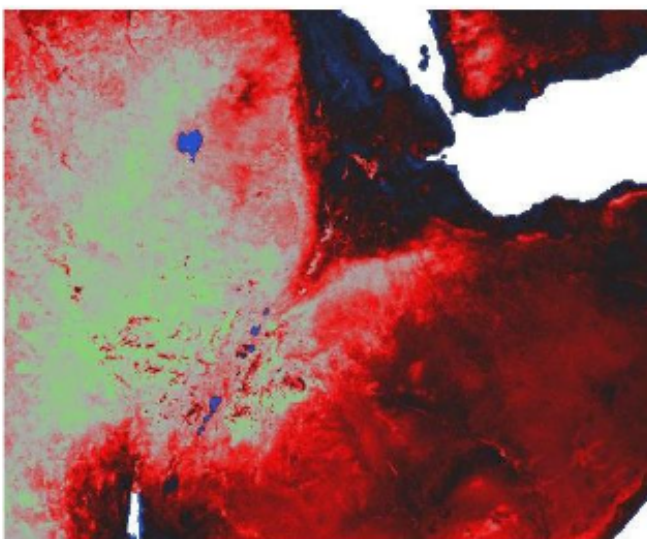
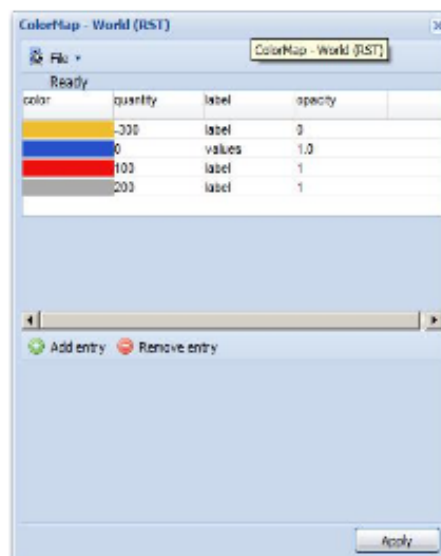
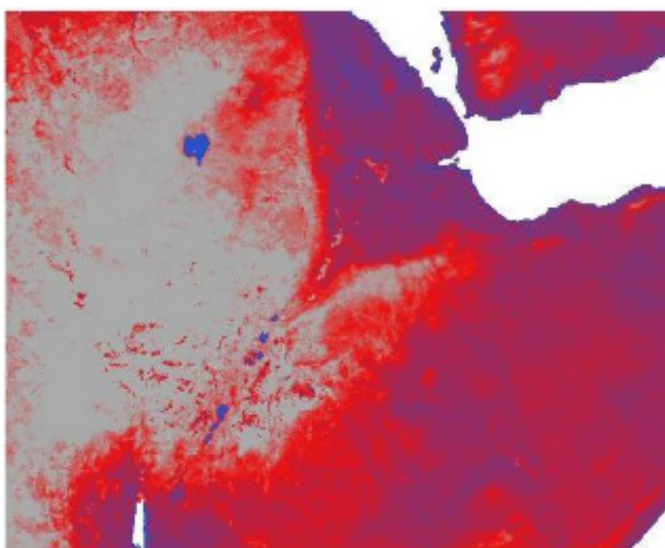
The 'type' attribute specifies the kind of ColorMap to use. There are three different types of ColorMaps that can be specified through this attribute: ramp, intervals and values.

The 'ramp' is the default ColorMap type and the outcome is like the one presented at the beginning of this section (if into the ColorMap tag the attribute 'type' is not specified, the default value is 'ramp').

The 'values' means that only the specified entry quantities will be rendered, i.e. no color interpolation is applied between the entries.

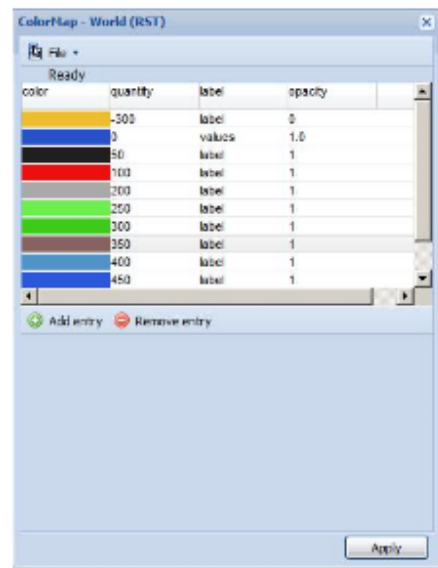
The following example can clarify this aspect:







```
<ColorMap type="values">
  <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>
```



The 'intervals' value means that every interval defined by two entries will be colored using the value of the first entry, i.e. no color interpolation is applied between the intervals:

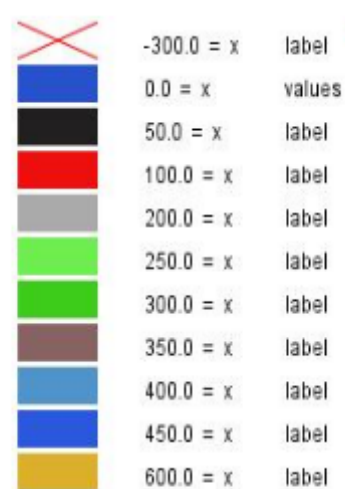
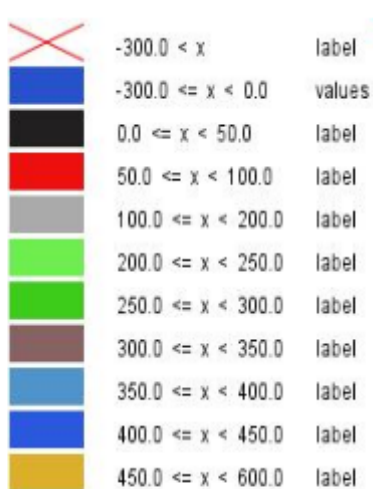
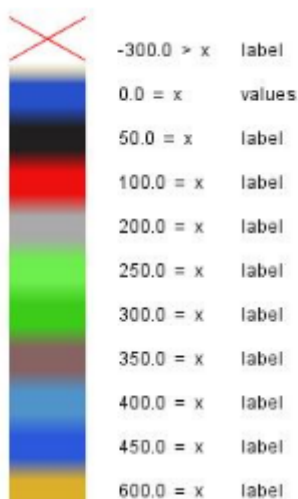
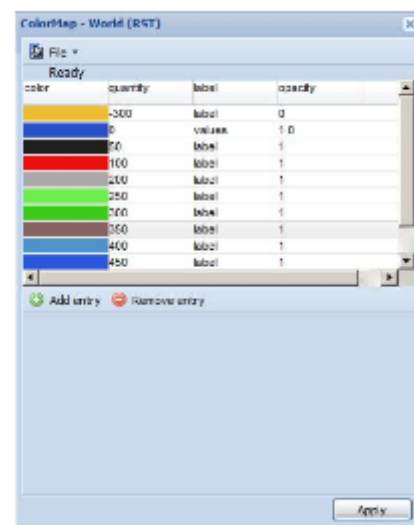
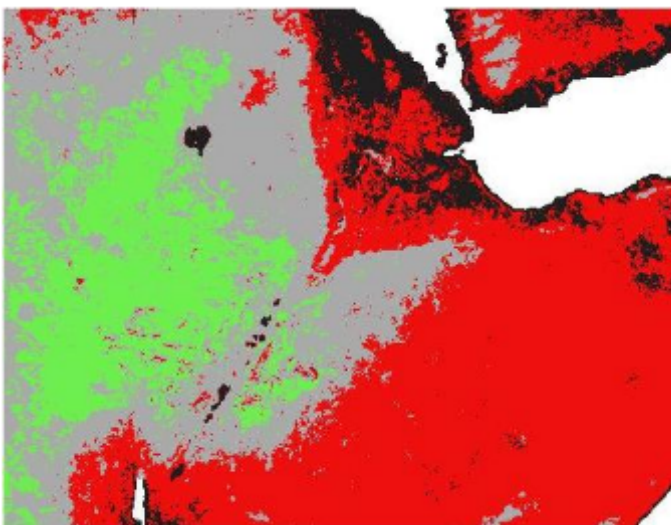
```
<ColorMap type="intervals" extended="true">
  <ColorMapEntry color="#EEBE2F" quantity="-300" label="label" opacity="0"/>
  ...
  <ColorMapEntry color="#DDB02C" quantity="600" label="label" opacity="1"/>
</ColorMap>
```

The 'extended' attribute allows ColorMap to compute gradients using 256 or 65536 colors; extended=false means that the color scale is calculated on 8 bit, else 16 bit if the value is true.

The difference between ramp, values and intervals values is also visible into raster legend. In order to get the raster legend from GeoServer the typically request is:

<http://localhost:8080/geoserver/wms?REQUEST=GetLegendGraphic&VERSION=1.0.0&STYLE=raster100&FORMAT=ir>

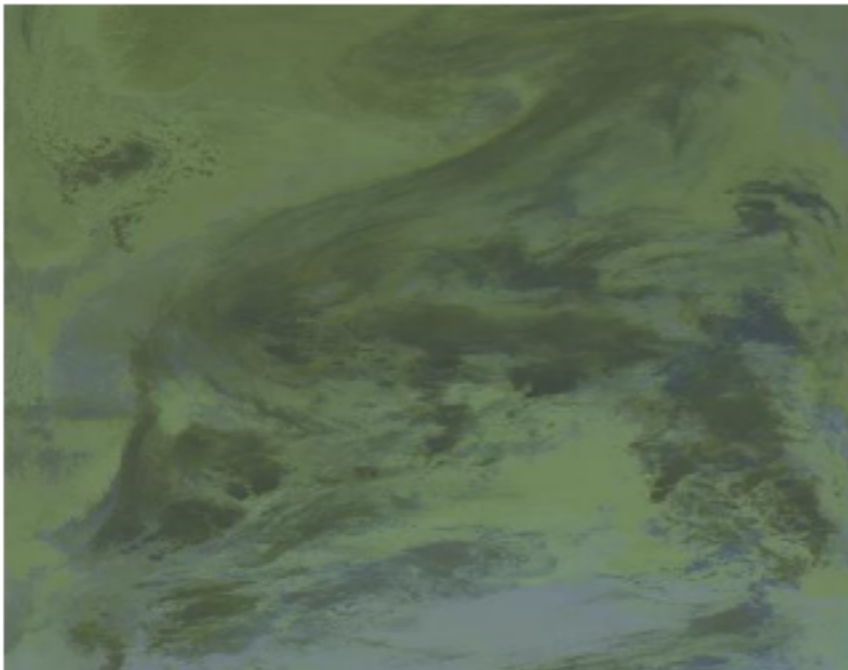
the results are:



## ChannelSelection

This element specifies which color channel to access in the dataset. A dataset may contain standard three-channel colors (red, green, and blue channels) or one grayscale channel. Using `<ChannelSelection>` allows the mapping of a dataset channel to either a red, green, blue, or gray channel:

```
<ChannelSelection>
  <RedChannel>
    <SourceChannelName>1</SourceChannelName>
  </RedChannel>
  <GreenChannel>
    <SourceChannelName>2</SourceChannelName>
  </GreenChannel>
  <BlueChannel>
    <SourceChannelName>3</SourceChannelName>
  </BlueChannel>
</ChannelSelection>
```

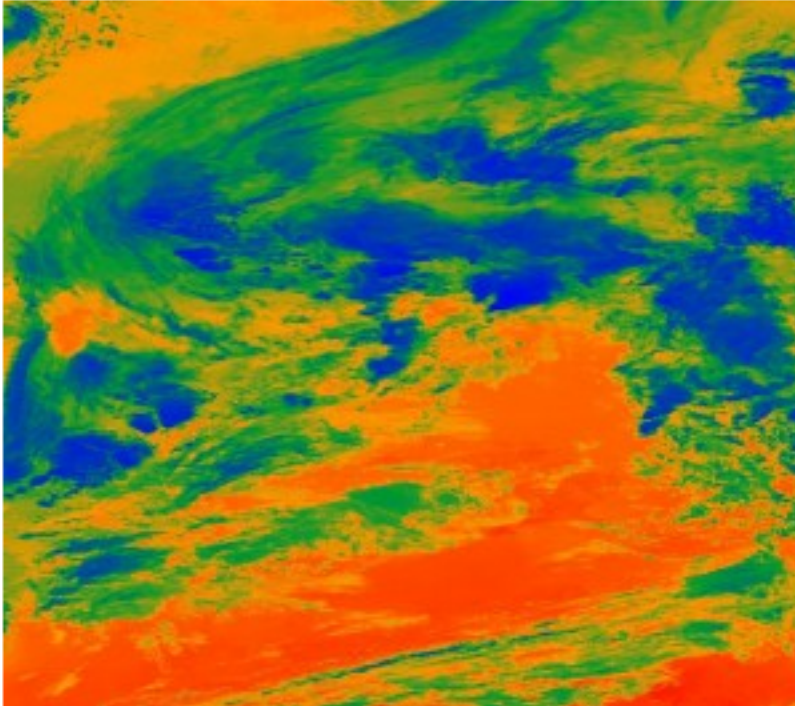


The above would map source channels 1, 2, and 3 to the red, green, and blue Channels, respectively.

This is the result of gray ChannelSelection operation applied to an RGB image and re-colored through 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 in color channels to adjust the relative brightness of the data in that 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. Finally, GammaValue is a scaling factor that adjusts the brightness of the data, with a value less than one (1) darkening the image, and a value greater than one (1) brightening it. (Normalize and Histogram do not have any parameters.) One can use `<ContrastEnhancement>` on a specific channel (say red only) as opposed to globally, if it is desired. In this way, different enhancements can be used on each channel:

```
<ContrastEnhancement>
  <Normalize/>
</ContrastEnhancement>
```

```
<ContrastEnhancement>
  <Histogram/>
</ContrastEnhancement>
```

These examples turn on Normalize and Histogram, respectively:

```
<ContrastEnhancement>
  <GammaValue>2</GammaValue>
</ContrastEnhancement>
```

The above increases the brightness of the data by a factor of two.

## ShadedRelief

**Warning:** Support for this elements 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 elements 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 elements 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.

## 8.3.5 TextSymbolizer

The `TextSymbolizer` specifies **text labels**.

### Syntax

A `<TextSymbolizer>` contains the following tags:

Tag	Required?	Description
<code>&lt;Label&gt;</code>	Yes	Specifies the content of the text label
<code>&lt;Font&gt;</code>	No	Specifies the font information for the labels.
<code>&lt;LabelPlacement&gt;</code>	No	Sets the position of the label relative its associate feature.
<code>&lt;Halo&gt;</code>	No	Creates a colored background around the text label, for low contrast situations.
<code>&lt;Fill&gt;</code>	No	Determines the fill color of the text label.

Each of the above tags have additional sub tags. For the `<Label>` tag:

Tag	Required?	Description



Within the `<Font>` tag, the `<CssParameter>` tag is the only tag included. There are four types of parameters for this tag, each included inside the `<CssParameter>` tag:

Parameter	Required?	Description
<code>name="font-family"</code>	No	Determines the family name of the font to use for the label. Default is Times.
<code>name="font-style"</code>	No	Determines the style of the font. Options are normal, italic, and oblique. Default is normal.
<code>name="font-weight"</code>	No	Determines the weight of the font. Options are normal and bold. Default is normal.
<code>name="font-size"</code>	No	Determines the size of the font in pixels. Default is 10.

Within the `<LabelPlacement>` tag, there are many tags for specifying the placement of the label:

Tag	Required?	Description

Within the `<Halo>` tag, there are two tags to specify the details of the halo:

Tag	Required?	Description
<code>&lt;Radius&gt;</code>	No	Sets the size of the halo radius in pixels. Default is 1.
<code>&lt;Fill&gt;</code>	No	Sets the color of the halo in the form of #RRGGBB. Default is white (#FFFFFF).

The `<Fill>` tag is identical to that described in the WHERE~? above.

## Example

### 8.3.6 Labeling

#### Controlling Label Placement

Controlling where the WMS server places labels with SLD is bit complex. The SLD specification only defines the most basic way of controlling placement explicitly says that defining more control is “a real can of worms”. Geoserver fully supports the SLD specification plus adds a few extra parameters so you can make pretty maps.

#### Basic SLD Placement

The SLD specification indicates two types of LabelPlacement:

- for Point Geometries (“PointPlacement”)
- for Linear (line) geometries (“LinePlacement”)

**Note:** Relative to Where?

See below for the actual algorithm details, but:

- Polygons are intersected with the viewport and the centroid is used.
- Lines are intersected with the viewport and the middle of the line is used.

## Code

```
<xsd:element name="PointPlacement">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="sld:AnchorPoint" minOccurs="0"/>
      <xsd:element ref="sld:Displacement" minOccurs="0"/>
      <xsd:element ref="sld:Rotation" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
...
<xsd:element name="LinePlacement">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="sld:PerpendicularOffset" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

### PointPlacement

When you use a `<PointPlacement>` element, the geometry you are labeling will be reduced to a single point (usually the “middle” of the geometry - see algorithm below for details). You can control where the label is relative to this point using the options:

Option	Meaning (Name)
Anchor-Point	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	This is in PIXELS and lets you fine-tune the location of the label.
Rotation	This is the clockwise rotation of the label in degrees.

The best way to understand these is with examples:

### AnchorPoint

The anchor point determines where the label is placed relative to the label point. These measurements are relative to the bounding box of the label. The (x,y) location inside the label’s bounding box (specified by the `AnchorPoint`) is placed at the label point.

The anchor point is defined relative to the label’s bounding box. The bottom left is (0,0), the top left is (1,1), and the middle is (0.5,0.5).

```
<PointPlacement>
  <AnchorPoint>
    <AnchorPointX>
      0.5
    </AnchorPointX>
    <AnchorPointY>
```



```

    0.5
  </AnchorPointY>
</AnchorPoint>
</PointPlacement>

```

By changing the values, you can control where the label is placed.

(x=0,y=0.5) DEFAULT - place the label to the right of the label point

(x=0.5,y=0.5) - place the centre of the label at the label point

(x=1,y=0.5) - place the label to the left of the label point

(x=0.5,y=0) - place the label centered above the label point

## Displacement

Displacement allows fine control of the placement of the label. The displacement values are in pixels and simply move the location of the label on the resulting image.

```

<PointPlacement>
  <Displacement>
    <DisplacementX>
      10
    </DisplacementX>
    <DisplacementY>
      0
    </DisplacementY>
  </Displacement>
</PointPlacement>

```

displacement of x=10 pixels, compare with anchor point (x=0,y=0.5) above

displacement of y=-10 pixels, compare with anchor point (x=0.5,y=1.0) not shown

## Rotation

Rotation is simple - it rotates the label clockwise the number of degrees you specify. See the examples below for how it interacts with AnchorPoints and displacements.

```

<Rotation>
  45
</Rotation>

```

simple 45 degrees rotation

45 degrees rotation with anchor point (x=0.5,y=0.5)

45 degrees with 40 pixel X displacement

45 degrees rotation with 40 pixel Y displacement with anchor point (x=0.5,y=0.5)

## LinePlacement

When you are labeling a line (i.e. a road or river), you can specify a `<LinePlacement>` element. This tells the labeling system two things: (a) that you want Geoserver to determine the best rotation and placement for the label (b) a minor option to control how the label is placed relative to the line.

The line placement option is very simple - it only allows you to move a label up-and-down from a line.

```
<xs:element name="LinePlacement">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="sld:PerpendicularOffset" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
...
<xs:element name="PerpendicularOffset" type="sld:ParameterValueType"/>
```

This is very similar to the `DisplacementY` option (see above).

```
<LabelPlacement>
  <LinePlacement>
    <PerpendicularOffset>
      10
    </PerpendicularOffset>
  </LinePlacement>
</LabelPlacement>
```

PerpendicularOffset=0

PerpendicularOffset=10 pixels

## Composing labels from multiple attributes

The `<Label>` element in `TextSymbolizer` is said to be mixed, that is, its content can be a mixture of plain text and OGC Expressions. The mix gets interpreted as a concatenation, this means you can leverage it to get 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 such as **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 mandated 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 Specific Enhanced Options

The following options are all extensions of the SLD specification. Using these options gives much more control over how the map looks, since the SLD standard isn't expressive enough to handle all the options one might want. In time we hope to have them be an official part of the specification.

### Priority Labeling (<Priority>)

GeoServer has extended the standard SLD to also include priority labeling. This allows you to control which labels are rendered in preference to other labels.

For example, lets assume you have a data set like this:

City Name	population
Yonkers	197,818
Jersey City	237,681
Newark	280,123
New York	8,107,916

Most people don't know where "Yonkers" city is, but do know where "New York" city is. On our map, we want to give "New York" priority so its more likely to be labeled when it's in conflict (overlapping) "Yonkers".

#### Note: Standard SLD Behavior

If you do not have a <Priority> tag in your SLD then you get the default SLD labeling behavior. This basically means that if there's a conflict between two labels, there is no 'dispute' mechanism and its random which label will be displayed.

In our TextSymbolizer we can put an Expression to retrieve or calculate the priority for each feature:

```
<Priority>
  <PropertyName>population</PropertyName>
</Priority>
```

Location of the cities (see population data above)

New York is labeled in preference to the less populated cities. Without priority labeling, "Yonkers" could be labeled in preference to New York, making a difficult to interpret map.

Notice that larger cities are more readily named than smaller cities.

## Grouping Geometries (<VendorOption name="group">)

Sometimes you will have a set of related features that you only want a single label for. The grouping option groups all features with the same label text, then finds a representative geometry for the group.

Roads data is an obvious example - you only want a single label for all of "main street", not a label for every piece of "main street."

When the grouping option is off (default), grouping is not performed and each geometry is labeled (space permitting).

With the grouping option on, all the geometries with the same label are grouped together and the label position is determined from ALL the geometries.

Geometry	Representative Geometry
Point Set	first point inside the view rectangle is used.
Line Set	lines are (a) networked together (b) clipped to the view rectangle (c) middle of the longest network path is used.
Polygon Set	polygons are (a) clipped to the view rectangle (b) the centroid of the largest polygon is used.

```
<VendorOption name="group">yes</VendorOption>
```

**Warning:** Watch out - you could group together two sets of features by accident. For example, you could create a single group for "Paris" which contains features for Paris (France) and Paris (Texas).

## Overlapping and Separating Labels (<VendorOption name="spaceAround">)

By default geoserver will not put labels "on top of each other". By using the spaceAround option you can allow labels to overlap and you can also add extra space around a label.

```
<VendorOption name="spaceAround">10</VendorOption>
```

Default behavior ("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 spaceAround value of 10 for all TextSymbolizers, each label will be 20 pixels apart from each other (see below).

**NOTE:** the value you specify (an integer in pixels) actually provides 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 distance between them will be 8. For two labels in the first symbolizer ("5") they will each be 5 pixels apart from each other, for a total of 10 pixels!

**Note:** Interaction with different values in different TextSymbolizers

You can have multiple TextSymbolizers in your SLD file, each with a different spaceAround option. This will normally do what you would think if all your spaceAround options are  $\geq 0$ . 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 too 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 place the following in your `<TextSymbolizer>`.

```
<VendorOption name="followLine">true</VendorOption>
```

It is required to use `<LinePlacement>` along with this option to ensure that all labels are correctly following the lines:

```
<LabelPlacement>  
  <LinePlacement/>  
</LabelPlacement>
```

## maxDisplacement

The **maxDisplacement** option controls the displacement of the label along a line. Normally GeoServer would label a line at its center point only, provided the location is not busy with another label, and not label it at all otherwise. When set, the labeller will search for another location within **maxDisplacement** pixels from the pre-computed label point.

When used in conjunction with **repeat**, the value for **maxDisplacement** should always be lower than the value for **repeat**.

```
<VendorOption name="maxDisplacement">10</VendorOption>
```

## repeat

The **repeat** option determines how often GeoServer labels a line. Normally GeoServer would label each line only once, regardless of their length. Specify a positive value to make it draw the label every **repeat** pixels.

```
<VendorOption name="repeat">100</VendorOption>
```

## labelAllGroup

The **labelAllGroup** option makes sure that all of the segments in a line group are labeled instead of just the longest one.

```
<VendorOption name="labelAllGroup">true</VendorOption>
```

## maxAngleDelta

Designed to use 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 value, given in pixels. Make sure to give a dimension wide enough to accommodate the longest word other wise this option will split words over multiple lines.

```
<VendorOption name="autoWrap">50</VendorOption>
```

## forceLeftToRight

The labeller always tries to draw labels so that they can be read, meaning the label does not always follow the line orientation, but sometimes it's flipped 180° instead to allow for normal reading. This may get in the way if the label is a directional arrow, and you're trying to show one way directions (assuming the geometry is oriented along the one way, and that you have a flag to discern one ways from streets with both circulations).

The following setting disables label flipping, making the label always follow the natural orientation of the line being labelled:

```
<VendorOption name="forceLeftToRighth">>false</VendorOption>
```

## conflictResolution

By default labels are subjected to conflict resolution, meaning the renderer will not allow any label to overlap with a label that has been drawn already. Setting this parameter to false pull the label out of the conflict resolution game, meaning the label will be drawn even if it overlaps with other labels, and other labels drawn after it won't mind overlapping with it.

```
<VendorOption name="conflictResolution">>false</VendorOption>
```

## Goodness of Fit

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 <code>LabelCacheDefault#goodnessOfFit()</code> )

The default value is 0.5, but it can be modified using:

```
<VendorOption name="goodnessOfFit">0.3</VendorOption>
```

## Polygon alignment

GeoServer normally tries to place horizontal labels within a polygon, and give up in case the label position is busy or if the label does not fit enough in the polygon. This options allows GeoServer to try alternate rotations for the labels.

```
<VendorOption name="polygonAlign">mbr</VendorOption>
```

Option	Description
manual	The default value, only the rotation manually specified in the <code>&lt;Rotation&gt;</code> tag will be used
ortho	If the label does not fit horizontally and the polygon is taller than wider the 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

### 8.3.7 Filters

A *filter* is the mechanism in SLD for specifying predicates. Similar in nature to a “WHERE” clause in SQL, filters are the language for specifying which styles should be applied to which features in a data set.

The filter language used by SLD is itself an [OGC standard](#) defined in the Filter Encoding specification freely available.

A filter is used to select a subset of features of a dataset to apply a symbolizer to.

There are three types of filters:

#### Attribute filters

Attribute filters are used to constrain the non-spatial attributes of a feature. Example

```
1 <PropertyIsEqualTo>
2   <PropertyName>NAME</PropertyName>
3   <Literal>Bob</Literal>
4 </PropertyIsEqualTo>
```

The above filter selects those features which have a {{NAME}} attribute which has a value of “Bob”. A variety of equality operators are available:

- PropertyIsEqualTo
- PropertyIsNotEqualTo
- PropertyIsLessThan
- PropertyIsLessThanOrEqualTo
- PropertyIsGreaterThan
- PropertyIsGreaterThanOrEqualTo
- PropertyIsBetween

## Spatial filters

Spatial filters used to constrain the spatial attributes of a feature. Example

```
1 <Intersects>
2   <PropertyName>GEOMETRY</PropertyName>
3   <Literal>
4     <gml:Point>
5       <gml:coordinates>1 1</gml:coordinates>
6     </gml:Point>
7   </Literal>
8 </Intersects>
```

The above filter selects those features with a geometry that intersects the point (1,1). A variety of spatial operators are available:

- Intersects
- Equals
- Disjoint
- Within
- Overlaps
- Crosses
- DWithin
- Beyond
- Distance

## Logical filters

Logical filters are used to create combinations of filters using the logical operators And, Or, and Not. Example



```

1 <And>
2   <PropertyIsEqualTo>
3     <PropertyName>NAME</PropertyName>
4     <Literal>Bob</Literal>
5   </PropertyIsEqualTo>
6   <Intersects>
7     <PropertyName>GEOMETRY</PropertyName>
8     <Literal>
9       <gml:Point>
10        <gml:coordinates>1 1</gml:coordinates>
11      </gml:Point>
12    </Literal>
13  </Intersects>
14 </And>

```

## Rules

A *rule* combines a number of symbolizers with a filter to define the portrayal of a feature. Consider the following example:

```

<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>

```

The above rule applies only to features which have a POPULATION attribute greater than 100,000 and symbolizes then with a red point.

An SLD document can contain many rules. Multiple rule SLD's are the basis for "thematic styling". Consider the above example expanded:

```

<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>
<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>
```

The above snippet defines an additional rule which engages when POPULATION is less than 100,000 and symbolizes the feature as a green point.

Rules support the notion of *scale dependence* which allows one to specify the scale at which a rule should engage. This allows for different portrayals of a feature based on map scale. Consider the following example:

```
<Rule>
  <MaxScaleDenominator>20000</MaxScaleDenominator>
  <PointSymbolizer>
    <Graphic>
      <Mark>
        <Fill><CssParameter name="fill">#FF0000</CssParameter>
      </Mark>
    </Graphic>
  </PointSymbolizer>
</Rule>
<Rule>
  <MinScaleDenominator>20000</MinScaleDenominator>
  <PointSymbolizer>
    <Graphic>
      <Mark>
        <Fill><CssParameter name="fill">#0000FF</CssParameter>
      </Mark>
    </Graphic>
  </PointSymbolizer>
</Rule>
```

The above rules specify that at a scale below 1:20000 features are symbolized with red points, and at a scale above 1:20000 features are symbolized with blue points.

### 8.3.8 Scale

## 8.4 SLD Extensions in GeoServer

GeoServer sports a number of vendor specific extensions to SLD 1.0. While not portable they allow to make map making more flexible and to generate better looking maps.

### 8.4.1 Geometry transformations in SLD

Each symbolizer in SLD 1.0 contains a `<Geometry>` element allowing the user to specify which geometry is to be used for rendering. In the most common case it is not specified, but it becomes useful in the case a feature has multiple geometries inside.

SLD 1.0 forces the `<Geometry>` content to be a `<ogc:PropertyName>`, GeoServer relaxes this constraint and allows a generic `sld:expression` to be used instead. Common expressions cannot manipulate geometries, but GeoServer provides a number of filter functions that can actually manipulate geometries by transforming them into something different: this is what we call *geometry transformations* in SLD.

A full list of transformations is available in the [Filter Function Reference](#).

Transformations are pretty flexible, the major limitation of them is that they happen in the geometry own reference system and unit, before any reprojection and rescaling to screen happens.

Let's look into 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:

```

1  <PointSymbolizer>
2    <Geometry>
3      <ogc:Function name="vertices">
4        <ogc:PropertyName>the_geom</ogc:PropertyName>
5      </ogc:Function>
6    </Geometry>
7    <Graphic>
8      <Mark>
9        <WellKnownName>square</WellKnownName>
10       <Fill>
11         <CssParameter name="fill">#FF0000</CssParameter>
12       </Fill>
13     </Mark>
14     <Size>6</Size>
15   </Graphic>
16 </PointSymbolizer>

```

#### View the full "Vertices" SLD

Applied to the sample `tasmania_roads` layer this will result in:

#### Start and end point

The `startPoint` and `endPoint` functions can be used to extract the start and end point of a line.

```

1  <PointSymbolizer>
2    <Geometry>
3      <ogc:Function name="startPoint">
4        <ogc:PropertyName>the_geom</ogc:PropertyName>
5      </ogc:Function>
6    </Geometry>
7    <Graphic>
8      <Mark>
9        <WellKnownName>square</WellKnownName>

```

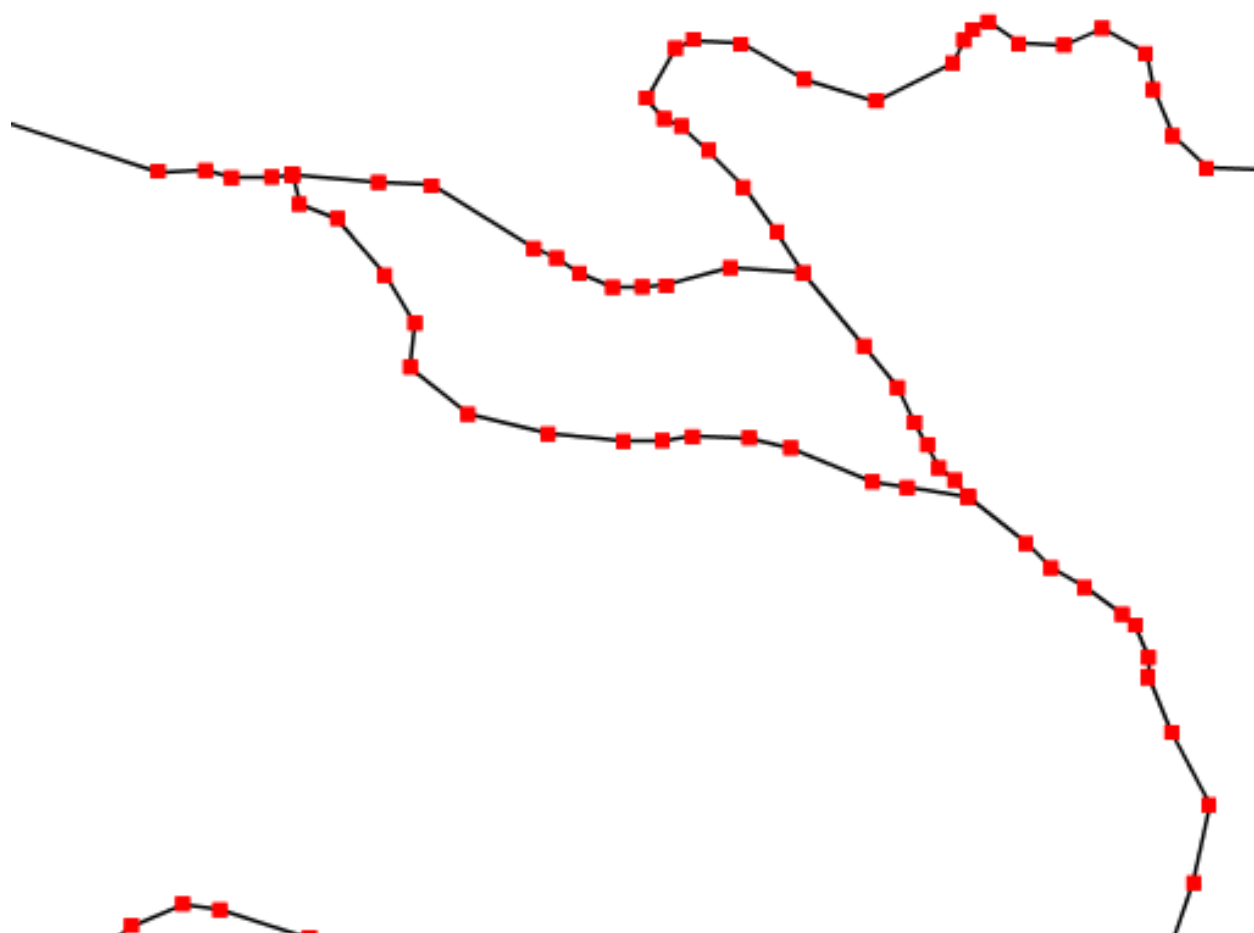


Figure 8.47: *Extracting and showing the vertices out of a geometry*

```

10     <Stroke>
11       <CssParameter name="stroke">0x00FF00</CssParameter>
12       <CssParameter name="stroke-width">1.5</CssParameter>
13     </Stroke>
14   </Mark>
15   <Size>8</Size>
16 </Graphic>
17 </PointSymbolizer>
18 <PointSymbolizer>
19   <Geometry>
20     <ogc:Function name="endPoint">
21       <ogc:PropertyName>the_geom</ogc:PropertyName>
22     </ogc:Function>
23   </Geometry>
24   <Graphic>
25     <Mark>
26       <WellKnownName>circle</WellKnownName>
27       <Fill>
28         <CssParameter name="fill">0xFF0000</CssParameter>
29       </Fill>
30     </Mark>
31     <Size>4</Size>
32   </Graphic>
33 </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 the odd offset value, set this way because the data used in the example is in geographic coordinates.

```

1 <PolygonSymbolizer>
2   <Geometry>
3     <ogc:Function name="offset">
4       <ogc:PropertyName>the_geom</ogc:PropertyName>
5       <ogc:Literal>0.00004</ogc:Literal>
6       <ogc:Literal>-0.00004</ogc:Literal>
7     </ogc:Function>
8   </Geometry>
9   <Fill>
10    <CssParameter name="fill">#555555</CssParameter>
11  </Fill>
12 </PolygonSymbolizer>

```

#### View the full “Shadow” SLD

Applied to the sample *tasmania\_roads* layer this will result in:

#### Other possibilities

GeoServer set of transformations functions also contains a number of set related or constructive transformations, such as buffer, intersection, difference and so on. However, those functions are quite heavy in

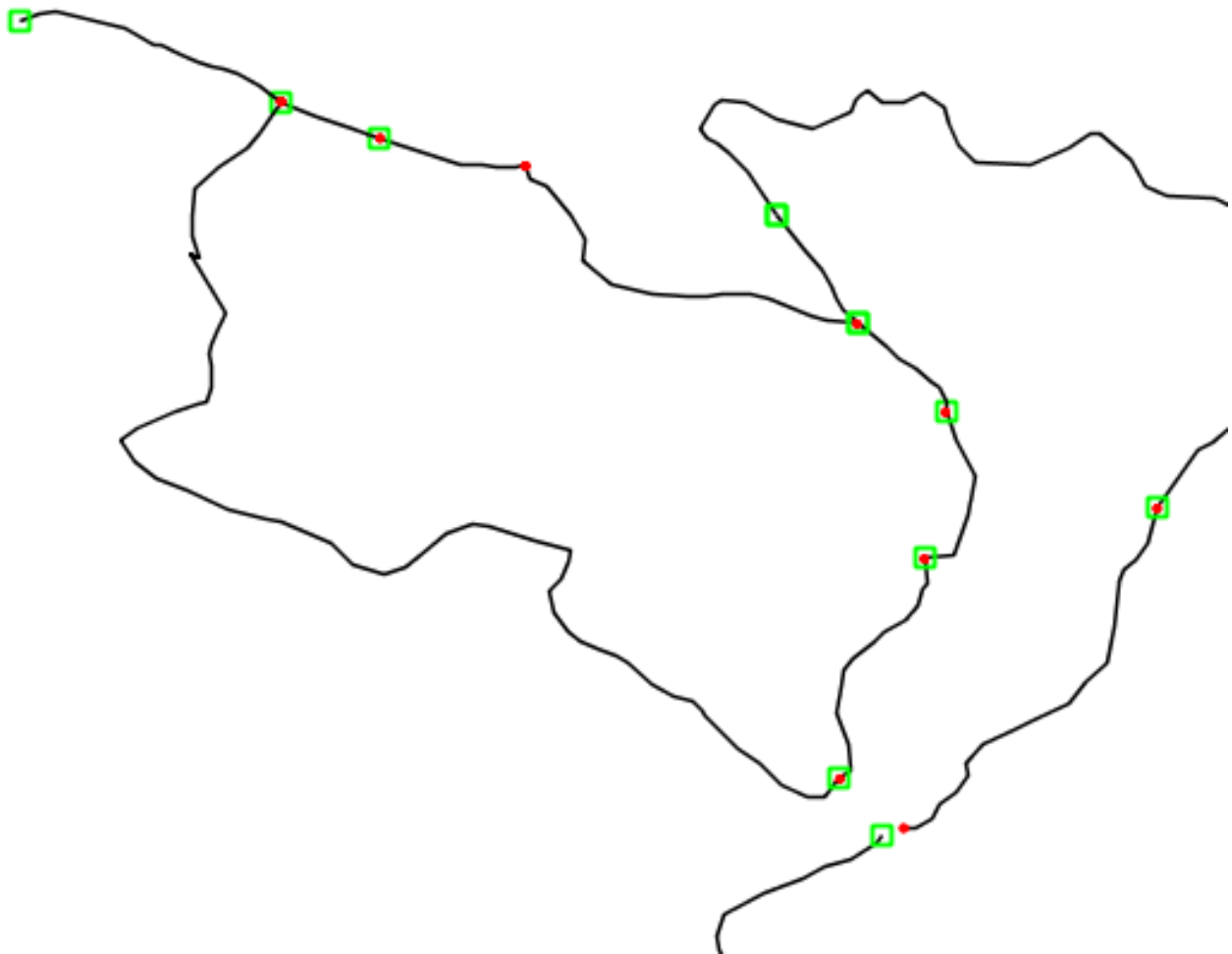


Figure 8.48: *Extracting start and end point of a line*

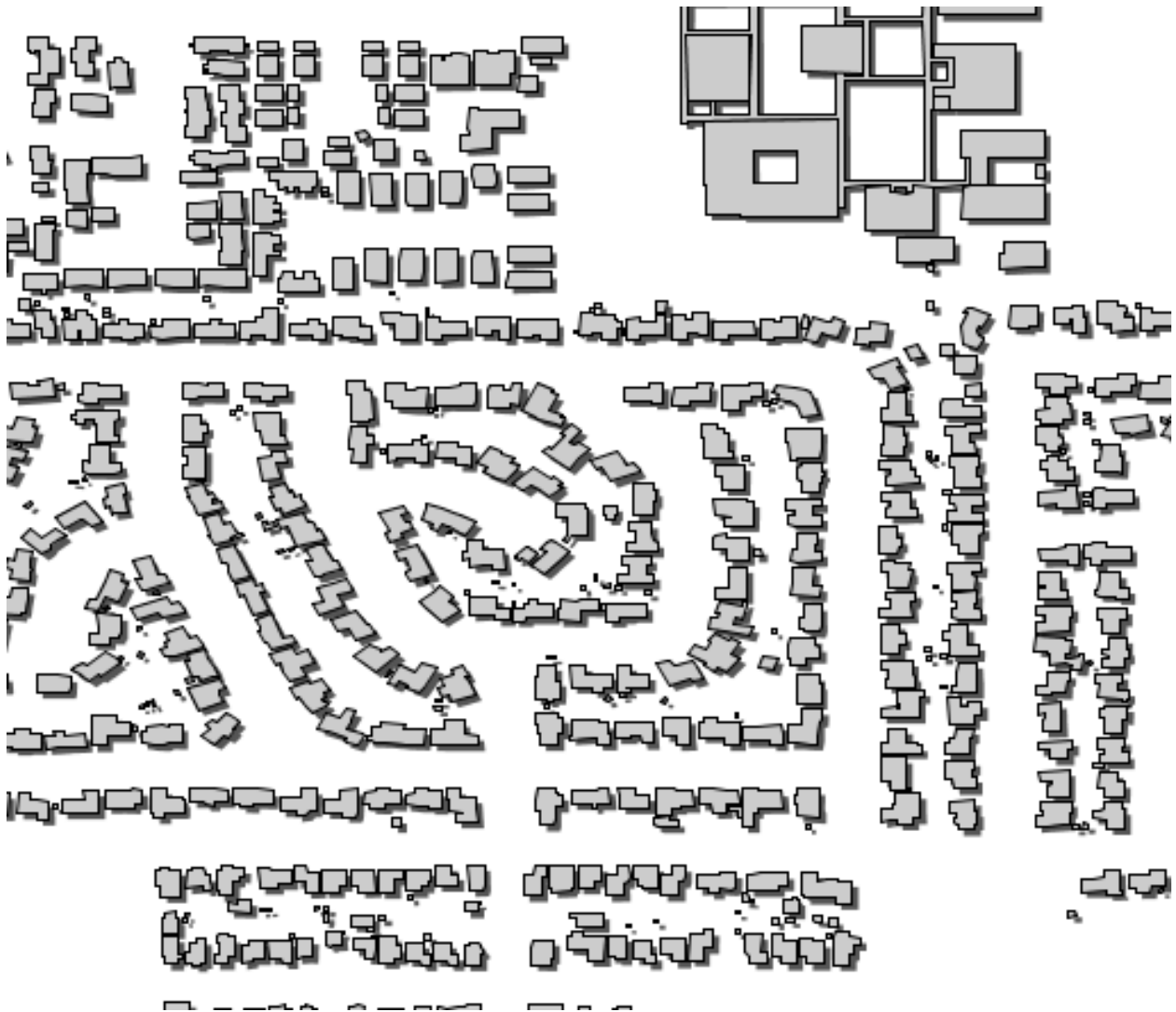


Figure 8.49: *Dropping building shadows*

terms of CPU consumption so it is advise to use them with care, activating them only at the higher zoom levels.

Buffering can often be approximated by adopting very large strokes and round line joins and line caps, without actually have to perform the geometry transformation.

### Adding new transformations

Filter functions are pluggable, meaning it's possible to build new ones in Java and then drop the resulting .jar file in GeoServer as a plugin. A guide is not available at this time, but have a look into the GeoTools main module for examples.

## 8.4.2 Parameter substitution in SLD

Parameter substitution in SLD is a GeoServer specific functionality (starting from version 2.0.2) allowing to pass down parameter values from the WMS request into an SLD style, allowing to dynamically change colors, fonts, sizes and filter thresholds dynamically.

The parameters are specified on GetMap requests using the `env` parameter:

```
...&env=paramName:value;otherParam=otherValue&...
```

and can be accessed from the SLD using the `env` function. In the simplest form the `env` function will retrieve the value of a specific parameter:

```
<ogc:Function name="env">
  <ogc:Literal>size</ogc:Literal>
</ogc:Function>
```

Alternatively a default value can be provided, to be used if the parameter value was not provided along with the GetMap request:

```
<ogc:Function name="env">
  <ogc:Literal>size</ogc:Literal>
  <ogc:Literal>6</ogc:Literal>
</ogc:Function>
```

The function can be called in any place where an OGC expression is used, so for example `CSSParameter`, sizes and offsets, and filter themselves. The SLD parser accepts it even in some places where an expression is not formally accepted, as the mark well known names.

### A working example

The following symbolizer has been parametrized in three places, every time with fall backs (full SLD is also available):

```
<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

The SLD renders the sample `sf:bugsites` as follows when no parameter is provided in the request:



Figure 8.50: *Default rendering*

If the request is changed to include the following parameter:

```
env=color:00FF00;name:triangle;size:12
```

the result will be instead:

### 8.4.3 Specifying symbolizers sizes in ground units

The SLD 1.0 specification allows the specification of sizes in just one unit: pixels.



Figure 8.51: *Rendering with parameters*

The Symbology Encoding 1.1 specification instead allows to use also meters and feet, as ground units, so that the size of the symbolizers changes on the screen as one zooms in and out.

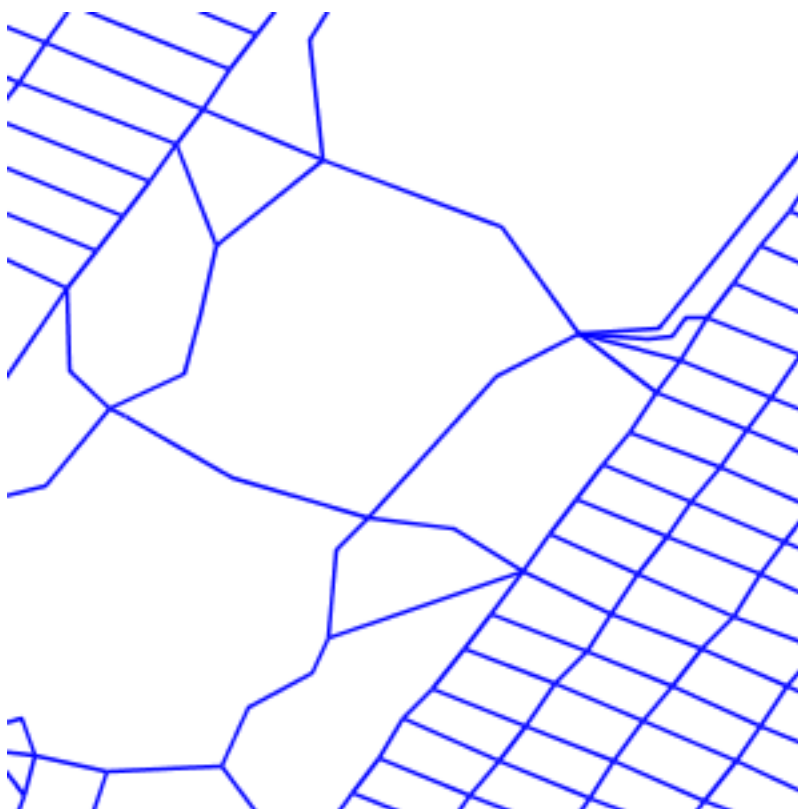
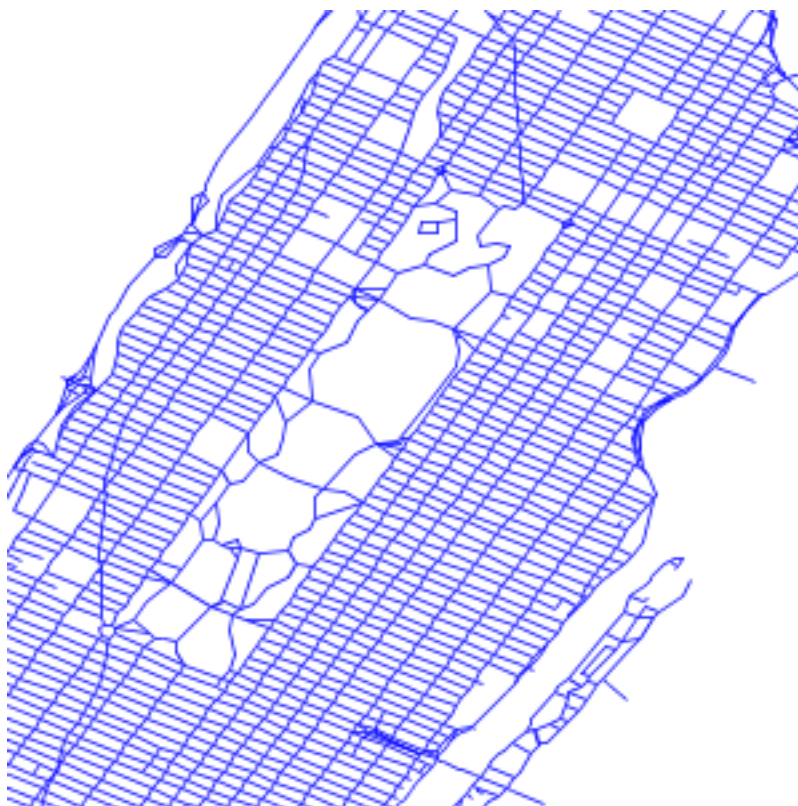
GeoServer supports the `uom` attribute just as specified in SE 1.1 in its extended SLD 1.0 support:

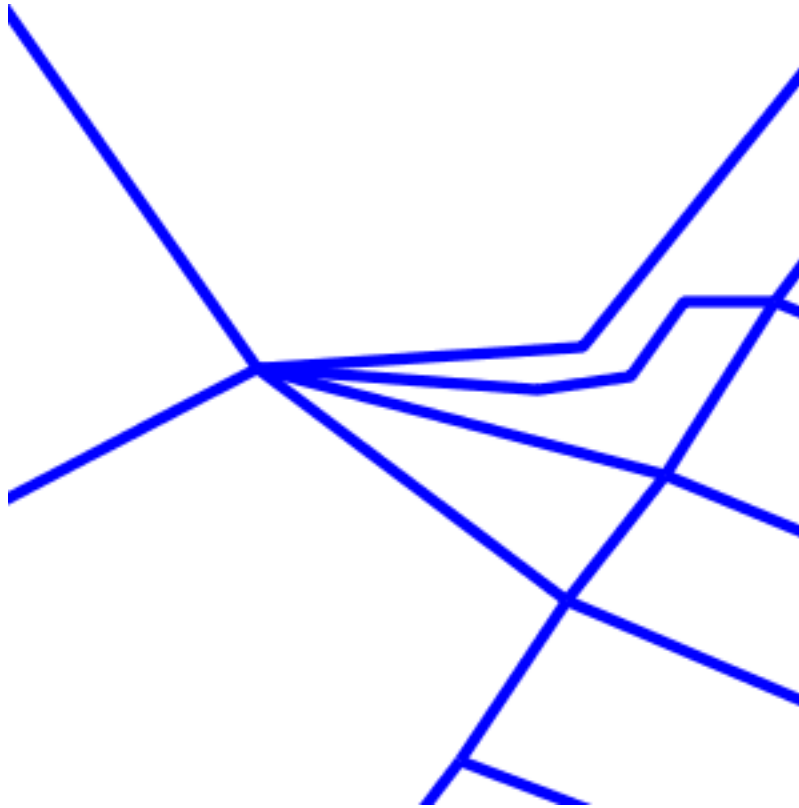
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StyledLayerDescriptor version="1.0.0" xmlns="http://www.opengis.net/sld" xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <NamedLayer>
    <Name>5m blue line</Name>
    <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 `tiger:tiger_roads` and zooming in we get:

The unit of measure supported are the same specified 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
```

This extended feature is officially supported starting with GeoServer 2.1.0, but it's already available in GeoServer 2.0.3 if the administrator starts the java virtual with the `-DenableDpiUomRescaling=true` system variable specification.

## 8.5 SLD Tips and Tricks

This section details various advanced strategies for working with SLD.

### 8.5.1 Dealing with mixed geometry types

On occasion one might have the need to render data with a single geometry column whose content type can be different for each feature (some have a polygon, some have a point, etc).

SLD 1.0 does not provide a clean solution for dealing with such a case. This is due to a mix of two issues. The first one is that point, line, and polygon symbolizers can apply to other geometry types:

- Point symbolizers can apply to any kind of geometry; if the geometry is not a point, the centroid of the feature will be used in its place.
- Line symbolizers can apply to both lines and polygons.

- Polygon symbolizers can apply to lines as well, by adding a segment connecting the last point of the line to the first.

The second issue is that there is no standard way to apply a filter identifying the type of the chosen geometry attribute.

There are a number of workarounds, either requiring 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 rendered without difficulties using only standard SLD constructs.

## Split the table

The first and obvious one is to split the table into a set of separate ones, 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 will be generated, 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:PropertyIsEqualsTo>
      <ogc:PropertyName>gtype</ogc:PropertyName>
      <ogc:Literal>Point</ogc:PropertyName>
    </ogc:PropertyIsEqualsTo>
  </ogc:Filter>
  <PointSymbolizer>
    ...
  </PointSymbolizer>
</Rule>
<Rule>
```

```
<ogc:Filter>
  <ogc:PropertyIsEqualsTo>
    <ogc:PropertyName>gtype</ogc:PropertyName>
    <ogc:Literal>Line</ogc:PropertyName>
  </ogc:PropertyIsEqualsTo>
</ogc:Filter>
<LineSymbolizer>
  ...
</LineSymbolizer>
</Rule>
<Rule>
  <ogc:Filter>
    <ogc:PropertyIsEqualsTo>
      <ogc:PropertyName>gtype</ogc:PropertyName>
      <ogc:Literal>Polygon</ogc:PropertyName>
    </ogc:PropertyIsEqualsTo>
  </ogc:Filter>
  <PolygonSymbolizer>
    ...
  </PolygonSymbolizer>
</Rule>
```

All of the above suggestions do work under the assumption that restructuring the data is technically possible, which is usually true in spatial databases that provide functions that allow to recognize 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 to be used only for rendering sake.

## Using non-standard SLD functions

SLD 1.0 uses the OGC Filter 1.0 specification for filtering out the data to be rendered by each rule. A function is a black box taking a number of parameters as inputs, and returning a result. It can implement many functionalities, such as computing a trigonometric function, formatting dates, or determining the type of a geometry.

However, none of the standards define a set of well known functions. This means that any SLD document that uses functions is valid, although it is not portable to another GIS system. If this is not a problem, filtering by geometry type can be done using the `geometryType` filter function, which takes a geometry property and returns a string, which can (currently) be one of `Point`, `LineString`, `LinearRing`, `Polygon`, `MultiPoint`, `MultiLineString`, `MultiPolygon` and `GeometryCollection`.

Using the function, a Rule matching only single points can be written as:

```
<Rule>
  <ogc:PropertyIsEqualsTo>
    <ogc:Function name="geometryType">
      <ogc:PropertyName>geom</ogc:PropertyName>
    </ogc:Function>
    <ogc:Literal>Point</ogc:Literal>
  </ogc:PropertyIsEqualsTo>
  <PointSymbolizer>
```

```

    ...
  </PointSymbolizer>
</Rule>

```

The filter becomes more complex if one has to match any kind of linear geometry. In this case, it would look like:

```

<Rule>
  <ogc:Filter>
    <ogc:PropertyIsEqualsTo>
      <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:PropertyIsEqualsTo>
  </ogc:Filter>
  <LineSymbolizer>
    ...
  </LineSymbolizer>
</Rule>

```

This filter would read like `geometryType(geom) in (LineString, LinearRing, MultiLineString)`. Filter functions in Filter 1.0 have a known number of arguments, so there are various in functions with different names, like `in2`, `in3`, ..., `in10`.





---

# Services

---

GeoServer serves data using standard protocols established by the [Open Geospatial Consortium](#). The **Web Feature Service** (WFS) allows for requests of geographical feature data (vectors). The **Web Map Service** (WMS) allows for requests of images generated from geographical data. Finally, the **Web Coverage Service** (WCS) allows for requests of coverage data (rasters). These services are the heart of GeoServer.

## 9.1 Web Feature Service

### 9.1.1 WFS basics

GeoServer provides support for Open Geospatial Consortium (OGC) Web Feature Service (WFS) versions 1.0 and 1.1. This is a standard for getting raw vector data - the 'source code' of the map - over the web. Using a compliant WFS makes it possible for clients to query the data structure and the actual data. Advanced WFS operations also enable editing and locking of the data.

GeoServer is the reference implementation of both the 1.0 and 1.1 versions of the standard, completely implementing every part of the protocol. This includes the Basic operations of `GetCapabilities`, `DescribeFeatureType` and `GetFeature`, as well as the more advanced `Transaction`, `LockFeature` and `GetGmlObject` operations. GeoServer's WFS also is integrated with GeoServer's [Security](#) system, to limit access to data and transactions. It also supports a wide variety of [WFS output formats](#), to make the raw data more widely available.

GeoServer additionally supports a special 'versioning' protocol in an extension: [WFS Versioning](#). This is not yet a part of the WFS specification, but is written to be compatible, extending it to provide a history of edits, differences between edits, and a rollback operation to take things to a previous state.

[WFS reference](#)

### Differences between WFS versions

The major differences between the WFS versions are:

- WFS 1.1.0 returns GML3 as the default GML. In WFS 1.0.0 the default was GML2. (GeoServer still supports requests in both GML3 and GML2 formats.) GML3 has slightly different ways of specifying a geometry.

- In WFS 1.1.0, the way to specify the SRS (Spatial Reference System, aka projection) is `urn:x-ogc:def:crs:EPSG:XXXX`, whereas in version 1.0.0 the specification was `http://www.opengis.net/gml/srs/epsg.xml#XXXX`. This change has implications on the axis order of the returned data.
- WFS 1.1.0 supports on-the-fly reprojection of data, which allows for data to be returned in a SRS other than the native.

### Axis ordering

WFS 1.0.0 servers return geographic coordinates in longitude/latitude (x/y) order. This is the most common way to distribute data as well (for example, most shapefiles adopt this order by default).

However, the traditional axis order for geographic and cartographic systems is latitude/longitude (y/x), the opposite and WFS 1.1.0 specification respects this. This can cause difficulties when switching between servers with different WFS versions, or when upgrading your WFS.

To sum up, the defaults are as follows:

- WFS 1.1.0 request = latitude/longitude
- WMS 1.0.0 request = longitude/latitude

GeoServer, however, in an attempt to minimize confusion and increase interoperability, has adopted the following conventions when specifying projections in the follow formats:

- `EPSG:xxxx` - longitude/latitude
- `http://www.opengis.net/gml/srs/epsg.xml#xxxx` - longitude/latitude
- `urn:x-ogc:def:crs:EPSG:xxxx` - latitude/longitude

## 9.1.2 WFS output formats

WFS returns features and feature information in a number of possible formats. This page shows a list of the output formats. In all cases the syntax for setting an output format is:

```
outputFormat=<outputformat>
```

where `<outputformat>` is any of the options below.

**Note:** Some additional output formats are available with the use of an extension, such as Excel. This list applies just to the basic GeoServer installation. The full list of output formats supported by your GeoServer instance can be found by requesting your WFS [GetCapabilities](#).

Format	Syntax	Notes
GML2	<code>outputFormat=GML2</code>	Default option using WFS 1.0.0
GML3	<code>outputFormat=GML3</code>	Default option using WFS 1.1.0
Shapefile	<code>outputFormat=shape-zip</code>	Created in a ZIP archive
JSON	<code>outputFormat=json</code>	
CSV	<code>outputFormat=csv</code>	

### Zipped shapefile customisation

Starting with GeoServer version 2.0.3 the zipped shapefile output format output can be customized by preparing a Freemarker template which will drive the file names of the zip file and the shapefiles in it. The default template looks like the following:

```
zip=${typename}
shp=${typename}${geometryType}
txt=wfsrequest
```

Structurally this is a property file, the `zip` property is the name of the zip file, the `shp` property the name of the shapefile for a given feature type and `txt` is the dump of the WFS request (the request dump is also available starting with version 2.0.3).

The properties available in the template are:

- `typename`: the feature type name (for the `zip` property it will be the first feature type in case of a request containing many)
- `geometryType`: the type of geometry contained in the shapefile (it used only if the output geometry type is generic and the variuos geometries are fanned out in one shapefile per type)
- `workspace`: the workspace of the feature type
- `timestamp`: a Date object with the request timestamp
- `iso_timestamp`: a string, the ISO timestamp of the request at GMT, in the yyyyMMdd\_HHmss format

### 9.1.3 WFS vendor parameters

WFS Vendor parameters are options that are not defined in the official WFS specification, but are allowed by it. GeoServer supports a range of custom WFS parameters.

#### CQL filters

When specifying a WFS *GetFeature* GET request, a filter can be specified in CQL (Common Query Language), as opposed to encoding the XML into the request. CQL sports a much more compact and human readable syntax compared to OGC filters. CQL isn't as flexible as OGC filters, however, and can't encode as many types of filters as the OGC specification does. In particular, filters by feature ID are not supported.

#### Example

A sample filter request using an OGC filter taken from a GET request:

```
filter=%3CFilter%20xmlns:gml=%22http://www.opengis.net/gml%22%3E%3CIntersects%3E%3CPropertyName%3Ethe
```

The same filter using CQL:

```
cql_filter=INTERSECT(the_geom,%20POINT%20(-74.817265%2040.5296504))
```

#### Reprojection

WFS 1.1 allows the ability to reproject data (to have GeoServer store the data in one projection and return GML in another).

GeoServer supports this using WFS 1.0 as well. When doing a WFS 1.0 *GetFeature* GET request you can add this parameter to specify the reprojection SRS:

```
srsName=<srsName>
```

where `<srsName>` is the code for the projection (such as EPSG:4326).

For POST requests, you can add the same code to the `Query` element.

### XML request validation

By default, GeoServer is slightly more forgiving than the WFS specification requires. To force incoming XML requests to be strictly valid, use the following parameter:

```
strict=[true|false]
```

where `false` is the default option.

### Example

Consider the following POST request:

```
<wfs:GetFeature service="WFS" version="1.0.0" xmlns:wfs="http://www.opengis.net/wfs">
  <Query typeName="topp:states"/>
</wfs:GetFeature>
```

This request will be processed successfully in GeoServer, but technically this request is invalid:

- The `Query` element should be prefixed with `wfs:`
- The namespace prefix has not been mapped to a namespace URI

Executing the above command with `strict=true` results in an error. For the request to be processed, it must be altered:

```
<wfs:GetFeature service="WFS" version="1.0.0" xmlns:wfs="http://www.opengis.net/wfs" xmlns:topp="http://topp.com" >
  <wfs:Query typeName="topp:states"/>
</wfs:GetFeature>
```

### GetCapabilities namespace filter

WFS *GetCapabilities* requests can be filtered to only return layers corresponding to a particular namespace. To do this, add the following code to your request:

```
namespace=<namespace>
```

where `<namespace>` is the namespace prefix you wish to filter on.

Using an invalid namespace prefix will not cause any errors, but the document returned will contain no information on any layers.

**Note:** This only affects the capabilities document, and not any other requests. WFS requests given to other layers, even when a different namespace is specified, will still be processed.

**Warning:** Using this parameter may cause your capabilities document to become invalid (as the WFS specification requires the document to return at least one layer).

## 9.1.4 WFS reference

### Introduction

The [Web Feature Service](#) (WFS) is a standard created by the OGC that refers to the sending and receiving of geospatial data through HTTP. WFS encodes and transfers information in Geography Markup Language, a subset of XML. The current version of WFS is 1.1.0. GeoServer supports both version 1.1.0 (the default since GeoServer 1.6.0) and version 1.0.0. There are differences between these two formats, some more subtle than others, and this will be noted where differences arise. The current version of WFS is 1.1. WFS version 1.0 is still used in places, and we will note where there are differences. However, the syntax will often remain the same.

An important distinction must be made between WFS and [Web Map Service](#), which refers to the sending and receiving of geographic information after it has been rendered as a digital image.

### Benefits of WFS

One can think of WFS as the “source code” to the maps that one would ordinarily view (via WMS). WFS leads to greater transparency and openness in mapping applications. Instead of merely being able to look at a picture of the map, as the provider wants the user to see, the power is in the hands of the user to determine how to visualize (style) the raw geographic and attribute data. The data can also be downloaded, further analyzed, and combined with other data. The transactional capabilities of WFS allow for collaborative mapping applications. In short, WFS is what enables open spatial data.

### Operations

WFS can perform the following operations:

Operation	Description
GetCapabilities	Retrieves a list of the server’s data, as well as valid WFS operations and parameters
DescribeFeatureType	Retrieves information and attributes about a particular dataset
GetFeature	Retrieves the actual data, including geometry and attribute values
LockFeature	Prevents a feature type from being edited
Transaction	Edits existing featurtypes by creating, updating, and deleting.
GetGMLObject	(Version 1.1.0 only) - Retrieves element instances by traversing XLinks that refer to their XML IDs.

A WFS server that supports **transactions** is sometimes known as a WFS-T. **GeoServer fully supports transactions.**

### GetCapabilities

The **GetCapabilities** operation is a request to a WFS 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/wfs`):

Using a GET request (standard HTTP):

```
http://www.example.com/wfs?
service=wfs&
version=1.1.0&
request=GetCapabilities
```

The equivalent 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, so we will discuss them in detail, but the POST requests are analogous. (The actual requests would be all on one line, with no line breaks, but our convention here is to break lines for clarity.) Here there are three parameters being passed to our WFS server, `service=wfs`, `version=1.1.0`, and `request=GetCapabilities`. At a bare minimum, it is required that a WFS request have these three parameters (service, version, and request). 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 WFS server that a WFS request is forthcoming. The `version` key refers to which version of WFS is being requested. Note that there are only two version numbers officially supported: “1.0.0” and “1.1.0”. Supplying a value like “1” or “1.1” will likely return an error. The `request` key is where the actual GetCapabilities operation is specified.

The Capabilities document that is returned is a long and complex chunk of XML, but very important, and so it is worth taking a closer look. (The 1.0.0 Capabilities document is very different from the 1.1.0 document discussed here, so beware.) There are five main components we will be discussing (other components are beyond the scope of this document.):

<b>ServiceIdentification</b>	This section contains basic “header” information such as the Name and ServiceType. The ServiceType mentions which version(s) of WFS are supported.
<b>Service-Provider</b>	This section provides contact information about the company behind the WFS server, including telephone, website, and email.
<b>Operations-Metadata</b>	This section describes the operations that the WFS server recognizes and the parameters for each operation. A WFS server can be set up not to respond to all aforementioned operations.
<b>Feature-TypeList</b>	This section lists the available featuretypes. They are listed in the form “namespace:featuretype”. Also, the default projection of the featuretype is listed here, along with the resultant bounding box for the data in that projection.
<b>Filter-Capabilities</b>	This section lists filters available in which to request the data. SpatialOperators (Equals, Touches), ComparisonOperators (LessThan, GreaterThan), and other functions are all listed here. These filters are not defined in the Capabilities document, but most of them (like the ones mentioned here) are self-evident.

## DescribeFeatureType

The purpose of the **DescribeFeatureType** is to request information about an individual featuretype before requesting the actual data. Specifically, **DescribeFeatureType** will request a list of features and attributes for the given featuretype, or list the featuretypes available.

Let’s say we want a list of featuretypes. The appropriate GET request would be:

```
http://www.example.com/wfs?
service=wfs&
version=1.1.0&
request=DescribeFeatureType
```

Note again the three required fields (`service`, `version`, and `request`). This will return the list of featurtypes, sorted by namespace.

If we wanted information about a specific featurtype, the GET request would be:

```
http://www.example.com/wfs?
  service=wfs&
  version=1.1.0&
  request=DescribeFeatureType&
  typeName=namespace:featurtype
```

The only difference between the two requests is the addition of `typeName=namespace:featurtype`, where `featurtype` is the name of the featurtype and `namespace` is the name of the namespace that featurtype is contained in.

## GetFeature

The **GetFeature** operation requests the actual spatial data. This is the “source code” spoken about previously. More so than the other operations, it is complex and powerful. Obviously, not all of its abilities will be discussed here.

The simplest way to run a **GetFeature** command is without any arguments.

```
http://www.example.com/wfs?
  service=wfs&
  version=1.1.0&
  request=GetFeature&
  typeName=namespace:featurtype
```

This syntax should be familiar from previous examples. The only difference is the `request=GetFeature`.

It is not recommended to run this command in a web browser, as this will return the geometries for all features in a featurtype. This can be a great deal of data. One way to limit the output is to specify a feature. In this case, the GET request would be:

```
http://www.example.com/wfs?
  service=wfs&
  version=1.1.0&
  request=GetFeature&
  typeName=namespace:featurtype&
  featureID=feature
```

Here there is the additional parameter of `featureID=feature`. Replace `feature` with the ID of the feature you wish to retrieve.

If the name of the feature is unknown, or if you wish to limit the amount of features returned, there is the `maxFeatures` parameter.

```
http://www.example.com/wfs?
  service=wfs&
  version=1.1.0&
  request=GetFeature&
  typeName=namespace:featurtype&
  maxFeatures=N
```

In the above example, N is the number of features to return.

A question that may arise at this point is how the WFS server knows which N Features to return. The bad news is that it depends on the internal structure of the data, which may not be arranged in a very helpful way. The good news is that it is possible to sort the features based on an attribute, via the following syntax. (This is new as of 1.1.0.)

```
http://www.example.com/wfs?
  service=wfs&
  version=1.1.0&
  request=GetFeature&
  typeName=namespace:featuretype&
  maxFeatures=N&
  sortBy=property
```

In the above example, `sortBy=property` determines the sort. Replace `property` with the attribute you wish to sort by. The default is to sort ascending. Some WFS servers require sort order to be specified, even if ascending. If so, append a `+A` to the request. To sort descending, add a `+D` to the request, like so:

```
http://www.example.com/wfs?
  service=wfs&
  version=1.1.0&
  request=GetFeature&
  typeName=namespace:featuretype&
  maxFeatures=N&
  sortBy=property+D
```

It is not necessary to use `sortBy` with `maxFeatures`, but they can often complement each other.

To narrow the search not by feature, but instead by an attribute, use the `propertyName` key in the form `propertyName=property`. You can specify a single property, or multiple properties separated by commas. For a single property from all features, use the following:

```
http://www.example.com/wfs?
  service=wfs&
  version=1.1.0&
  request=GetFeature&
  typeName=namespace:featuretype&
  propertyName=property
```

For a single property from just one feature:

```
http://www.example.com/wfs?
  service=wfs&
  version=1.1.0&
  request=GetFeature&
  typeName=namespace:featuretype&
  featureID=feature&
  propertyName=property
```

Or more than one property from a feature:

```
http://www.example.com/wfs?
  service=wfs&
  version=1.1.0&
  request=GetFeature&
  typeName=namespace:featuretype&
```



```
featureID=feature&
propertyName=property1,property2
```

All of these permutations so far have centered around parameters of a non-spatial nature, but it is also possible to query for features based on geometry. While there are very limited tools available in a GET request for spatial queries (much more are available in POST requests using filters) one of the most important can be used. This is known as the “bounding box” or BBOX. The BBOX allows us to ask for only such features that are contained (or partially contained) inside a box of the coordinates we specify. The form of the bbox query is `bbox=a1,b1,a2,b2` where `a`, `b`, `c`, and `d` refer to coordinates.

Notice that the syntax wasn’t `bbox=x1,y1,x2,y2` or `bbox=y1,x1,y2,x1`. The reason the coordinate-free `a,b` syntax was used above is because the order depends on the coordinate system used. To specify the coordinate system, append `srsName=CRS` to the WFS request, where `CRS` is the coordinate reference system. As for which corners of the bounding box to specify (bottom left / top right or bottom right / top left), that appears to not matter, as long as the bottom is first. So the full request for returning features based on bounding box would look like this:

```
http://www.example.com/wfs?
  service=wfs&
  version=1.1.0&
  request=GetFeature&
  typeName=namespace:featuretype&
  bbox=a1,b1,a2,b2
```

## Transaction

The **Transaction** operation performs edits of actual data that is exposed by the WFS. A transaction can add, modify and remove features. Each transaction consists of zero or more Insert, Update and Delete elements. Each element is performed in order. In GeoServer every transaction is ‘atomic’, meaning that if any of the elements fails then the data is left unchanged.

More information on the syntax of transactions can be found in the WFS specification, and in the GeoServer sample requests.

## LockFeature

The **LockFeature** operation is theoretically useful in conjunction with transactions, so users can ‘lock’ an area of the map that they are editing, to ensure that other users don’t edit it. In practice no widely used clients support the LockFeature operation.

## GetGMLObject

**GetGMLObject** is another operation that is little used in practical client applications. It only really makes sense in situations that require [Complex Features](#). It allows clients to extract just a portion of the nested properties.

## 9.1.5 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. The automatic mapping is performed with the following rules:

1. The name of the feature element maps to the name of the dataset

2. The name of the feature type maps to the name of the dataset with the string “Type” appended to it
3. The name of each attribute of the dataset maps to the name of an element particle contained in the feature type
4. The type of each attribute of the dataset maps to the appropriate xml schema type (ex: xs:int, xs:double, etc...)

As an example, consider a dataset with the following schema:

```
myDataset(intProperty:Integer, stringProperty:String, floatProperty:Float, geometry:Point)
```

The above dataset would be mapped to the following XML schema, available from 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/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:string"/>
          <xsd:element maxOccurs="1" minOccurs="0" name="floatProperty" nillable="true" type="xsd:double"/>
          <xsd:element maxOccurs="1" minOccurs="0" name="geometry" nillable="true" type="gml:PointPropertyType"/>
        </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 customization with regard to schema output. A custom schema can be used to:

- Limit the attributes which are exposed in the feature type schema
- *Changing* the types of attributes in the schema
- Change the structure of the schema, for example changing the base feature type

A mapped schema is customized by creating a file called `schema.xsd` in the appropriate feature type directory of the GeoServer data directory. As an simple example consider the use case of limiting the exposed attributes in the above dataset.

It is useful to start with the default output as a base as it is a complete schema. With the feature type schema shown above, a GetFeature request would result in features that look like the following:

```
<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>
```

Now consider the case of removing the floatProperty from attribute. To achieve this:

1. The original schema is modified and the floatProperty is removed, 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:int"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="stringProperty" nillable="true" type="xsd:string"/>
        <!-- remove the floatProperty element -->
        <xsd:element maxOccurs="1" minOccurs="0" name="floatProperty" nillable="true" type="xsd:float"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="geometry" nillable="true" type="gml:Point"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

2. The result is saved in a file named schema.xsd.
3. The schema.xsd file is copied into the feature type directory for the topp:myDataset:

```
copy schema.xsd $GEOSERVER_DATA_DIR/workspaces/<workspace>/<datastore>/myDataset/
```

Where <workspace> is the name of the workspace containing your datastore and <datastore> is the name of the data store which contains myDataset

In order for the new schema to be picked up by GeoServer the configuration must be reloaded. This can be done by logging into the admin interface and clicking the Load button the Config page. Or alternatively by restarting the entire Server.

Another DescribeFeatureType request for the topp:myDataset type now results in the floatProperty element being 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/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="topp:intProperty"/>
          <xsd:element maxOccurs="1" minOccurs="0" name="stringProperty" nillable="true" type="topp:stringProperty"/>
          <xsd:element maxOccurs="1" minOccurs="0" name="geometry" nillable="true" type="gml:GeometryProperty"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="myDataset" substitutionGroup="gml:_Feature" type="topp:myDatasetType"/>
</xsd:schema>
```

Another GetFeature request now results in features in which the floatProperty is absent:

```
<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 can be used to do a limited amount of *type changing*. Limited by the fact that a changed type must be in the same “domain” as the original type. For example integers types must be changed to integer types, temporal types to temporal types, etc...

The most common case is for geometry attributes. Often it is the case that the underlying dataset does not have the metadata necessary to report the specific type (Point, LineString, Polygon, etc...) of a geometry attribute. In these cases the automatic schema mapping would result in an element particle like the following:

```
<xsd:element maxOccurs="1" minOccurs="0" name="geometry" nillable="true" type="gml:GeometryProperty"/>
```

However it is often the case that the user knows the specific type of the geometry, for example Point. The above element could be changed to:

```
<xsd:element maxOccurs="1" minOccurs="0" name="geometry" nillable="true" type="gml:PointPropertyType"/>
```

## 9.2 Web Map Service

### 9.2.1 WMS basics

GeoServer provides support for Open Geospatial Consortium (OGC) Web Map Service (WMS) versions 1.1.1 and 1.3.0. This is a standard for generating maps on the web - it is how all the visual mapping that GeoServer does is produced. Using a compliant WMS makes it possible for clients to overlay maps from several different sources in a seamless way.

GeoServer's implementation fully supports most every part of 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.

The WMS implementation of GeoServer also supports reprojection in to any reference system in the EPSG database, and it is also possible to add additional projections if the Well Known Text is known. It also fully supports the Styled Layer Descriptor (SLD) standard, and indeed uses SLD files as its native rendering rules. For more information on how to style GeoServer data in the WMS 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 from 1.1.1 is now `crs` in 1.3.0. Although GeoServer supports both regardless of version.
- In the GetFeatureInfo operation the `x`, `y` parameters from 1.1.1 are now `i`, `j` in 1.3.0. Although GeoServer will support `x`, `y` in 1.3.0 when running on non cite compliance mode.

### Axis Ordering

The WMS 1.3 specification has mandated 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. For example, consider the WMS 1.1 request:

```
geoserver/wms?VERSION=1.1.1&REQUEST=GetMap&SRS=epsg:4326&BBOX=-180,-90,180,90&...
```

The equivalent WMS 1.3 request would be:

```
geoserver/wms?VERSION=1.1.1&REQUEST=GetMap&CRS=epsg:4326&BBOX=-90,-180,90,180&...
```

The coordinates specified by the BBOX parameter are essentially flipped.

## 9.2.2 WMS output formats

WMS returns images in a number of possible formats. This page shows a list of the output formats. In all cases 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 your 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
Geo-TIFF	format=image/geotiff	Same as TIFF, but includes extra GeoTIFF metadata
Geo-TIFF8	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	
Open-Layers	format=application/javascript	Generates an OpenLayers HTML application.

### 9.2.3 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 each option name, a description, and the minimum GeoServer version at which the option is available (old versions will just ignore it if set).

Option	Description	Version
<b>maxRequestMemory</b>	Sets the maximum amount of memory, in kilobytes, a single GetMap request is allowed to use. Each output format will make a best effort attempt to respect the maximum using the highest consuming portion of the request processing as a reference. For example, the PNG output format will take into consideration the memory used to prepare the image rendering surface in memory, usually proportional to the image size multiplied by the number of bytes per pixel	1.7.5
<b>maxRenderingTime</b>	Sets the maximum amount of time, in seconds, GeoServer will use to process the request. This time limits the “blind processing” portion of the request serving, that is, the part in which GeoServer is computing the results before writing them out to the client. The portion that is writing results back to the client is not under the control of this parameter, since this time is also controlled by how fast the network between the server and the client is. So, for example, in the case of PNG/JPEG image generation, this option will control the pure rendering time, but not the time used to write the results back.	1.7.5
<b>maxRenderingErrors</b>	Sets the maximum amount of rendering errors tolerated by a GetMap. Usually GetMap skips over faulty features, reprojection errors and the like in an attempt to serve the results anyways. This makes for a best effort rendering, but also makes it harder to spot issues, and consumes CPU cycles as each error is handled and logged	1.7.5

The default value of each limit is 0, in this case the limit won't be applied.

Once any of the set limits is exceeded, the GetMap operation will stop and a `ServiceException` will be returned to the client.

It is suggested that the administrator sets all of the above limits taking into consideration peak conditions. For example, while a GetMap request under normal circumstance may take less than a second, under high load it is acceptable for it to take longer, but usually, it's not sane that a request goes on for 30 minutes straight. The following table shows an example or reasonable values for the configuration options above:

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 if you are using GeoWebCache or TileCache. Mind the rendering process will use an extra in memory buffer for each subsequent FeatureTypeStyle in your SLD, so this will also limit the size of the image. For example, if the SLD contains two FeatureTypeStyle element in order to draw cased lines for an highway the maximum image size will be limited to 1448x1448 (the memory goes like the square of the image size, so halving the memory does not halve the image size)
maxRenderingTime	120	A request that processes for two minutes straight is probably drawing a lot of features independent of the current load. It might be the result of a client making a GetMap against a big layer using a custom style that does not have the proper scale dependencies
maxRenderingErrors	100	Encountering 100 errors is probably the result of a request that is trying to reproject a big data set into a projection that is not suited to area it covers, resulting in many reprojection failures.

## 9.2.4 WMS vendor parameters

WFS vendor parameters are options that are not defined in the official WMS specification, but are allowed by it. GeoServer supports a range of custom WMS parameters.

## angle

Starting with GeoServer 2.0.2 `angle=x` rotates the map around its center by  $x$  degrees clockwise. The rotation is supported in all raster formats, PDF and SVG based on the Batik producer (the default one).

## buffer

The `buffer` parameter specifies the number of extra pixels that should be taken into account when rendering a map (using the [GetMap](#) operation). This is important for catching features that are outside the current bounding box, but whose styling is thick enough to be visible inside the relevant area. GeoServer will try to compute this buffer automatically by parsing the SLD, but that may fail if line widths and point sizes are not literal values. When these size are linked to attributes, this parameter may be necessary.

The syntax for using a buffer is:

```
buffer=<bufferwidth>
```

where `<bufferwidth>` is the radius of the buffer in pixels.

Buffer also applies to the [GetFeatureInfo](#) operation. This creates a “search radius”, where feature info will be returned for the area around the location of the request. This is useful when working with an OpenLayers map (such as those generated by the [Layer Preview](#) page) as it relaxes the need to click precisely on a point for the appropriate feature info to be returned.

Both in the [GetMap](#) and in the [GetFeatureInfo](#) cases the default `buffer` value is computed automatically for each layer by inspecting the associated style. This happens by visiting the style, checking all active symbolizers, and returning the size of the biggest one (biggest point symbolizer, thickest line symbolizer). This automatic inspection won’t work 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 case the automatic evaluation fails, the following defaults apply:

- 0 pixels for [GetMap](#) requests
- 2 pixels for [GetFeatureInfo](#) requests

## cql\_filter

The `cql_filter` parameter is similar to the `filter` parameter, except that the filter is encoded using CQL (Common Query Language). This makes the request much more human readable. However, CQL isn’t as flexible as OGC filters, and can’t encode as many types of filters as the OGC specification does. In particular, filters by feature ID are not supported. If more than one layer is specified in the `layers` parameter, then more than one filter can be specified here, each corresponding to a layer.

An example of the same filter as above using CQL:

```
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:



```
param1:value1;param2:value2;...
```

### 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 seen in this example:

```
featureid=states.1,states.45
```

### filter

The WMS specification does not allow for much filtering of data. GeoServer's WMS filter options are expanded to match those allowed by WFS.

The `filter` parameter encodes a list of OGC filters (in XML). The list is enclosed in () parenthesis. When this parameter is used in a GET request, the brackets of XML need to be URL-encoded. If more than one layer is specified in the `layers` parameter, then more than one filter can be specified here, each corresponding to a layer.

An example of an OGC filter encoded as part of a GET request:

```
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 options in it are expressed as:

```
param1:value1;param2:value2;...
```

The currently recognized format options are:

- `antialiasing` (on, off, text): allows to control the use of antialiased rendering in raster outputs.
- `dpi`: sets the rendering dpi in raster outputs. The OGC standard dpi is 90, but if you need to perform high resolution printouts it is advised to grab a larger image and set a higher dpi. For example, to print at 300dpi a 100x100 image it is advised to ask for a 333x333 image setting the dpi value at 300. In general the image size should be increased by a factor equal to `targetDpi/90` and the target dpi set in the format options.
- `layout`: chooses a named layout for decorations, a tool for visually annotating GeoServer's WMS output. Layouts can be used to add information such as compasses and legends to the maps you retrieve from GeoServer. [WMS Decorations](#) are discussed further in the [Advanced GeoServer Configuration](#) section.

### kmattr

The `kmattr` parameter determines whether the KML returned by GeoServer should include clickable attributes or not. This parameter primarily affects Google Earth rendering. The syntax is:

```
kmattr=[true|false]
```

## kmscore

The `kmscore` parameter sets whether GeoServer should render KML data as vector or raster. This parameter primarily affects Google Earth rendering. The syntax is:

```
kmscore=<value>
```

The possible values for this parameter are between 0 (force raster output) and 100 (force vector output).

## maxFeatures and startIndex

GeoServer WMS supports the parameters `maxFeatures` and `startIndex`. Both can be used together to provide “paging” support. This 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.

Note that not every layer will support paging.

The `startIndex` parameter specifies with a positive integer the index in an ordered list of features to start rendering. For a layer to be queried this way, the underlying feature source shall support paging (such as PostGIS).

The `maxfeatures` parameter sets a limit on the amount of features rendered, using a positive integer. When used with `startIndex`, the features rendered will be the ones starting at the `startIndex` value.

## namespace

WMS [GetCapabilities](#) requests can be filtered to only return layers corresponding to a particular namespace. The syntax is:

```
namespace=<namespace>
```

where `<namespace>` is the namespace prefix.

Using an invalid namespace prefix will not cause any errors, but the document returned will not contain information on any layers, only layer groups.

**Note:** This only affects the capabilities document, and not any other requests. WMS requests given to other layers, even when a different namespace is specified, will still be processed.

## 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, you can use the syntax `palette=safe`. For more information see the tutorial on [Paletted Images](#)

## tiled

When using a tiled client such as OpenLayers, there can be issues with duplicated labels. To deal with this, GeoServer can create metatiles, that is, 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 properly, the tile size must be set to 256x256 pixels, and two extra parameters must be set.

The `tiled` parameter is of the form:

```
tiled=[yes|no]
```

For metatiling to function, this must be set to `yes`.

## tilesorigin

The `tilesorigin` parameter, also necessary for metatiling, is of the form:

```
tilesorigin=x,y
```

where `x` and `y` are the coordinates of the lower left corner (the “origin”) of the tile grid system in OpenLayers. A good way to setup the `tilesorigin` in OpenLayers is referencing the map extents directly (if the max extents are modified dynamically, also remember to update the `tilesorigin` of each meta-tiled layer accordingly):

```

1  var options = {
2      ...
3      maxExtent: new OpenLayers.Bounds(-180, -90, 180, 90),
4      ...
5  };
6  map = new OpenLayers.Map('map', options);
7
8  tiled = new OpenLayers.Layer.WMS(
9      "Layer name", "http://localhost:8080/geoserver/wms",
10     {
11         srs: 'EPSG:4326',
12         width: 391,
13         styles: '',
14         height: 550,
15         layers: 'layerName',
16         format: 'image/png',
17         tiled: true,
18         tilesorigin: [map.maxExtent.left, map.maxExtent.bottom]
19     },
20     {buffer: 0}
21 );
```

## 9.2.5 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.

### 9.2.6 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=200
```

which would produce four 20x20 icons that graphically represent the rules of the default style of the `topp:states` layer.



Figure 9.1: 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.
FEATURE-TYPE	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.
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.

## 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:

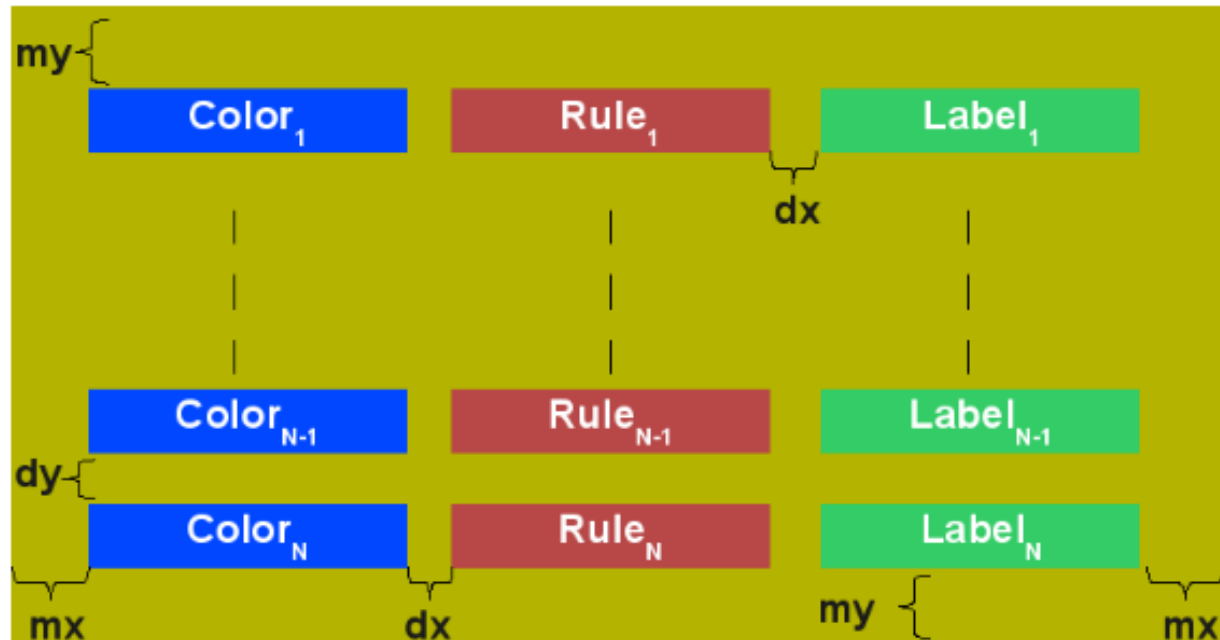


Figure 9.2: The structure of a typical legend

```
<ColorMapEntry color="#732600" quantity="9888" opacity="1.0" label="<-70 mm"/>
```

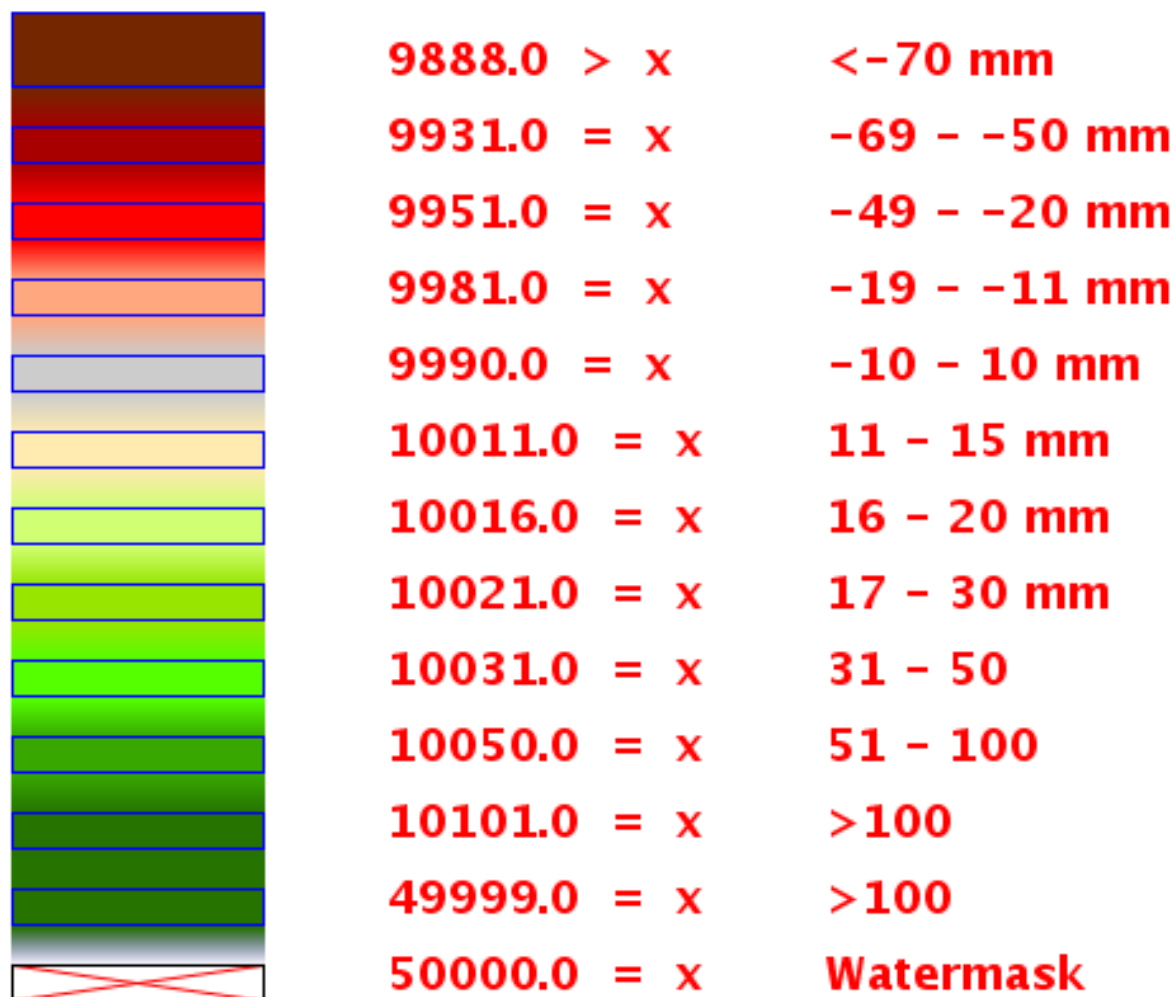
The producer for the raster legend will make use of this elements in order to build the legend, with this regards, notice that:

- 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
- 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=1000
```

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](#).



**Band selection is 1**

**ColorMap type is RAMP**

**ColorMap is not extended**

Figure 9.3: *Example of a raster legend*

## 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.

### ColorMap type is CLASSES

Refer to the SLD rainfall\_classes.sld in attachment.

### 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 border 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.

## The various control parameters and how to set them

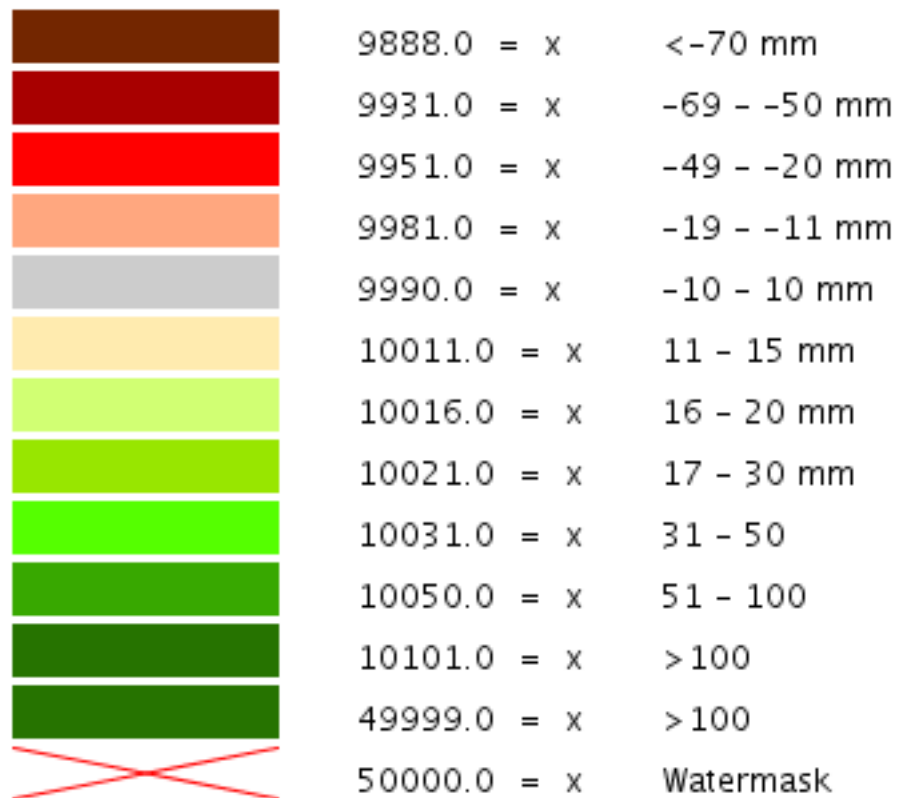
I am now going to briefly explain the various parameters tht we can use to control the layout and content of the legend (refer also to this [page](#)). Here below I have put a request that puts all the various options at tow:

```
http://localhost:8081/geoserver/wms?REQUEST=GetLegendGraphic&VERSION=1.0.0&FORMAT=image/png&WIDTH=1000
```

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 default 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



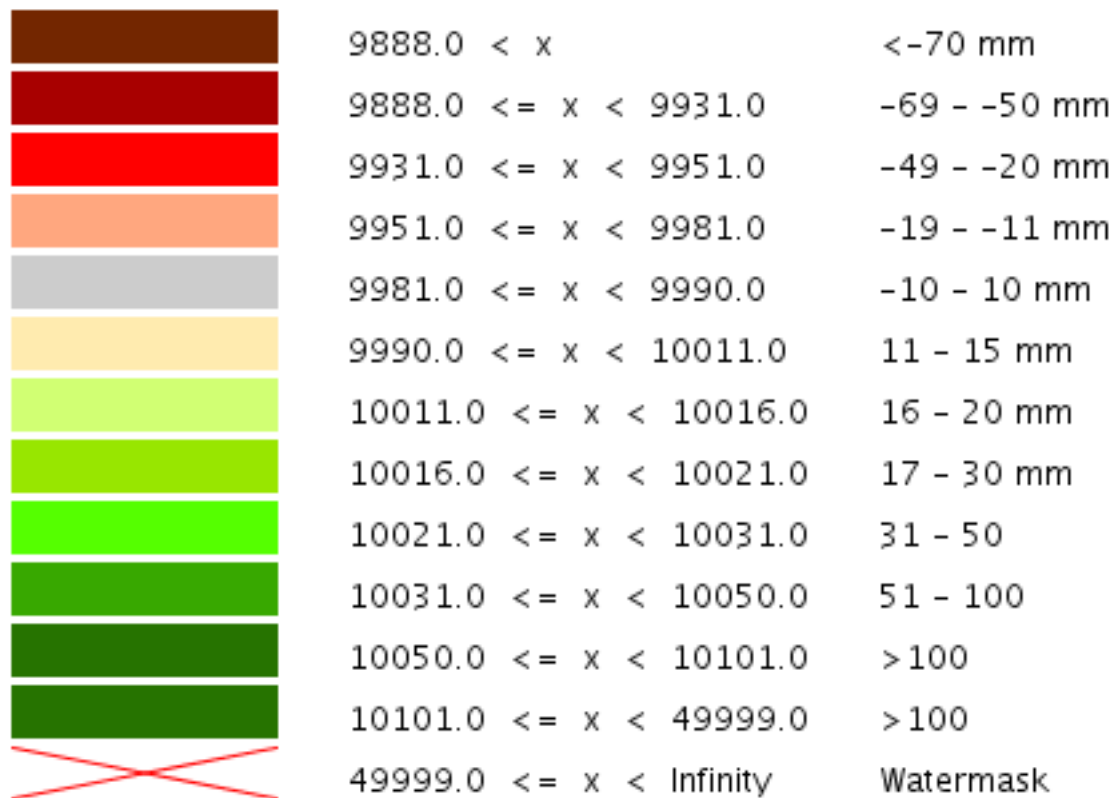


Band selection is 1

ColorMap type is UNIQUE\_VALUES

ColorMap is not extended

Figure 9.4: *Raster legend - VALUES type*



Band selection is 1

ColorMap type is CLASSES

ColorMap is not extended

Figure 9.5: *Raster legend - CLASSES type*

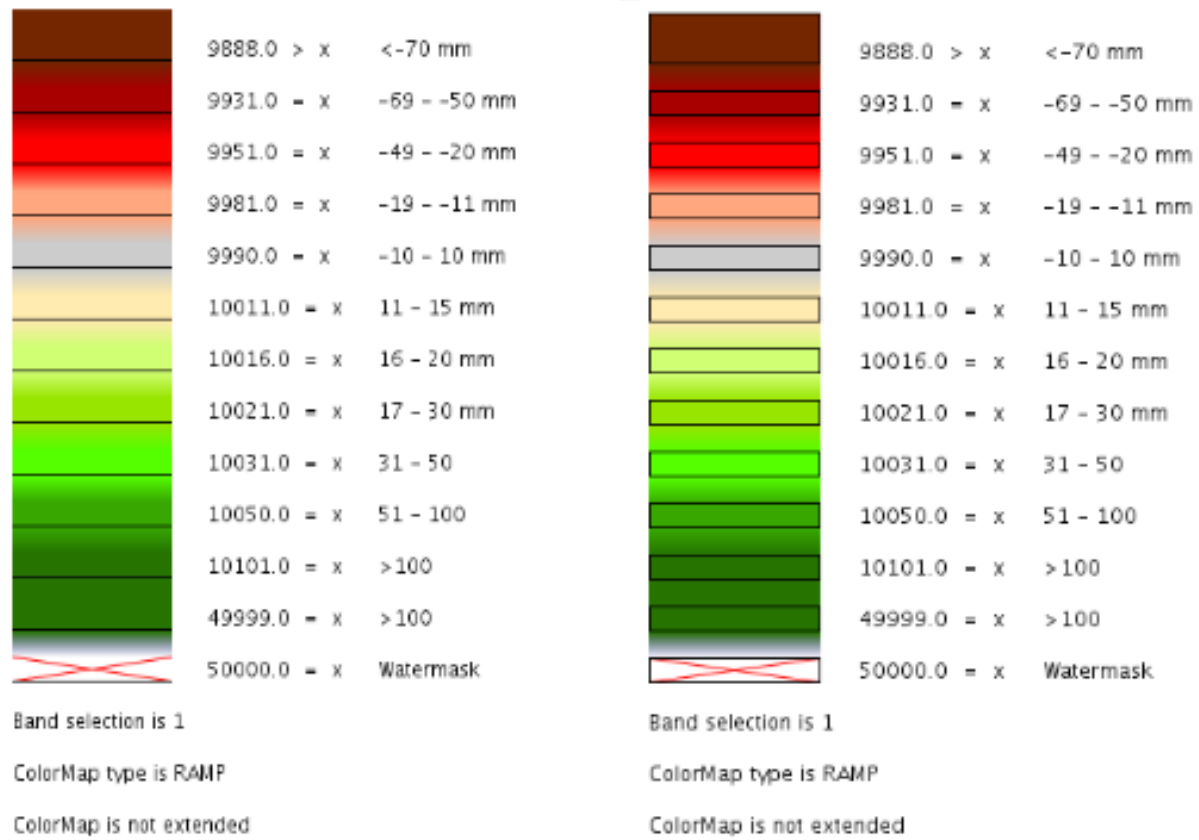


Figure 9.6: Raster legend - RAMP type

- **fontStyle (string)** can be set to italic or bold to control the text style. Other combinations 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.
- **border (boolean)** activates or deactivates the border on the color elements in order to make the separations clear. 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 as explained above. Valid values are hex values without the leading #.
- **fontColor (hex)** allows us to set the color for the text of rules and labels (see above for recommendation on how to create values).

## 9.2.7 WMS reference

### Introduction

The [Web Map Service](#) (WMS) is a standard created by the OGC that refers to the sending and receiving of georeferenced images over HTTP. These images can be produced from both vector and raster data formats. The most widely used version of WMS is 1.1.1, which GeoServer supports. The Styled Layer Descriptor (SLD) standard specifies extensions to WMS to control the styling of the WMS over the web, and GeoServer supports all these additional operations as well.

An important distinction must be made between WMS and [Web Feature Service](#), which refers to the sending and receiving of raw geographic information, before it has been rendered as a digital image.

### Benefits of WMS

WMS provides a standard interface for how to request a geospatial image. The main benefit of this is that clients can request images from multiple servers, and then combine them into one 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 can perform the following operations:

Operation	Description
GetCapabilities	Retrieves a list of the server's data, as well as valid WMS operations and parameters
GetMap	Retrieves the image requested by the client
GetFeatureInfo (optional)	Retrieves the actual data, including geometry and attribute values, for a pixel location
DescribeLayer (optional)	Indicates the WFS or WCS to retrieve additional information about the layer.
GetLegendGraphic (optional)	General mechanism for retrieving generated legend symbols

Additionally a WFS server that supports **transactions** is sometimes known as a WFS-T. **GeoServer fully supports transactions.**

## GetCapabilities

The **GetCapabilities** operation is a request to a WMS 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/wms`):

Using a GET request (standard HTTP):

```
http://www.example.com/wms?
service=wms&
version=1.1.1&
request=GetCapabilities
```

Here there are three parameters being passed to our WMS server, `service=wms`, `version=1.1.1`, and `request=GetCapabilities`. At a bare minimum, it is required that a WFS request have these three parameters (service, version, and request). 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 WMS server that a WMS request is forthcoming. The `version` key refers to which version of WMS is being requested. The `request` key is where the actual GetCapabilities operation is specified.

The Capabilities document that is returned is a long and complex chunk of XML, but very important, and so it is worth taking a closer look. There are three main components we will be discussing (other components are beyond the scope of this document.):

<b>Service</b>	This section contains basic “header” information such as the Name and basic service metadata, as well as contact information about the company behind the WMS Server.
<b>Request</b>	This section describes the operations that the WMS server recognizes and the parameters and output formats for each operation. A WMS server can be set up not to respond to all aforementioned operations.
<b>Layer</b>	This section lists the available projections and layers. In GeoServer they are listed in the form “namespace:layer”. Each layer also includes service metadata, like title, abstract and keywords.

## GetMap

The purpose of the GetMap request is to get the actual image. A client should have all the information it needs to make such a request after it understands the Capabilities document. Detailing all the potential parameters for a GetMap request is currently outside the scope of this document. But the basics are letting clients specify a bounding box, a width and a height, a spatial reference system, a format, and a style.

A great way to get to know the GetMap parameters is to experiment with the [WMS Reflector](#). The options for the **format** parameter in GeoServer can be found in the [WMS output formats](#) section. And there are a number of vendor specific parameters that GeoServer makes available, see [WMS output formats](#)

## GetFeatureInfo

The **GetFeatureInfo** operation requests the actual spatial data. It is very similar to the WFS **GetFeature** operation, and indeed since GeoServer always provides a WFS we recommend using it whenever possible. It provides more flexibility in both input and output. The one advantage that GetFeatureInfo has is that it issues its request as an x,y pixel value from a returned WMS image. So it is easier to use by a naive client that doesn’t understand all the geographic referencing needed.

The operation also allows an html output that is defined on the server side. Again, we recommend that the client use WFS GetFeature, and style the raw data in the way that it wants. But for those who for some reason need to style html on the server side, GeoServer makes this possible. See the tutorial on [GetFeatureInfo Templates](#) for information on how to template the html output.

## DescribeLayer

The **DescribeLayer** 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 good operations to do this, thankfully the **DescribeLayer** operation just routes the client to the appropriate service.

## GetLegendGraphic

**GetLegendGraphic** is an operation that provides a general mechanism for acquiring legend symbols, beyond the LegendURL reference of WMS Capabilities. It will generate a legend automatically, based on the style defined on the server, or even based on a user supplied SLD. For more information on this operation and the various options that GeoServer supports see [GetLegendGraphic](#).

## 9.3 Web Coverage Service

### 9.3.1 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.

### 9.3.2 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
<b>Lower resolution overview</b>	Looks up the two overviews with a resolution closest to the one requested and chooses the one at the lower resolution.	2.0.3
<b>Don't use overviews</b>	Overviews will be ignored, the data at its native resolution will be used instead. This is the default value.	2.0.3
<b>Higher resolution overview</b>	Looks up the two overviews with a resolution closest to the one requested and chooses the one at the higher resolution.	2.0.3
<b>Closest overview</b>	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 whether 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
<b>Maximum input memory</b>	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 of a pixel (e.g., a RGBA image will have 32bit pixels, a batimetry might have 16bit signed int ones)	2.0.3
<b>Maximum output memory</b>	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 example 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 resulting 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

### 9.3.3 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).

### 9.3.4 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.

### 9.3.5 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 `AcceptVersion` 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.

## 9.4 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 every 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.

### 9.4.1 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.

### 9.4.2 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>
```

### 9.4.3 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.



**GeoServer**

**Server**

- Server Status
- Contact Information
- Global Settings**
- JAI Settings
- About GeoServer

**Services**

- GWC
- WFS
- WMS

**Data**

- Workspaces
- Stores
- Layers
- Layer Groups

## Global Settings

Settings that apply to the entire server.

- ☐ Verbose Messages
- ☐ Verbose Exception Reporting
- ☒ **Enable Global Services**

**Number of Decimals**

8

**Character Set**

UTF-8

**Proxy Base URL**



---

# RESTful Configuration

---

GeoServer provides a RESTful interface through which clients can configure an instance through simple HTTP calls. With it clients can programatically configure the data served by GeoServer.

To learn more proceed to the [Overview of REST](#) section. For details about the API and some hands on examples proceed to the [REST Configuration API Reference](#) or [REST Configuration Examples](#) sections.

## 10.1 Overview of REST

REST is an acronym for “REpresentational State Transfer”. The basic idea of REST is to rely on 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, if you retrieve data in an XML format, you can send data back to the server in the same XML format to set it.

Operations on resources are implemented with the standard primitives of HTTP: GET, DELETE, PUT, POST, HEAD, etc. Each resource is represented as a standard URI.

For more information about REST visit the [wikipedia page](#).

## 10.2 REST Configuration API Reference

### 10.2.1 Formats and representations

A `format` specifies how a 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 a number of ways. The first is with the `Accepts` header. For instance setting the header to “text/xml” would specify the desire to have the resource returned as XML. The second method of specifying the format is via file extension. For example consider the resource “foo”. To request a representation of foo as XML the request uri would end with “foo.xml”. To request 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.

In a POST or PUT operation the format specifies 1) the representation of the content being sent to the server, and 2) the representation of the response to be sent back. The former is specified with the `Content-type` header. To send a representation in XML, the content type “text/xml” or “application/xml” would be used. The latter is specified with the `Accepts` header as specified in the above paragraph describing a GET operation.

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

### 10.2.2 Authentication

POST, PUT, and DELETE requests (requests that modify resources) require the client to be authenticated. Currently the only supported method of authentication is Basic authentication. See the [examples](#) section for examples of how to perform authentication with various clients and environments.

### 10.2.3 Status codes

A 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](#).

### 10.2.4 Workspaces

A workspace is a grouping of data stores. More commonly known as a namespace, it is commonly used to group data that is related in some way.

**Note:** For GeoServer 1.x a workspace can be considered the equivalent of a namespace, and the two are kept in sync. For example, the namespace “topp, <http://openplans.org/topp>” corresponds to the workspace “topp”.

### Operations

`/workspaces[.<format>]`

Method	Action	Return Code	Formats	Default Format
GET	List all workspaces	200	HTML, XML, JSON	HTML
POST	Create a new workspace	201 with <code>Location</code> header	XML, JSON	
PUT		405		
DELETE		405		

*Representations:*

- HTML
- XML
- JSON



/workspaces/<ws>[.<format>]

Method	Action	Return Code	Formats	Default Format	Parameters
GET	Returns workspace <i>ws</i>	200	HTML, XML, JSON	HTML	
POST		405			
PUT	200	Modify workspace <i>ws</i>	XML, JSON		
DELETE	200	Delete workspace <i>ws</i>	XML, JSON		<i>recurse</i>

Representations:

- HTML
- XML
- JSON

Exceptions:

- 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

The *recurse* parameter is used to recursively delete all resources contained by the specified workspace. This includes data stores, coverage stores, feature types, etc... Allowable values for this parameter are “true” or “false”. The default value is “false”.

/workspaces/default[.<format>]

Method	Action	Return Code	Formats	Default Format
GET	Returns default workspace	200	HTML, XML, JSON	HTML
POST		405		
PUT	200	Set default workspace	XML, JSON	
DELETE		405		

## 10.2.5 Namespaces

A namespace is a uniquely identifiable grouping of feature types. A namespace is identified by a prefix and a uri.

**Note:** In GeoServer 1.7.x a namespace is used to group data stores, serving the same purpose as a workspace. In 1.7.x the two are kept in sync. Therefore when adding a new namespace a workspace whose name matches the prefix of the namespace is implicitly created.

### Operations

/namespaces[.<format>]

Method	Action	Return 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		

Representations:

- HTML
- XML

- **JSON**

/namespaces/<ns>[.<format>]

Method	Action	Return Code	Formats	Default Format
GET	Returns namespace <i>ns</i>	200	HTML, XML, JSON	HTML
POST		405		
PUT	200	Modify namespace <i>ns</i>	XML, JSON	
DELETE	200	Delete namespace <i>ns</i>	XML, JSON	

*Representations:*

- **HTML**
- **XML**
- **JSON**

*Exceptions:*

- 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>]

Method	Action	Return Code	Formats	Default Format
GET	Returns default namespace	200	HTML, XML, JSON	HTML
POST		405		
PUT	200	Set default namespace	XML, JSON	
DELETE		405		

## 10.2.6 Data stores

A `data store` is a source of spatial data that is vector based. It can be a file in the case of a Shapefile, a database in the case of PostGIS, or a server in the case of a remote Web Feature Service.

### Operations

/workspaces/<ws>/datastores[.<format>]

Method	Action	Return Code	Formats	Default Format
GET	List all data stores in workspace <i>ws</i>	200	HTML, XML, JSON	HTML
POST	Create a new data store	201 with <code>Location</code> header	XML, JSON	
PUT		405		
DELETE		405		

*Representations:*

- **HTML**
- **XML**
- **JSON**

/workspaces/<ws>/datastores/<ds>[.<format>]

Method	Action	Return Code	Formats	Default Format	Parameters
GET	Return data store ds	200	HTML, XML, JSON	HTML	<i>recurse</i>
POST		405			
PUT	Modify data store ds				
DELETE	Delete data store ds				

*Representations:*

- **HTML**
- **XML**
- **JSON**

*Exceptions:*

- 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

The *recurse* parameter is used to recursively delete all feature types contained by the specified data store. Allowable values for this parameter are “true” or “false”. The default value is “false”.

/workspaces/<ws>/datastores/<ds>/file[.<extension>] /workspaces/<ws>/datastores/<ds>/url[.<extension>]  
/workspaces/<ws>/datastores/<ds>/external[.<extension>]

This operation uploads a file containing spatial data into an existing datastore, or creates a new datastore.

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

The *file*, *url*, and *external* endpoints are used to specify the method that is used to upload the file.

The *file* method is used to directly upload a file from a local source. The body of the request is the file itself.

The *url* method is used to indirectly upload 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.

The *external* method is used to forgo upload and use an existing file on the server. The body of the request is the absolute path to the existing file.

Method	Action	Re- turn Code	For- mats	Default Format	Parame- ters
GET	Get the underlying files for the data store as a zip file with mime type <code>application/zip</code> . <i>Deprecated.</i>	200			
POST		405			
PUT	Uploads files to the data store <code>ds</code> , creating it if necessary.	200	See <a href="#">notes</a> below.		<a href="#">configure</a> , <a href="#">target</a> , <a href="#">update</a>
DELETE		405			

Exceptions:

- GET for a data store that does not exist -> 404
- GET for a data store that is not file based -> 404

When the file for a datastore are PUT, it can be as a standalone file, or as a zipped archive. The standalone file method is only applicable to data stores that work from a single file, GML for example. Data stores like Shapefile must be sent as a zip archive.

When uploading a zip archive the `Content-type` should be set to `application/zip`. When uploading a standalone file the content type should be appropriately set based on the file type. The `configure` parameter is used to control how the data store is configured upon file upload. It can take one of the three values “first”, “none”, or “all”.

- `first` - Only setup the first feature type available in the data store. This is the default.
- `none` - Do not configure any feature types.
- `all` - Configure all feature types.

The `target` parameter is used to control the type of datastore that is created on the server when the data-store being PUT to does not exist. The allowable values for this parameter are the same as for the [extension parameter](#). The `update` parameter is used to control how existing data is handled when the file is PUT into a datastore that (a) already exists and (b) already contains a schema that matches the content of the file. It can take one of the two values “append”, or “overwrite”.

- `append` - Data being uploaded is appended to the existing data. This is the default.
- `overwrite` - Data being uploaded replaces any existing data.

### 10.2.7 Feature types

A `feature type` is a vector based spatial resource or data set that originates from a data store. In some cases, like Shapefile, a feature type has a one-to-one relationship with its data store. In other cases, like PostGIS, the relationship of feature type to data store is many-to-one, with each feature type corresponding to a table in the database.

#### Operations

```
/workspaces/<ws>/datastores/<ds>/featuretypes[.<format>]
```

Method	Action	Return Code	Formats	Default Format	Parameters
GET	List all feature types in datastore <code>ds</code>	200	HTML, XML, JSON	HTML	<a href="#">list</a>
POST	Create a new feature type	201 with <code>Location</code> header	XML, JSON		
PUT		405			
DELETE		405			

Representations:

- HTML
- XML
- JSON

Exceptions:

- 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

The `list` parameter is used to control the category of feature types that are returned. It can take one of the three values “configured”, “available”, or “all”.

- `configured` - Only setup or configured feature types are returned. This is the default value.
- `available` - Only unconfigured feature types (not yet setup) but are available from the specified datastore will be returned.
- `all` - The union of configured and available.

/workspaces/<ws>/datastores/<ds>/featuretypes/<ft>[.<format>]

Method	Action	Return Code	Formats	Default Format	Parameters
GET	Return feature type <code>ft</code>	200	HTML, XML, JSON	HTML	
POST		405			
PUT	Modify feature type <code>ft</code>	200	XML,JSON		
DELETE	Delete feature type <code>ft</code>	200			<a href="#">recurse</a>

Representations:

- HTML
- XML
- JSON

Exceptions:

- 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

The `recurse` parameter is used to recursively delete all layers that reference by the specified feature type. Allowable values for this parameter are “true” or “false”. The default value is “false”.

## 10.2.8 Coverage stores

A `coverage store` is a source of spatial data that is raster based.

## Operations

/workspaces/<ws>/coveragestores[.<format>]

Method	Action	Return Code	Formats	Default Format
GET	List all coverage stores in workspace <i>ws</i>	200	HTML, XML, JSON	HTML
POST	Create a new coverage store	201 with <i>Location</i> header	XML, JSON	
PUT		405		
DELETE		405		

Representations:

- **HTML**
- **XML**
- **JSON**

/workspaces/<ws>/coveragestores/<cs>[.<format>]

Method	Action	Return Code	Formats	Default Format	Parameters
GET	Return coverage store <i>cs</i>	200	HTML, XML, JSON	HTML	
POST		405			
PUT	Modify coverage store <i>cs</i>				
DELETE	Delete coverage store <i>ds</i>				<i>recurse</i>

Representations:

- **HTML**
- **XML**
- **JSON**

Exceptions:

- 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

The *recurse* parameter is used to recursively delete all coverages contained by the specified coverage store. Allowable values for this parameter are “true” or “false”. The default value is “false”.

/workspaces/<ws>/coveragestores/<cs>/file[.<extension>]

The *extension* parameter specifies the type of coverage store. The following extensions are supported:

Extension	Coveragestore
geotiff	GeoTIFF
worldimage	Geo referenced image (JPEG,PNG,TIF)
imagemosaic	Image mosaic

Method	Action	Return Code	Formats	Default Format	Parameters
GET	Get the underlying files for the coverage store as a zip file with mime type <code>application/zip</code> .	200			
POST		405			
PUT	Creates or overwrites the files for coverage store <code>cs</code> .	200	See <a href="#">notes</a> below.		<a href="#">configure</a> , <a href="#">coverage-Name</a>
DELETE		405			

Exceptions:

- GET for a data store that does not exist -> 404
- GET for a data store that is not file based -> 404

When the file for a coverage store is PUT, it can be as a standalone file, or as a zipped archive. The standalone file method is only applicable to coverage stores that work from a single file, GeoTIFF for example. Coverage stores like Image mosaic must be sent as a zip archive.

When uploading a zip archive the `Content-type` should be set to `application/zip`. When uploading a standalone file the content type should be appropriately set based on the file type. The `coverageName` parameter is used to specify 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:** Currently the relationship between a coverage store and a coverage is one to one. However there is currently work underway to support multi-dimensional coverages, so in the future this parameter is likely to change.

## 10.2.9 Coverages

A coverage is a raster based data set which originates from a coverage store.

### Operations

`/workspaces/<ws>/coveragestores/<cs>/coverages[.<format>]`

Method	Action	Return Code	Formats	Default Format
GET	List all coverages in coverage store <code>cs</code>	200	HTML, XML, JSON	HTML
POST	Create a new coverage	201 with <code>Location</code> header	XML, JSON	
PUT		405		
DELETE		405		

Representations:

- HTML
- XML
- JSON

`/workspaces/<ws>/coveragestores/<cs>/coverages/<c>[.<format>]`

Method	Action	Return Code	Formats	Default Format	Parameters
GET	Return coverage <i>c</i>	200	HTML, XML, JSON	HTML	<i>recurse</i>
POST		405			
PUT	Modify coverage <i>c</i>	200	XML,JSON		
DELETE	Delete coverage <i>c</i>	200			

Representations:

- **HTML**
- **XML**
- **JSON**

Exceptions:

- GET for a coverage that does not exist -> 404
- PUT that changes name of coverage -> 403
- PUT that changes coverage store of coverage -> 403

The `recurse` parameter is used to recursively delete all layers that reference by the specified coverage. Allowable values for this parameter are “true” or “false”. The default value is “false”.

### 10.2.10 Styles

A *style* describes how a resource (feature type or coverage) should be symbolized or rendered by a Web Map Service. In GeoServer styles are specified with [SLD](#).

#### Operations

/styles[.<format>]

Method	Action	Return Code	Formats	Default Format	Parameters
GET	Return all styles	200	HTML, XML, JSON	HTML	<i>name</i>  <i>purge</i>
POST	Create a new style	201 with Location header	SLD, XML, JSON See <i>notes</i> below		
PUT		405			
DELETE		405			

Representations:

- **HTML**
- **XML**
- **JSON**

When POSTing or PUTing a style as SLD, the `Content-type` header should be set to `application/vnd.ogc.sld+xml`. The `name` parameter specifies the name to be given to the style. This option is most useful when POSTing a style in SLD format, and an appropriate name can be not be inferred from the SLD itself.

/styles/<s>[.<format>]



Method	Action	Return Code	Formats	Default Format
GET	Return style <i>s</i>	200	SLD, HTML, XML, JSON	HTML
POST		405		
PUT	Modify style <i>s</i>	200	SLD, XML, JSON See <a href="#">notes</a> above	
DELETE	Delete style <i>s</i>	200		

The `purge` parameter specifies whether the underlying SLD file for the style should be deleted on disk. It is specified as a boolean value (`true|false`). When set to `true` the underlying file will be deleted.

*Representations:*

- SLD
- HTML
- XML
- JSON

*Exceptions:*

- 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

## 10.2.11 Layers

A `layer` is a *published* resource (feature type or coverage).

**Note:** In GeoServer 1.x a layer can be considered the equivalent of a feature type or a coverage. In GeoServer 2.x, the two will be separate entities, with the relationship from a feature type to a layer being one-to-many.

### Operations

`/layers[.<format>]`

Method	Action	Return Code	Formats	Default Format
GET	Return all layers	200	HTML, XML, JSON	HTML
POST		405		
PUT		405		
DELETE		405		

*Representations:*

- HTML
- XML
- JSON

`/layers/<l>[.<format>]`

Method	Action	Return Code	Formats	Default Format	Parameters
GET	Return layer <i>l</i>	200	HTML, XML, JSON	HTML	
POST		405			
PUT	Modify layer <i>l</i>	200	XML,JSON		
DELETE	Delete layer <i>l</i>	200			

*Representations:*

- HTML
- XML
- JSON

*Exceptions:*

- GET for a layer that does not exist -> 404
- PUT that changes name of layer -> 403
- PUT that changes resource of layer -> 403

The `recurse` parameter is used to recursively delete all resources referenced by the specified layer. Allowable values for this parameter are “true” or “false”. The default value is “false”.

`/layers/<l>/styles[.<format>]`

Method	Action	Return Code	Formats	Default Format
GET	Return all styles for layer <code>l</code>	200	SLD, HTML, XML, JSON	HTML
POST	Add a new style to layer <code>l</code>	201, with <code>Location</code> header	XML, JSON	
PUT		405		
DELETE		405		

### 10.2.12 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 often referred to as a “base map”.

#### Operations

`/layergroups[.<format>]`

Method	Action	Return Code	Formats	Default Format
GET	Return all layer groups	200	HTML, XML, JSON	HTML
POST	Add a new layer group	201, with <code>Location</code> header	XML,JSON	
PUT		405		
DELETE		405		

*Representations:*

- HTML
- XML
- JSON

`/layergroups/<lg>[.<format>]`

Method	Action	Return Code	Formats	Default Format
GET	Return layer group <code>lg</code>	200	HTML, XML, JSON	HTML
POST		405		
PUT	Modify layer group <code>lg</code>	200	XML,JSON	
DELETE	Delete layer group <code>lg</code>	200		

*Representations:*

- HTML
- XML

- **JSON**

*Exceptions:*

- 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

### 10.2.13 Configuration reloading

Reloads the catalog and configuration from disk. This operation is used to reload GeoServer 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	Return Code	Formats	Default Format
GET		405		
POST	Reloads the configuration from disk	200		
PUT	Reloads the configuration from disk	200		
DELETE		405		

## 10.3 REST Configuration Examples

This section contains a number of examples which illustrate various uses of the REST data configuration api. The examples are grouped by environment/language.

### 10.3.1 cURL

The examples in this section use the [cURL](#) utility, which is a handy command line tool for executing HTTP requests and transferring files. Though cURL is used the examples apply to any HTTP-capable tool or library.

#### Adding a new workspace

The following creates a new workspace named “acme” with a POST request:

```
curl -u admin:geoserver -v -XPOST -H 'Content-type: text/xml' \
  -d '<workspace><name>acme</name></workspace>' \
  http://localhost:8080/geoserver/rest/workspaces
```

The response should contain the following:

```
< HTTP/1.1 201 Created
< Date: Fri, 20 Feb 2009 01:56:28 GMT
< Location: http://localhost:8080/geoserver/rest/workspaces/acme
< Server: Noelios-Restlet-Engine/1.0.5
< Transfer-Encoding: chunked
```

Note the `Location` response header which specifies the location of the newly created workspace. The following retrieves the new workspace as XML with a GET request:

```
curl -XGET -H 'Accept: text/xml' http://localhost:8080/geoserver/rest/workspaces/acme
```

The response should look like:

```
<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" />
  </dataStores>
  <coverageStores>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" href="http://localhost:8080/geoserver/rest/workspaces/acme/coverages" />
  </coverageStores>
</workspace>
```

Specifying the `Accept` header to relay the desired representation of the workspace can be tedious. The following is an equivalent (yet less RESTful) request:

```
curl -XGET http://localhost:8080/geoserver/rest/workspaces/acme.xml
```

## Uploading a Shapefile

In this example a new datastore will be created by uploading a Shapefile. The following uploads the zipped shapefile `roads.zip` and creates a new datastore named `roads`:

```
curl -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 following retrieves the created data store as XML:

```
curl -XGET http://localhost:8080/geoserver/rest/workspaces/acme/datastores/roads.xml
```

```
<dataStore>
  <name>roads</name>
  <workspace>
    <name>acme</name>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" href="http://localhost:8080/geoserver/rest/workspaces/acme/datastores" />
  </workspace>
  <connectionParameters>
    <namespace>http://acme</namespace>
    <url>file:/Users/jdeolive/devel/geoserver/1.7.x/data/minimal/data/roads/roads.shp</url>
  </connectionParameters>
  <featureTypes>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" href="http://localhost:8080/geoserver/rest/workspaces/acme/datastores/roads/featuretypes" />
  </featureTypes>
</dataStore>
```

By default when a shapefile is uploaded a feature type is automatically created. See [Layers](#) page for details on how to control this behaviour. The following retrieves the created feature type as XML:

```
curl -XGET
  http://localhost:8080/geoserver/rest/workspaces/acme/datastores/roads/featuretypes/roads.xml
```

```
<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/...>
  </namespace>
  ...
</featureType>
```

### Adding an existing Shapefile

In the previous example a Shapefile was uploaded directly by sending a zip file in the body of a request. This example shows how to add a Shapefile that already exists on the server.

Consider a directory on the server `/data/shapefiles/roads` that contains the Shapefile `roads.shp`. The following adds a new datastore for the Shapefile:

```
curl -u admin:geoserver -XPUT -H 'Content-type: text/plain' \
  -d 'file:///data/shapefiles/roads/roads.shp' \
  http://localhost:8080/geoserver/rest/workspaces/acme/datastores/roads/external.shp
```

Note the `external.shp` part of the request uri.

### Adding a directory of existing Shapefiles

In the previous example a datastore was created for a single Shapefile that already existed on the server. This example shows how to add a directory of Shapefiles.

Consider a directory on the server `/data/shapefiles` that contains a number of different Shapefiles. The following adds a new datastore for all the Shapefiles in the directory:

```
curl -u admin:geoserver -XPUT -H 'Content-type: text/plain' \
  -d 'file:///data/shapefiles/roads' \
  "http://localhost:8080/geoserver/rest/workspaces/acme/datastores/roads/external.shp?configure=all"
```

Note the `configure=all` query string parameter.

### Changing a feature type style

In the previous example a Shapefile was uploaded, and in the process a feature type was created. Whenever a feature type is created an layer is implicitly created for it. The following retrieves the layer as XML:

```
curl -XGET http://localhost:8080/geoserver/rest/layers/acme:roads.xml
```

```
<layer>
  <name>roads</name>
  <path>/</path>
  <type>VECTOR</type>
  <defaultStyle>
    <name>roads_style</name>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" href="http://localhost:8080/...>
```

```
</defaultStyle>
<styles>
  <style>
    <name>line</name>
    <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" href="http://localhost:8080/...>
  </style>
</styles>
<resource class="featureType">
  <name>roads</name>
  <atom:link xmlns:atom="http://www.w3.org/2005/Atom" rel="alternate" href="http://localhost:8080/...>
</resource>
<enabled>>false</enabled>
</layer>
```

When the layer is created a default style named `polygon` is assigned to it. This style can be viewed with a WMS GetMap request (<http://localhost:8080/geoserver/wms/reflect?layers=acme:roads>)

In this example a new style will be created and assigned to the layer created in the previous example. The following creates a new style named `roads_style`:

```
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/styles
```

Uploading the file `roads.sld`:

```
curl -u admin:geoserver -XPUT -H 'Content-type: application/vnd.ogc.sld+xml' \
  -d @roads.sld http://localhost:8080/geoserver/rest/styles/roads_style
```

The following applies the newly created style to the layer created in the previous example:

```
curl -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
```

The new style can be viewed with the same GetMap request (<http://localhost:8080/geoserver/wms/reflect?layers=acme:roads>) as above.

## Adding a PostGIS database

**Note:** This section assumes that a PostGIS database named `nyc` is present on the local system and is accessible by the user `bob`.

In this example a PostGIS database named `nyc` will be added as a new data store. In preparation create the database and import the `nyc.sql` file:

```
psql nyc < nyc.sql
```

The following represents the new data store:

```
<dataStore>
  <name>nyc</name>
  <connectionParameters>
    <host>localhost</host>
    <port>5432</port>
    <database>nyc</database>
```

```
<user>bob</user>
<dbtype>postgis</dbtype>
</connectionParameters>
</dataStore>
```

Save the above xml into a file named `nycDataStore.xml`. The following adds the new datastore:

```
curl -u admin:geoserver -XPOST -T nycDataStore.xml -H 'Content-type: text/xml' \
  http://localhost:8080/geoserver/rest/workspaces/acme/datastores
```

### Adding a PostGIS table

In this example two tables from the PostGIS database created in the previous example will be added as feature types. The following adds the table `buildings` as a new feature type:

```
curl -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 following retrieves the created feature type:

```
curl -XGET http://localhost:8080/geoserver/rest/workspaces/acme/datastores/nyc/featuretypes/buildings
```

This GetMap request (<http://localhost:8080/geoserver/wms/reflect?layers=acme:buildings>) shows the rendered buildings layer.

The following adds the table `parks` as a new feature type:

```
curl -u admin:geoserver -XPOST -H 'Content-type: text/xml' \
  -d '<featureType><name>parks</name></featureType>' \
  http://localhost:8080/geoserver/rest/workspaces/acme/datastores/nyc/featuretypes
```

This GetMap request (<http://localhost:8080/geoserver/wms/reflect?layers=acme:parks>) shows the rendered parks layer.

### Creating a layer group

In this example the layers added in previous examples will be used to create a layer group. First a few styles need to be added. The following adds a style for the buildings layer:

```
curl -u admin:geoserver -XPUT -H 'Content-type: application/vnd.ogc.sld+xml' -d @buildings.sld \
  http://localhost:8080/geoserver/rest/styles/buildings_style
```

The following adds a style for the parks layer:

```
curl -u admin:geoserver -XPUT -H 'Content-type: application/vnd.ogc.sld+xml' -d @parks.sld \
  http://localhost:8080/geoserver/rest/styles/parks_style
```

The following represents the new layer group:

```
<layerGroup>
  <name>nyc</name>
  <layers>
    <layer>roads</layer>
    <layer>parks</layer>
    <layer>buildings</layer>
  </layers>
  <styles>
    <style>roads_style</style>
    <style>parks</style>
    <style>buildings_style</style>
  </styles>
</layerGroup>
```

Save the following in a file named `nycLayerGroup.xml`. The following creates the new layer group:

```
curl -u admin:geoserver -XPOST -d @nycLayerGroup.xml -H 'Content-type: text/xml' \
  http://localhost:8080/geoserver/rest/layergroups
```

This GetMap request (<http://localhost:8080/geoserver/wms/reflect?layers=nyc>) shows the rendered layer group.

### 10.3.2 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
```

as well as some xml looking something like the example [here](#).

## 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
```

### 10.3.3 Python

We are looking for volunteers to flesh out this section with examples. But anyone looking to do python scripting of the GeoServer REST config API should use [gsconfig.py](#). It is quite capable, and used in production as part of [GeoNode](#), so examples can be found in that codebase. It just currently lacks documentation and examples.



---

# 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.

## 11.1 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.

### 11.1.1 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.

## 11.1.2 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, 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, 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.properties` 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.

### 11.1.3 Manually editing the CRS 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 forecast-
ing.',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](#).

## 11.2 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 altogether.

### 11.2.1 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

### 11.2.2 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 together 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.

### 11.2.3 Forcing GeoServer to relinquish Log4J control

GeoServer internally overrides the Log4J configuration by using the current logging configuration as a template and applying 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, environment variables, or servlet context parameters:

```
RELINQUISH_LOG4J_CONTROL=true
```

### 11.2.4 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, environment 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

## 11.3 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.

### 11.3.1 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 <a href="#">Decoration Types</a> )
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.

### 11.3.2 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

The **legend** decoration (`type="legend"`) overlays a graphic containing legends for the layers in the map.

### 11.3.3 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.

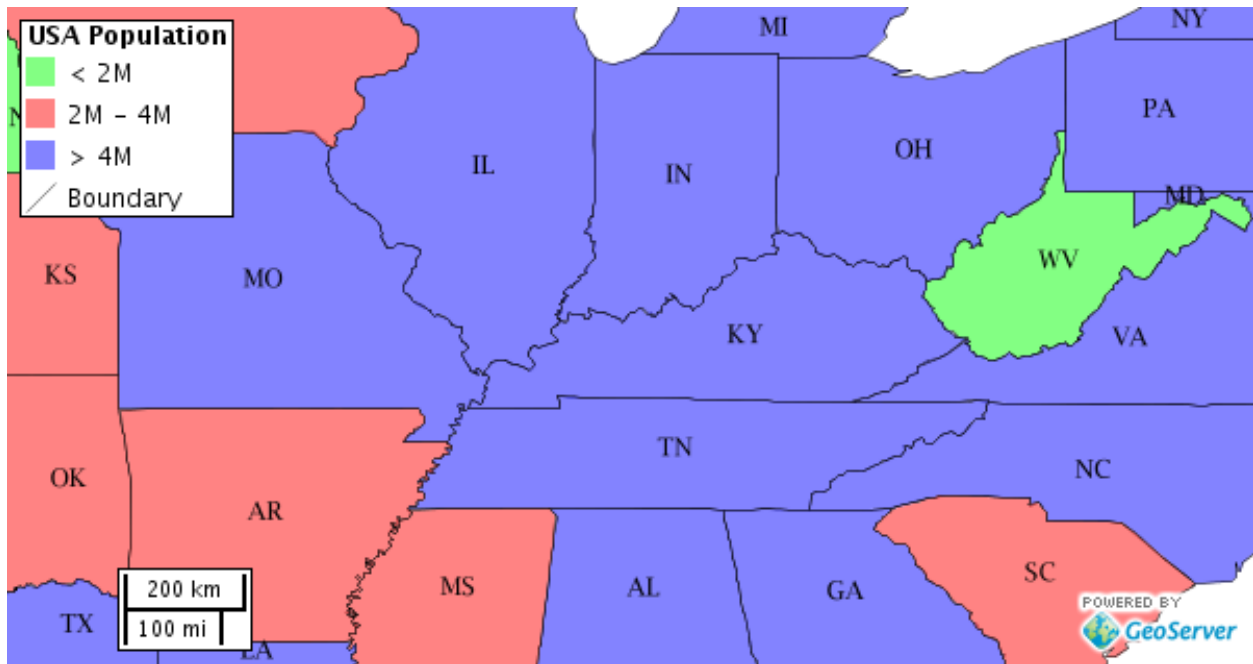


Figure 11.1: The default states layer, drawn with the decoration layout above.



---

# Security

---

This section details the security subsystem in GeoServer. The system is based on [Spring Security](#).

## 12.1 Accessing secured resources

The [Web Administration Interface](#) is secured by form-based authentication, with an optional “remember-me” cookie setting. The OGC services are secured using HTTP BASIC authentication, provided on each call.

The form-based authentication is based on browser session, so if the same browser is used to access services as well, the authentication will be remembered.

Here is the process for accessing a secured resource:

1. If no authentication is provided, anonymous login will be assumed.
2. If any authentication information is included, it will be used.
  - In the case of form-based information, a session will be created to store it.
  - In the case of HTTP BASIC authentication, session integration will be performed only if a session is already available (to avoid overhead).
3. If the resource being accessed is secured and the current user is anonymous, authentication will be requested either using HTTP BASIC authentication (for services) or by using form based login (for the web administration interface).
4. If the resource accessed is secured and the currently authenticated user lacks sufficient access rights, an HTTP 404 error will be returned.

## 12.2 Users and roles

Security in GeoServer is a **role-based system**. Roles are created to serve particular functions (Examples: access WFS, administer UI, read certain layers), and users are linked to those roles.

### 12.2.1 Setting users and roles

Linking users and roles is done via the file `users.properties`. This file is in the GeoServer data directory in the `security` directory. By default, this file contains one line:

```
admin=geoserver,ROLE_ADMINISTRATOR
```

There is only one predefined role: `ROLE_ADMINISTRATOR`. This role provides full access to all systems inside GeoServer. This file links the user **admin** (with password **geoserver**) to this role.

**Note:** It should go without saying that if you are using GeoServer in a production environment, this default behavior should be immediately changed.

Other users and roles can be created by adding to the `users.properties` file. The syntax is:

```
user=password,role[,role2,...]
```

where:

- **user** is the user name
- **password** is the password associated with that user
- **role[,role2,...]** is the name of the role(s) associated with this user

Although the default administrator role is `ROLE_ADMINISTRATOR`, the naming convention is not mandatory. Multiple users can be linked with the same role. Users and passwords are case-sensitive.

## 12.3 Service-level security

**Note:** Service-level security and [Layer-level security](#) cannot be combined. For example, it is not possible to specify access to a specific OGC service on one specific layer.

GeoServer allows access to be determined on a service level (WFS, WMS).

Access to services is linked to roles. (See also [Users and roles](#).) Services and roles are linked in a file called `services.properties`, which is located in the `security` directory in your GeoServer data directory.

### 12.3.1 Syntax

The syntax for setting security is as follows. (Parameters in brackets are optional.):

```
service[.method]=role[,role2,...]
```

where:

- **service** can be `wfs`, `wms`, or `wcs`
- **method** can be any method supported by the service. (Ex: `GetFeature` for WFS, `GetMap` for WMS)
- **role[,role2,...]** is the name(s) of predefined roles.

**Note:** Make sure that your role is linked to a user, unless you want to deny access to everyone. Set this in the `users.properties` file.



### 12.3.2 Examples

By default, no service-level security is set. Two examples are given in the `service.properties` file by default, commented out:

```
wfs.GetFeature=ROLE_WFS_READ
wfs.Transaction=ROLE_WFS_WRITE
```

The first line will link access to the WFS GetFeature method to the role `ROLE_WFS_READ`. The second line will link access to the WFS Transactions to the role `ROLE_WFS_WRITE`.

## 12.4 Layer-level security

**Note:** Layer-level security and [Service-level security](#) cannot be combined. For example, it is not possible to specify access to a specific OGC service on one specific layer.

GeoServer allows access to be determined on a per-layer basis.

Access to layers are linked to roles. (See also [Users and roles](#).) Layers and roles are linked in a file called `layers.properties`, which is located in the `security` directory in your GeoServer data directory.

### 12.4.1 Syntax

The syntax for setting security is as follows. (Parameters in brackets [] are optional):

```
namespace.layer.permission=role[,role2,...]
```

where:

- **namespace** is the name of the namespace. The wildcard `*` is used to indicate all namespaces.
- **layer** is the name of a featuretype or coverage. The wildcard `*` is used to indicate all layers.
- **permission** is the type of access permission (**r** for read access, **w** for write access).
- **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.

Starting with GeoServer 1.7.7, if a namespace or layer name is supposed to contain dots they can be escaped using `\\`. For example, if a rule must refer to `layer.with.dots` the following syntax can be used:

```
topp.layer\\.with\\.dots.r=ROLE1,ROLE2,...
```

Each entry must have a unique combination of namespace, 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 namespace is not specified, it inherits permissions from the global specification. If a permission for a layer is not specified, it inherits permissions from its namespace specification. If a user belongs to multiple roles, the **least restrictive** permission they inherit will apply.

The `layers.properties` file may contain a further directive that specifies the way in which GeoServer will advertise secured layers and behave when a secured layer is accessed without the necessary privileges. The line is:

```
mode=option
```

where **option** can 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. Because of this, it can sometimes 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 work fine. 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 whom you give direct data access links to get the data after authentication.

## 12.4.2 Examples

### Protecting a single namespace and a single layer

The following entries demonstrate configuring GeoServer so that it is primarily a read-only server:

```

*. *.r=*
*. *.w=NO_ONE
private.*.r=TRUSTED_ROLE
private.*.w=TRUSTED_ROLE
topp.congress_district.w=STATE_LEGISLATORS

```

In this example, here is the map of roles to permissions:

Role	private.*	topp.*	topp.congress_district	(all other namespaces)
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 entries demonstrate configuring GeoServer so that it is locked down:

```

*. *.r=TRUSTED_ROLE
*. *.w=TRUSTED_ROLE
topp.*.r=*
army.*.r=MILITARY_ROLE, TRUSTED_ROLE
army.*.w=MILITARY_ROLE, TRUSTED_ROLE

```

In this example, here is the map of roles to permissions:

Role	topp.*	army.*	(All other namespaces)
TRUSTED_ROLE	r/w	r/w	r/w
MILITARY_ROLE	r	r/w	(none)
(All other users)	r	(none)	(none)

## A more complex situation

The following entries demonstrate configuring GeoServer with global-, namespace-, 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
```

In this example, here is the map of roles to permissions:

Role	topp.state	topp.poly_landmarks	topp.military_bases	topp.(all other layers)	(All other namespaces)
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/w	r/w	(none)	r	(none)
(All other users)	(none)	r	(none)	r	(none)

**Note:** The entry `topp.states.w=NO_ONE` is not needed, because this permission would be inherited from the global level, i.e. the line `*.*.w=NO_ONE`.

## Invalid configuration file

The following set of entries would not be valid because the namespace, layer, and permission combinations of the entries are not unique:

```
topp.state.rw=ROLE1
topp.state.rw=ROLE2, ROLE3
```

## 12.5 REST Security

**Note:** RESTful security configuration is available in GeoServer versions greater than 2.0.1.

In addition to providing the ability to secure OWS style services GeoServer also allows for the securing of RESTful services.

As with layer and service security, RESTful security configuration is based on [Users and roles](#). Mappings from request uri to role are defined in a file named `rest.properties` located in the `security` directory of the GeoServer data directory.

### 12.5.1 Syntax

The following is the syntax for defining access control rules for RESTful services (parameters in brackets [] are optional):

```
uriPattern;method[,method,...]=role[,role,...]
```

where:

- **uriPattern** is the [ant pattern](#) that matches a set of request uri's..
- **method** is an HTTP request method, one of GET, POST, PUT, POST, DELETE, or HEAD
- **role** is the name of a predefined role. The wildcard `*` is used to indicate the permission is applied to all users, including anonymous users.

A few things to 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 a commonly used syntax for pattern matching directory and file paths. The [examples](#) section contains some basic examples. The apache ant [user manual](#) contains more sophisticated cases.

## 12.5.2 Examples

Most of the examples in this section are specific to the [rest configuration extension](#) but any RESTful GeoServer service can be configured in the same manner.

### Allowing only autenticated access to services

The most secure of configurations is one that forces any request to be authenticated. The following will lock down access to all requests:

```
/**;GET,POST,PUT,DELETE=ROLE_ADMINISTRATOR
```

A slightly 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 like the previous except it grants access to a specific role rather than the administrator:

```
/**;GET,POST,PUT,DELETE=ROLE_TRUSTED
```

Where `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
```

## 12.6 Disabling security

If you are using an external security subsystem, you may want to disable the built-in security to prevent conflicts.

**Warning:** Beware! If security is disabled, you'll have to make sure the external one locks down the administration interface, otherwise it will be completely unlocked!

To disable GeoServer security, first shut down GeoServer, open the `web.xml` file (located inside the `WEB-INF` directory) and comment out the "Spring Security Filter Chain Proxy" filter definition parameters. These two pieces of code should look something like this:

```
<filter>
  <filter-name>Spring Security Filter Chain Proxy</filter-name>
  <filter-class>org.springframework.security.util.FilterToBeanProxy</filter-class>
  <init-param>
    <param-name>targetClass</param-name>
    <param-value>org.springframework.security.util.FilterChainProxy</param-value>
  </init-param>
</filter>

<filter-mapping>
  <filter-name>Spring Security Chain Proxy</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

Comment these sections out. When GeoServer is restarted, the internal security subsystem will be completely disabled.



---

# 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.

## 13.1 Java Considerations

### 13.1.1 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). If this is not possible, use Oracle JRE 5 (also known as JRE 1.5). 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.

### 13.1.2 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.

## 13.2 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.

### 13.2.1 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
<code>-server</code>	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.
<code>-Xmx256M -Xms48m</code>	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. <code>-Xmx256M</code> 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 <a href="#">Web Administration Interface</a> section) so that it uses 75% of the heap (0.75). <code>-Xmx48m</code> will tell the virtual machine to grab a 48MB heap on startup, which will make heap management more stable during heavy load serving.
<code>-XX:SoftRefLRUPolicyThreshold=3</code>	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.
<code>-XX:MaxPermSize=128M</code>	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.
<code>-XX:XX:+UseParallelGC</code>	Enables the throughput garbage collector.

For more information about JVM configuration, see the article [Performance tuning garbage collection in Java](#).

## 13.3 Configuration Considerations

### 13.3.1 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.

### 13.3.2 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.

### 13.3.3 Personalize your server

This 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`)

### 13.3.4 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 featurtype 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

### 13.3.5 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).

### 13.3.6 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).

## 13.4 Data Considerations

### 13.4.1 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.

### 13.4.2 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 Data](#)), PostGIS is recommended. Oracle, DB2, and ArcSDE are also supported.

### 13.4.3 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
+-- ...
+-- mosaic_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).

## 13.5 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"
```

### 13.5.1 Debian/Ubuntu

Download the init script

### 13.5.2 Suse

Download the init script

## 13.6 Other Considerations

### 13.6.1 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.

### 13.6.2 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`.

### 13.6.3 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.

### 13.6.4 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.

## 13.7 Troubleshooting

### 13.7.1 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).

### 13.7.2 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.

### 13.7.3 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
```

### 13.7.4 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 order 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 processes running. It can be used to retrieve 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.ClassLoader
11521 -XX:MinHeapFreeRatio=10 -XX:MaxHeapFreeRatio=20 -Djava.library.path=/usr/lib/jni -Dosgi.require
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 JVMs 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 frame)
- 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 (Interpreted frame)
- org.apache.jk.common.ChannelSocket.acceptConnections() @bci=72, line=660 (Interpreted frame)
- org.apache.jk.common.ChannelSocket$SocketAcceptor.runIt(java.lang.Object[]) @bci=4, line=870 (Interpreted frame)
- org.apache.tomcat.util.threads.ThreadPool$ControlRunnable.run() @bci=167, line=690 (Interpreted frame)
- 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 frame)
- 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/libclic_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.hsquidb.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.hsquidb.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.

---

# 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 has been integrated into GeoServer, although it is also available as a [standalone product](#) for use with other map servers.

This section will discuss the version of GeoWebCache embedded in GeoServer. For information about the standalone product, please see the [GeoWebCache homepage](#).

## 14.1 Using GeoWebCache

**Note:** For an more in-depth discussion of using GeoWebCache, please see the [GeoWebCache documentation](#).

### 14.1.1 GeoWebCache integration with GeoServer WMS

GeoWebCache (as of GeoServer 2.1.0) is transparently integrated with the GeoServer WMS, and so requires no special endpoint or custom URL in order to be used. In this way one can have the simplicity of a standard WMS endpoint with the performance of a tiled client.

This direct integration is turned off by default. It can be enabled by going to the [GeoWebCache Settings](#) 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 the following conditions apply**:

1. `TILED=true` is included in the request.

2. All other request parameters (tile height and width) match up with a tile in the layer's gridset.
3. There are no vendor-specific parameters (such as `cql_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, however, very possible to create an OpenLayers application that caches tiles; just make sure that the tileorigin aligns with the gridset.

### 14.1.2 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.

### 14.1.3 Disk quota

GeoWebCache has a built-in disk quota feature to prevent disk space from growing unbounded. Disk quotas are turned off by default, but can be configured on the [GeoWebCache Settings](#) page in the [Web Administration Interface](#). 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).

### 14.1.4 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.

### 14.1.5 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)

If you need another projection, you can create a custom configuration file, `geowebcache.xml`, in the same directory that contains the cache (see the [GeoWebCache Configuration](#) page for information on how to set this). This configuration file is the same as used by the standalone version of GeoWebCache (see that documentation for more details). The configuration syntax directly supports the most common WMS parameters such as style, palette, and background color. To prevent conflicts, the layers in this file should be named differently from the ones that are loaded from GeoServer.

## 14.2 GeoWebCache Configuration

GeoWebCache is automatically configured to be used with GeoServer with 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 [GeoWebCache Demo page](#) to test the configuration.

**Note:** The `GEOSERVER_WMS_URL` parameter in `web.xml`, used in earlier versions of GeoServer, is deprecated and should not be used.

### 14.2.1 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.

### 14.2.2 Custom configuration

If you need to access more features than the automatic configuration offers, you can create a custom configuration file. Inside the GeoWebCache cache directory (see above), create a file named `geowebcache.xml`. Please refer to the [GeoWebCache documentation](#) for how to customize this file. Restart GeoServer for the changes to take effect. You may also wish to check the logfiles after starting GeoServer to verify that this file has been successfully read.

### 14.2.3 GeoWebCache with multiple GeoServer instances

For stability reasons, it is not recommended to use the embedded GeoWebCache with multiple GeoServer instances. If you want configure GeoWebCache as a front-end for multiple instances of GeoServer, we recommend using the [standalone GeoWebCache](#).

## 14.3 GeoWebCache Demo page

GeoWebCache comes with a demo page where you can view configured layers, reload the configuration (when changing settings or adding new layers), and seed/refresh the existing cache on a per-layer basis.



Layer name:	Grids Sets:	
nurc:Arc_Sample <a href="#">Seed this layer</a>	EPSG:900913 EPSG:4326	OpenLayers: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> ] OpenLayers: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> ] KML: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> , <a href="#">kml</a> ]
nurc:Img_Sample <a href="#">Seed this layer</a>	EPSG:900913 EPSG:4326	OpenLayers: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> ] OpenLayers: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> ] KML: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> , <a href="#">kml</a> ]
nurc:Pk50095 <a href="#">Seed this layer</a>	EPSG:900913 EPSG:4326	OpenLayers: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> ] OpenLayers: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> ] KML: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> , <a href="#">kml</a> ]
nurc:mosaic <a href="#">Seed this layer</a>	EPSG:900913 EPSG:4326	OpenLayers: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> ] OpenLayers: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> ] KML: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> , <a href="#">kml</a> ]
sf:archsites <a href="#">Seed this layer</a>	EPSG:900913 EPSG:4326	OpenLayers: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> ] OpenLayers: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> ] KML: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> , <a href="#">kml</a> ]
sf:bugsites <a href="#">Seed this layer</a>	EPSG:900913 EPSG:4326	OpenLayers: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> ] OpenLayers: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> ] KML: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> , <a href="#">kml</a> ]
sf:restricted	EPSG:900913	OpenLayers: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> ]

#### 14.3.1 Viewing

To view the GeoWebCache 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, GeoWebCache may be set up incorrectly. Verify the steps on the [Using GeoWebCache](#) page have been carried out successfully.

#### 14.3.2 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 in to the [Web Administration Interface](#). (See [Interface basics](#) for more information.) After a successful login, the number of layers found and loaded will be displayed.

**These are just quick demos. GeoWebCache also supports:**

- WMTS, TMS, Virtual Earth and Google Maps
- Proxying GetFeatureInfo, GetLegend and other WMS requests
- Advanced request and parameter filters
- Output format adjustments, such as compression level
- Adjustable expiration headers and automatic cache expiration
- RESTful interface for seeding and configuration (beta)

### Reload Configuration:

You can reload the configuration by pressing the following button. The username / password i

Reload Configuration

### 14.3.3 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 can be found on the [Seeding and refreshing](#) page.

## 14.4 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.

### 14.4.1 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.

### 14.4.2 Seeding options

The [GeoWebCache Demo 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 in to the [Web Administration Interface](#). (See [Interface basics](#) for more information.) After a successful login, 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 <b>1</b> and <b>16</b> .
Type of operation	Sets the operation. There are three possible values: <b>Seed</b> (creates tiles, but does not overwrite existing ones), <b>Reseed</b> (like Seed, but overwrites existing tiles) and <b>Truncate</b> (deletes all tiles within the given parameters)
SRS	Specifies the projection to use when creating tiles (default values are <b>EPSG:4326</b> and <b>EPSG:900913</b> )
Format	Sets the image format of the tiles. Can be <b>application/vnd.google-earth.kml+xml</b> (Google Earth KML), <b>image/gif</b> (GIF), <b>image/jpeg</b> (JPEG), <b>image/png</b> (24 bit PNG), and <b>image/png8</b> (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 <b>Zoom stop</b> .
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 <b>Zoom start</b> .
Bounding box	( <i>optional</i> ) 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 <b>Xmin</b> , <b>Ymin</b> , <b>Xmax</b> , and <b>Ymax</b> .

When finished, click **Submit**.

**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.

### 14.4.3 Manually deleting cached content

If you have direct access to the file system on the server, you can also delete the appropriate layers in the cache directory. The structure of the cache directory is `[root] / layer / projection_zoomlevel`.

## 14.5 Troubleshooting

Sometimes errors will occur when requesting data from GeoWebCache. Below are some of the most common reasons.

### 14.5.1 Grid misalignment

Sometimes errors will occur saying 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 set up GeoWebCache integration with the GeoServer WMS. See the section on [Seeding and refreshing](#) for more details.



---

# 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.

## 15.1 Overview

### 15.1.1 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.

### 15.1.2 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.

## 15.2 Quickstart

**Note:** If you are using GeoServer locally, the `GEOSERVER_URL` is usually

`http://localhost:8080/geoserver`

### 15.2.1 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**.

#### Layer Preview

List of all layers configured in GeoServer and provides previews in various formats for each.

<< < 1 > >> Results 1 to 19 (out of 19 items)

Type	Name	Title	Common Formats		All Formats
	nurc:Arc_Sample	A sample ArcGrid file	OpenLayers	KML	Select one
	nurc:Pk50095	Pk50095 is a A raster file accompanied by a spatial data file	OpenLayers	KML	Select one
	nurc:mosaic	Sample PNG mosaic	OpenLayers	KML	Select one
	nurc:Img_Sample	North America sample imagery	OpenLayers	KML	Select one
	sf:archsites	Spearfish archeological sites	OpenLayers	KML GML	Select one
	sf:bugsites	Spearfish bug locations	OpenLayers	KML GML	Select one
	sf:restricted	Spearfish restricted areas	OpenLayers	KML GML	Select one
	sf:roads	Spearfish roads	OpenLayers	KML GML	Select one
	sf:streams	Spearfish streams	OpenLayers	KML GML	Select one
	sf:sfdem	sfdem is a Tagged Image File Format with Geographic Information	OpenLayers	KML	Select one
	tiger:poi	Manhattan (NY) points of interest	OpenLayers	KML GML	Select one

Figure 15.1: *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**.

When Google Earth is finished loading the result will be similar to below.

### 15.2.2 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:

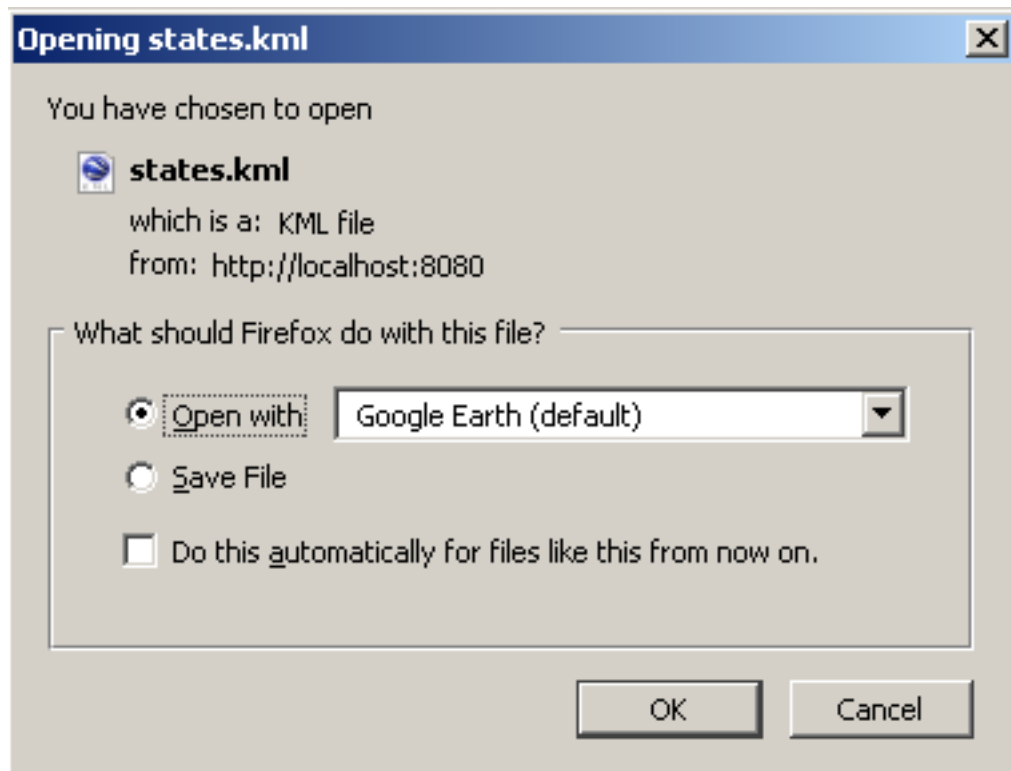


Figure 15.2: *Open with Google Earth*

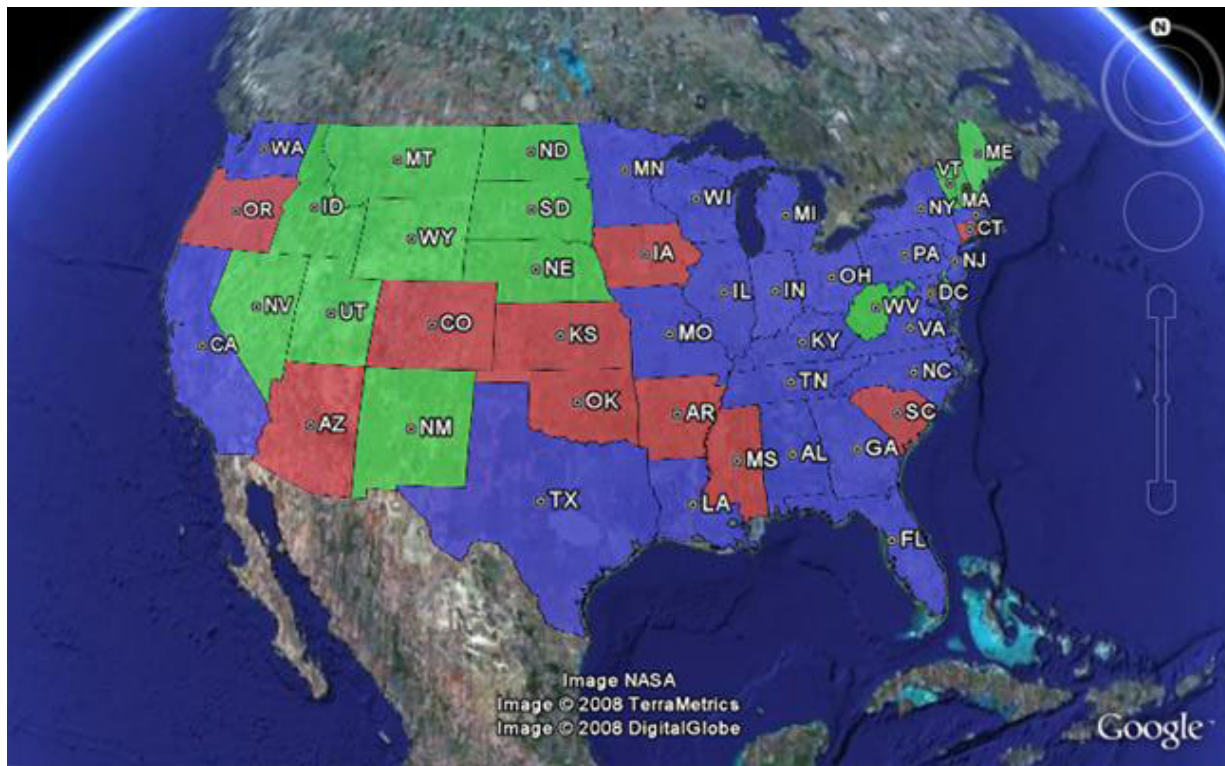


Figure 15.3: The `topp:states` layer rendered in Google Earth

`http://GEOSERVER_URL/wms/kml?layers=topp:states`

### 15.2.3 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.



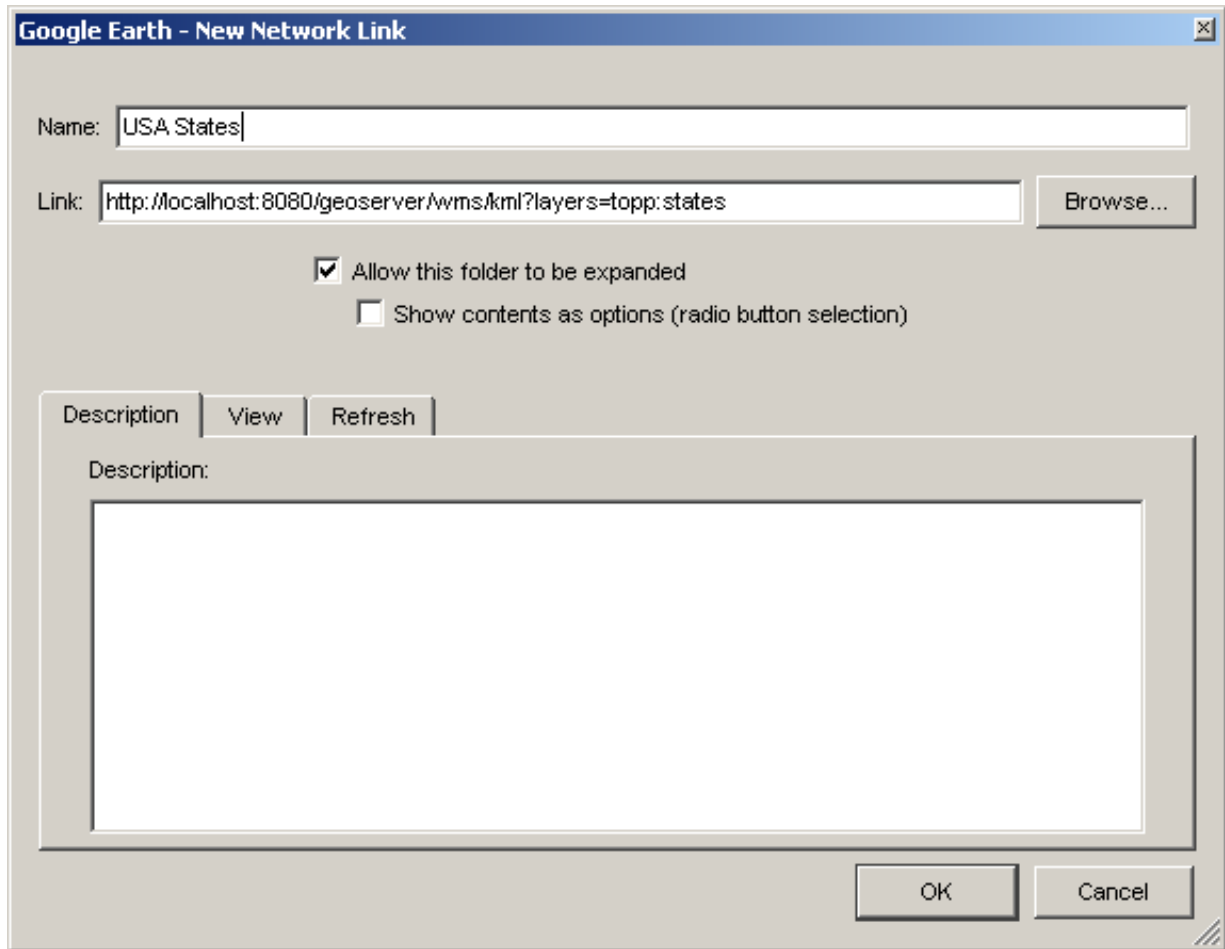


Figure 15.4: *Adding a network link*

## 15.3 KML Styling

### 15.3.1 Introduction

Keyhole Markup Language (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). 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

*[Basic SLD Creation Wizard](#)*

*[Creating SLD by hand](#)*

*[SLD Structure](#)*

*[Points](#)*

*[Lines](#)*

*[Polygons](#)*

*[Text Labels](#)*

*[Descriptions](#)*

### 15.3.2 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.

### 15.3.3 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.

### 15.3.4 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**.

### 15.3.5 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 15.5: 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 15.6: *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>
```

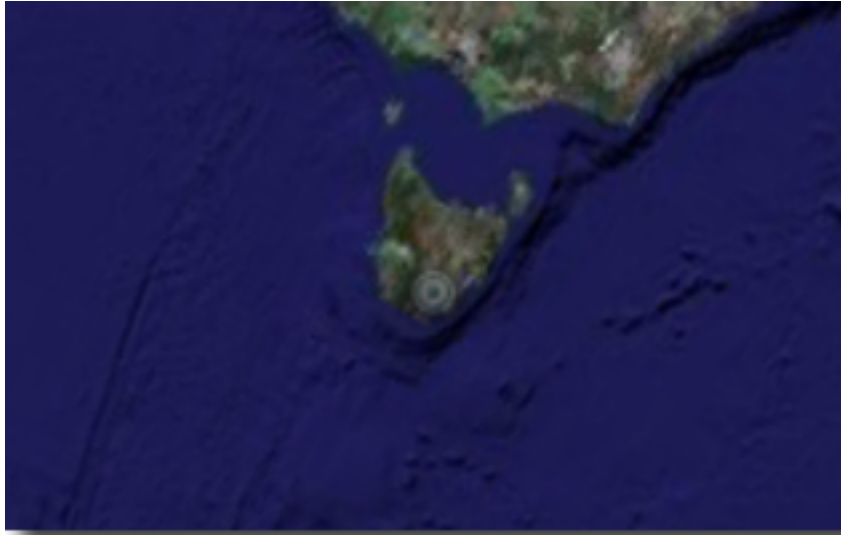


Figure 15.7: Figure 6: Setting the point opacity (0.5 = 50% opaque)

## 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>
```

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

## 15.3.6 Lines

Styles for lines are specified via a `LineSymbolizer`. An empty `LineSymbolizer` element will result in a default KML style:



Figure 15.8: Figure 7: A custom icon for points

```
<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.

```
<LineSymbolizer>
  <Stroke>
    <CssParameter name="stroke">#ff0000</CssParameter>
  </Stroke>
</LineSymbolizer>
```

### 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 15.9: *Figure 9: Default line*



Figure 15.10: *Figure 10: Line rendered with color #ff0000 (100% red)*





Figure 15.11: *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 15.12: Figure 12: A line rendered with 50% opacity

### 15.3.7 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 15.13: 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 15.14: 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 15.15: *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>
```



Figure 15.16: Figure 17: Polygon fill opacity of 0.5 (50% opaque)

### 15.3.8 Text Labels

There are two ways to specify a label for a feature in Google Earth. The first is with Freemarkert 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:

```
${STATE_NAME.value}
```



Figure 15.17: 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>

```



Figure 15.18: Figure 19: Using a TextSymbolizer to display the value of STATE\_NAME

The aspects of the resulting label which can be specified via a TextSymbolizer are *color* and *opacity*.

### 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 15.19: 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 15.20: Figure 21: *TextSymbolizer with opacity 0.5 (50% opaque)*

### 15.3.9 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.

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](#)

## 15.4 Tutorials

### 15.4.1 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:

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



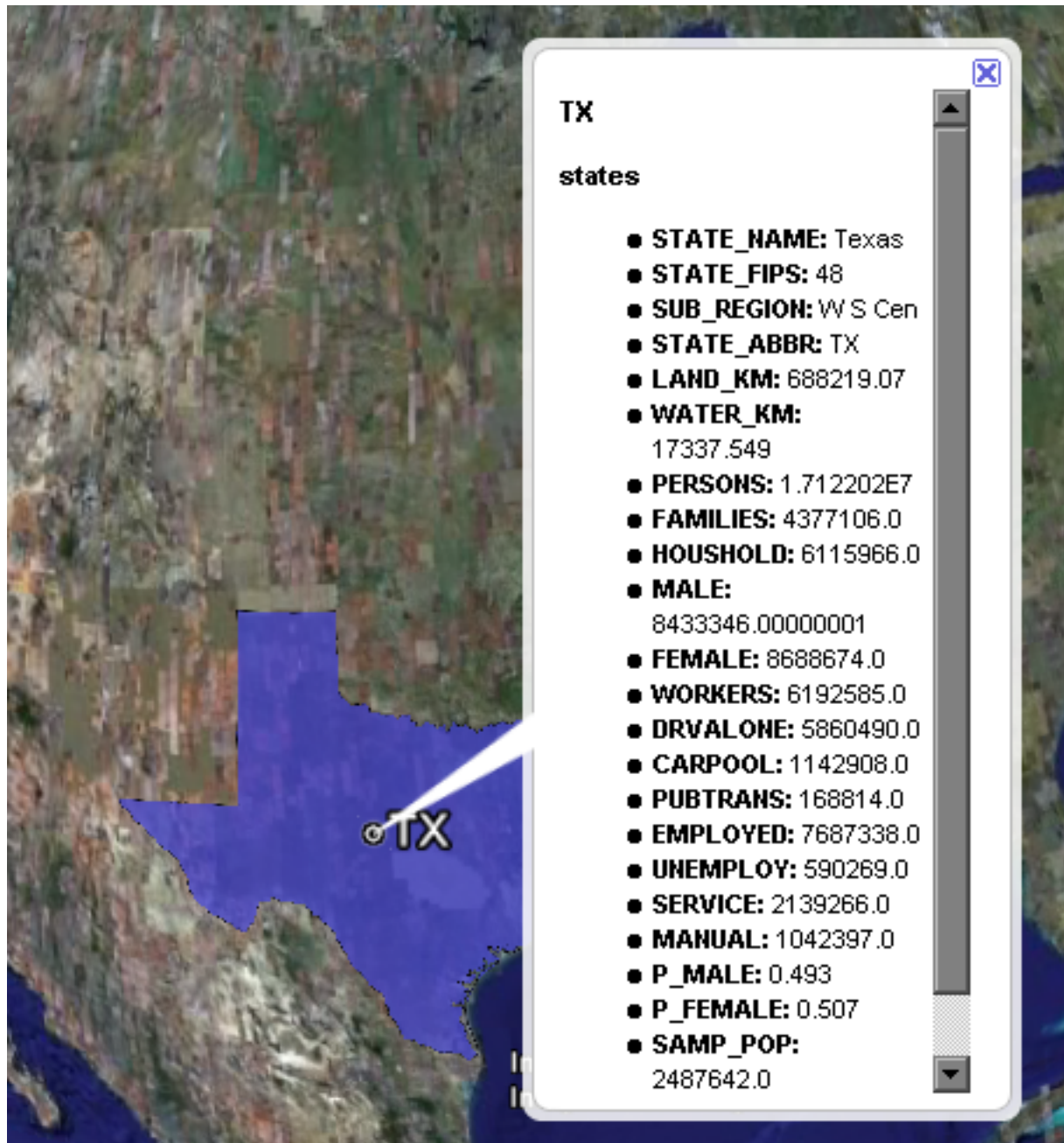


Figure 15.21: Figure 22: Default description for a feature



Figure 15.22: *Figure 23: A custom description*

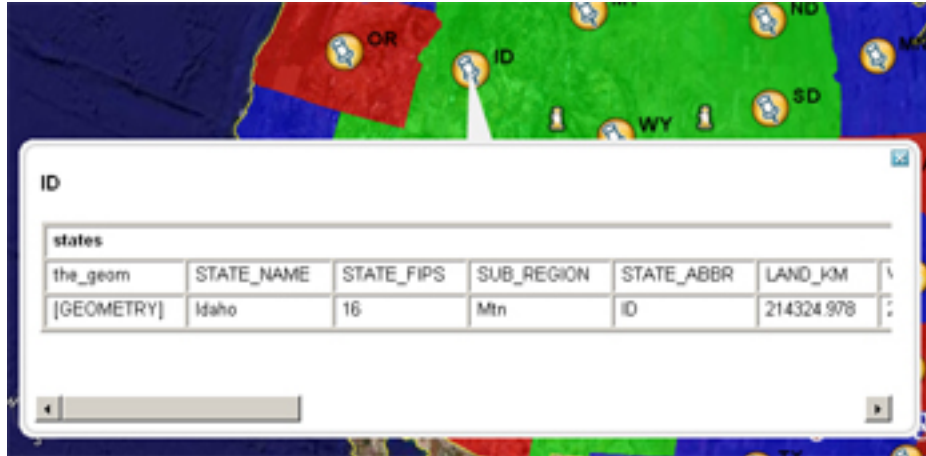


Figure 15.23: The default placemark

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

**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.

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:



Figure 15.24: *Hello World template.*

Provided by the [The Open Planning Project](http://topp.openplans.org).

Homepage of Topp

1. The logo of your organization:

```

```

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.

## Data Content

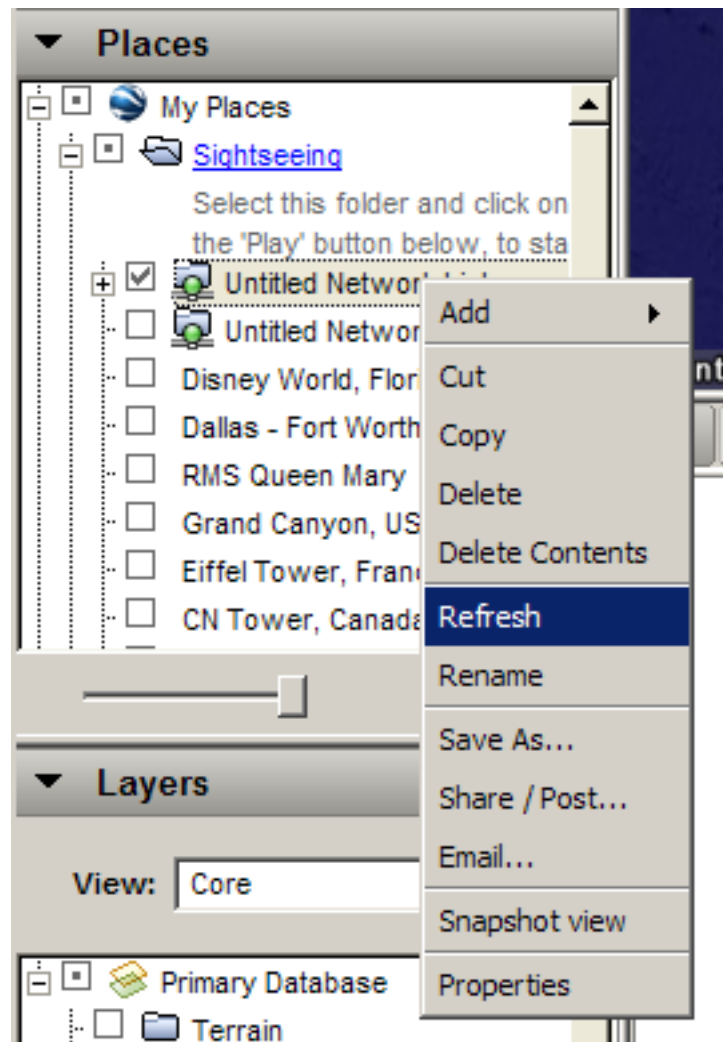
The real power of templates is the ability to easily access content, in the case of features this content is the attributes of features. In a KML placemark description template, there are a number of “template variables” available.

- 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:



Figure 15.25: *Refresh Template*

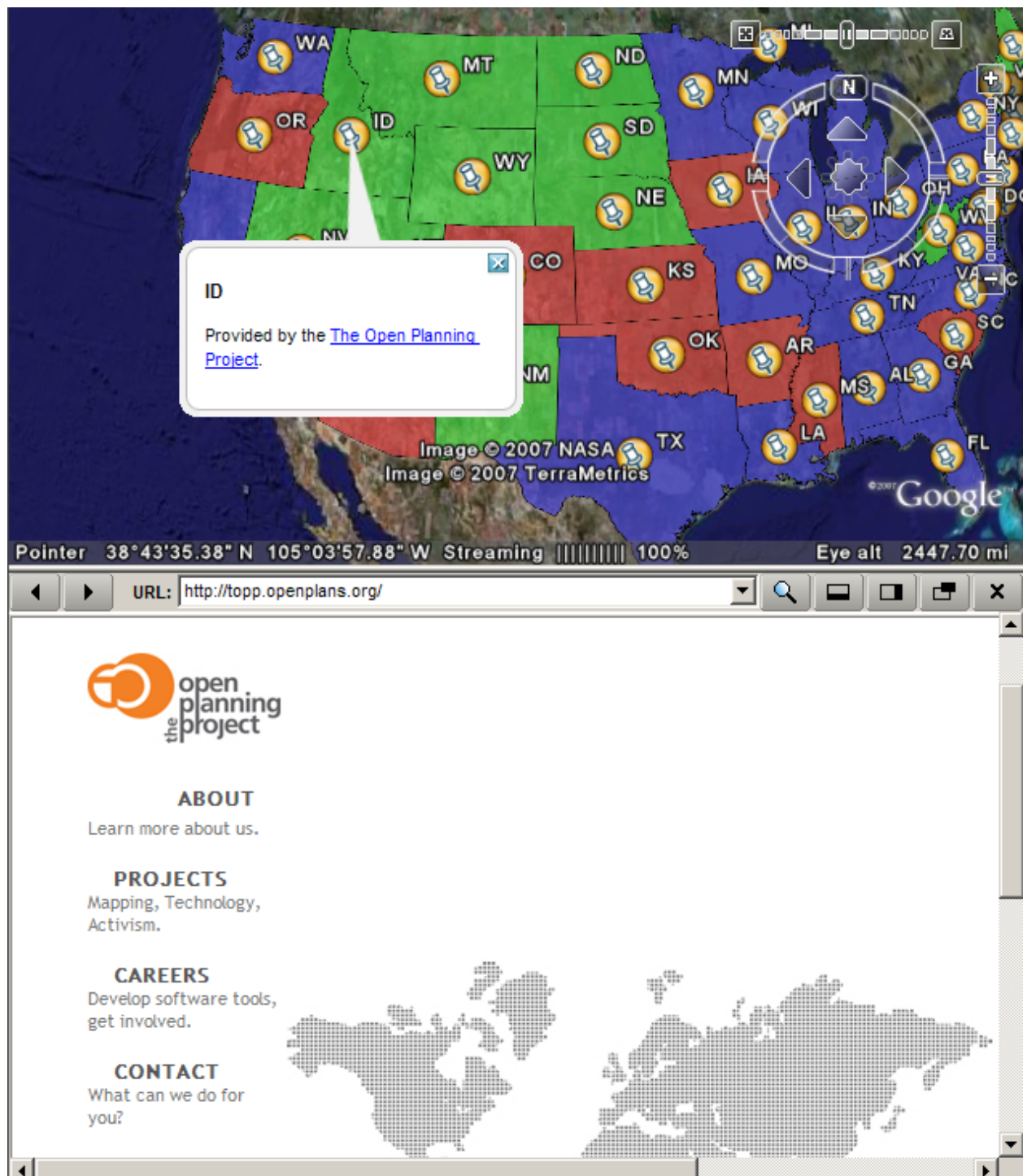


Figure 15.26: Homepage of Topp

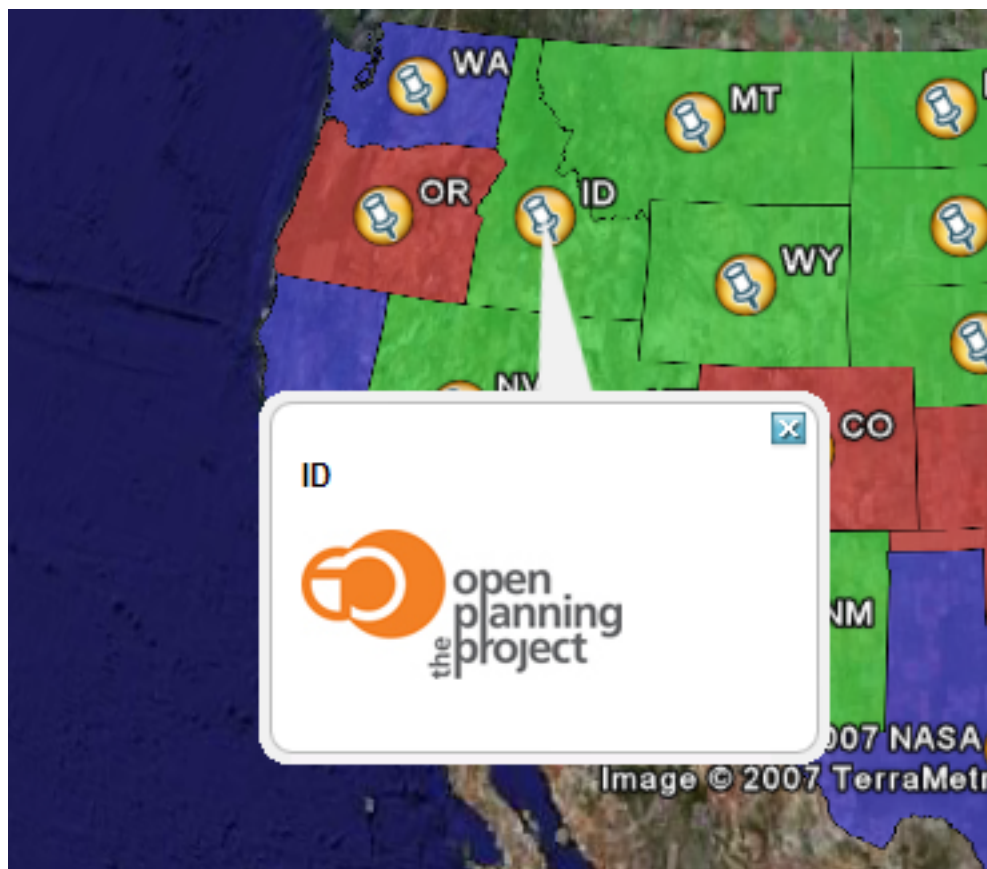


Figure 15.27: *Logo of Topp*

This is feature `${fid}` of type `${typeName}`.

This is a feature of 3.1 of type states.

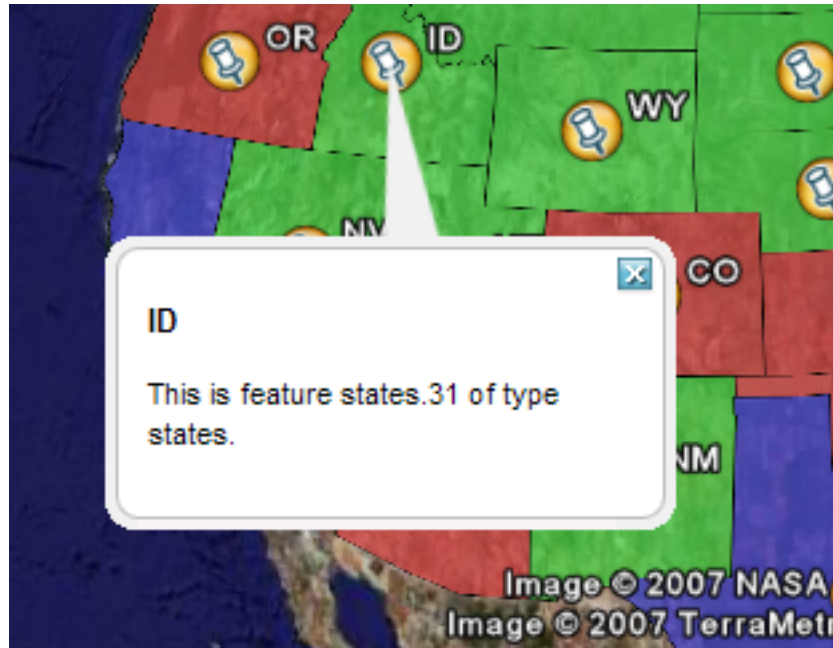


Figure 15.28: *FID*

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.

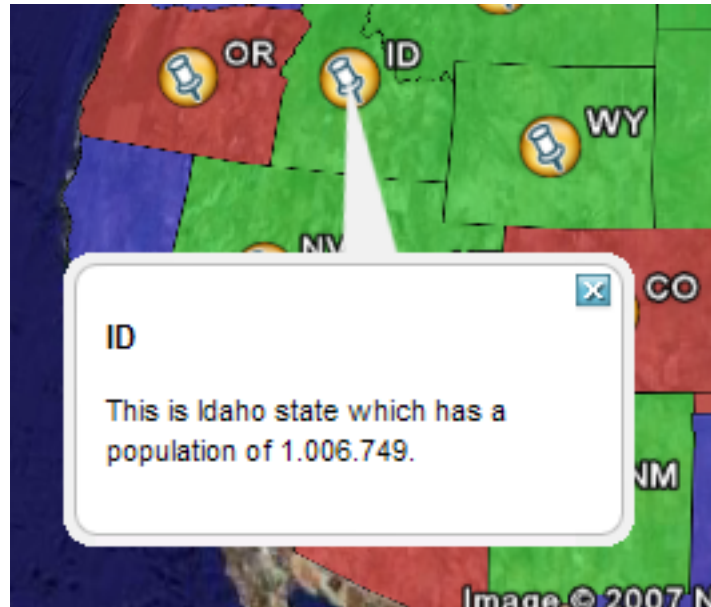
### Attribute Variables

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](http://localhost:8080/geoserver/popup_map/index.html) in your web browser.

Figure 15.29: *Attributes*

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:

```

```

2. Add the following network link in Google Earth:

```
http://localhost:8080/geoserver/wms/kml_reflect?layers=tiger:poi
```

Poi.4

## 15.4.2 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>.

### 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/>'\_.





Figure 15.30: WMS Example

### 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.kml
```

### 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**:

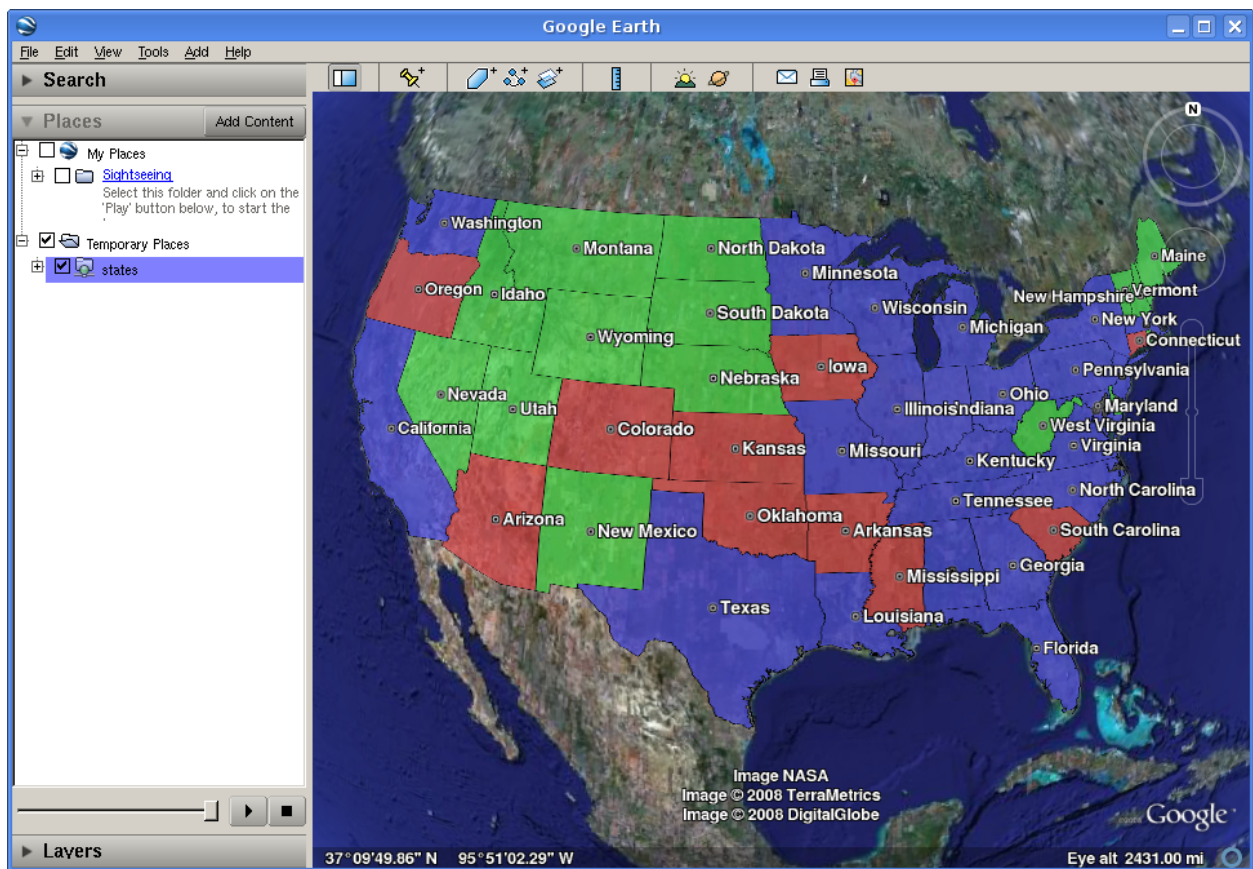


Figure 15.31: *topp:states* in Google Earth

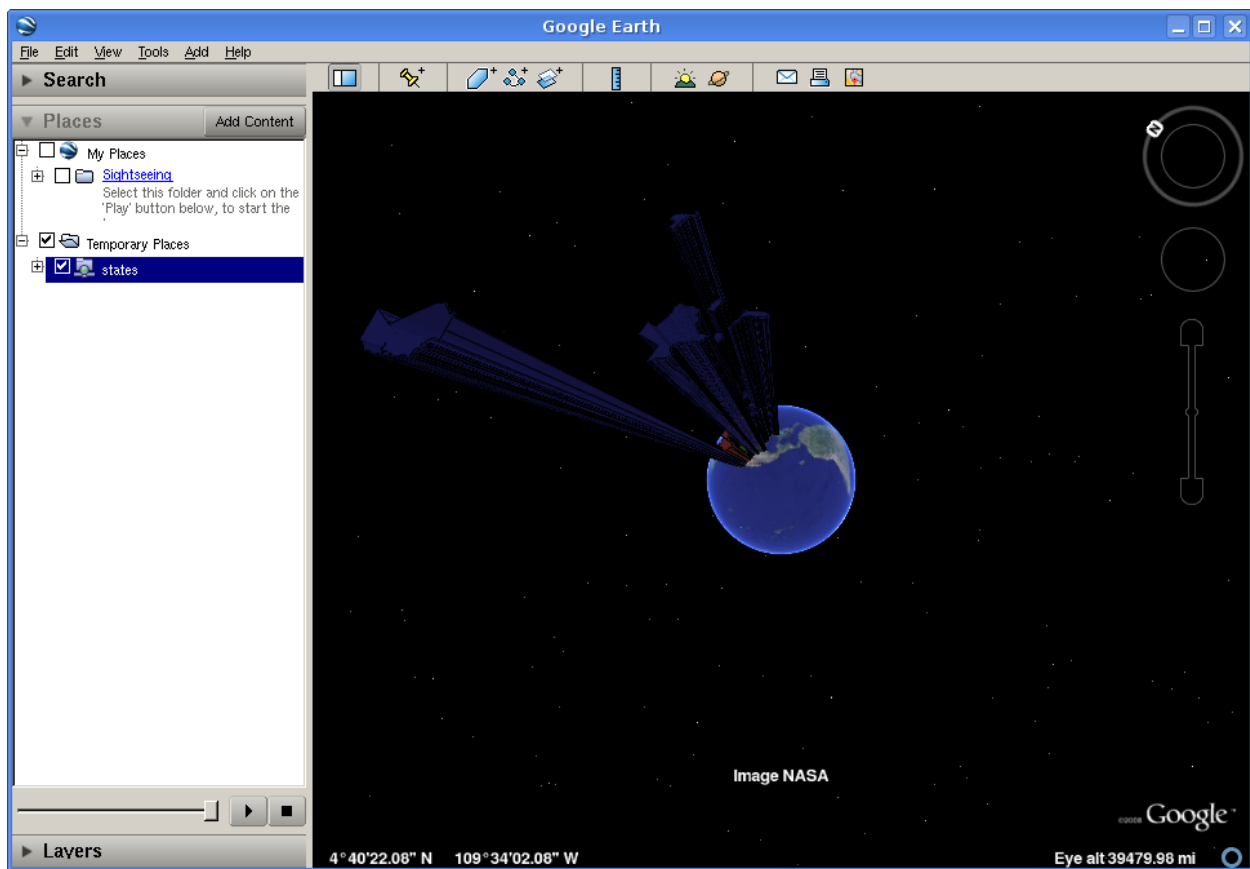


Figure 15.32: *Height by Population*



### 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:

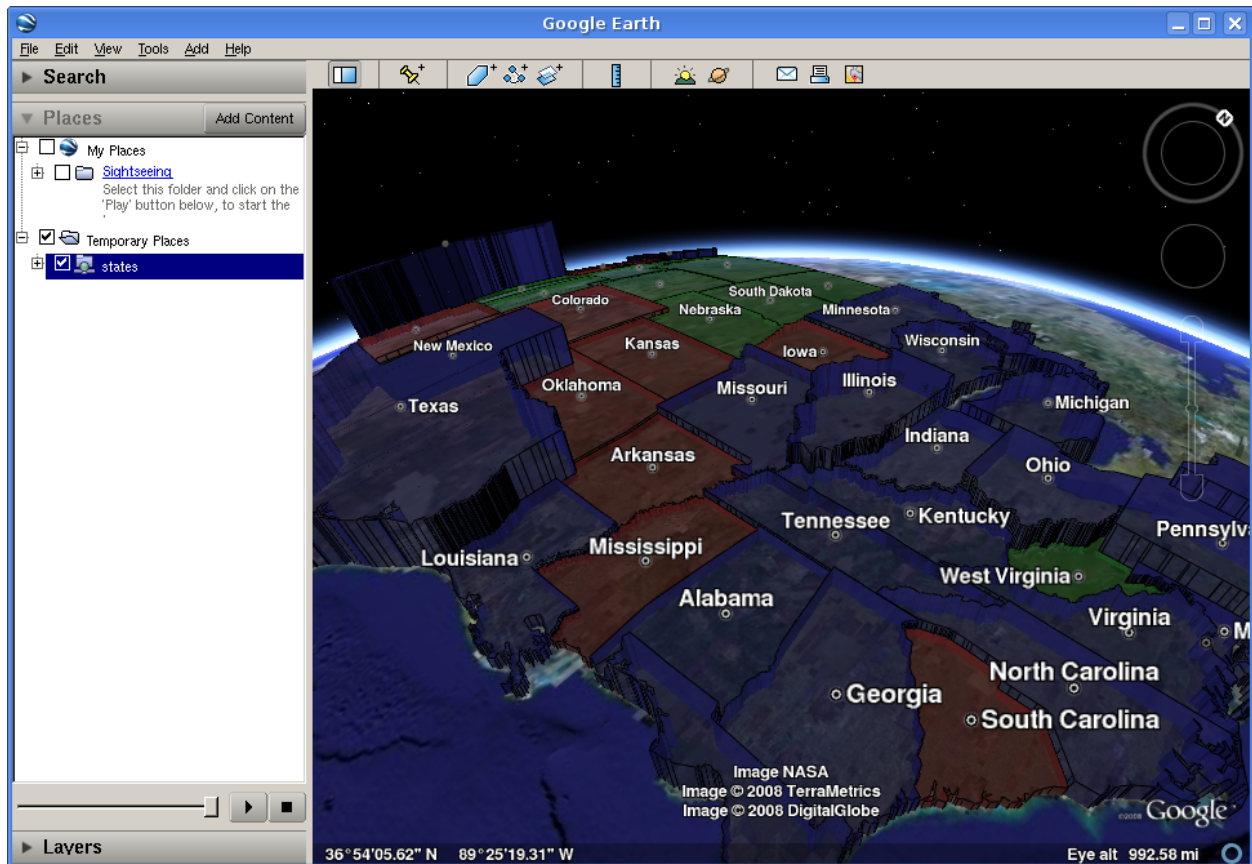


Figure 15.33: *Scaled Height*

## 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.kml
```

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.

<code>altitudeMode</code>	Purpose
<code>altitudeMode</code>	Interpret height as relative to ground level
<code>absolute</code>	Interpret height as relative to sea level
<code>clampToGround</code>	Ignore height entirely

## 15.4.3 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.

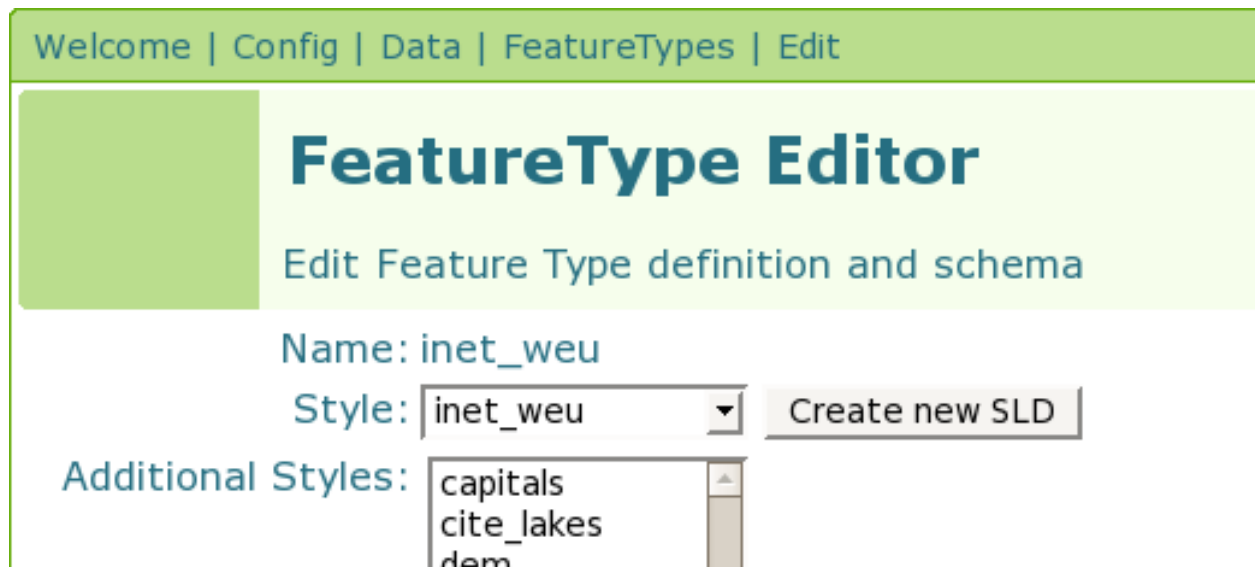
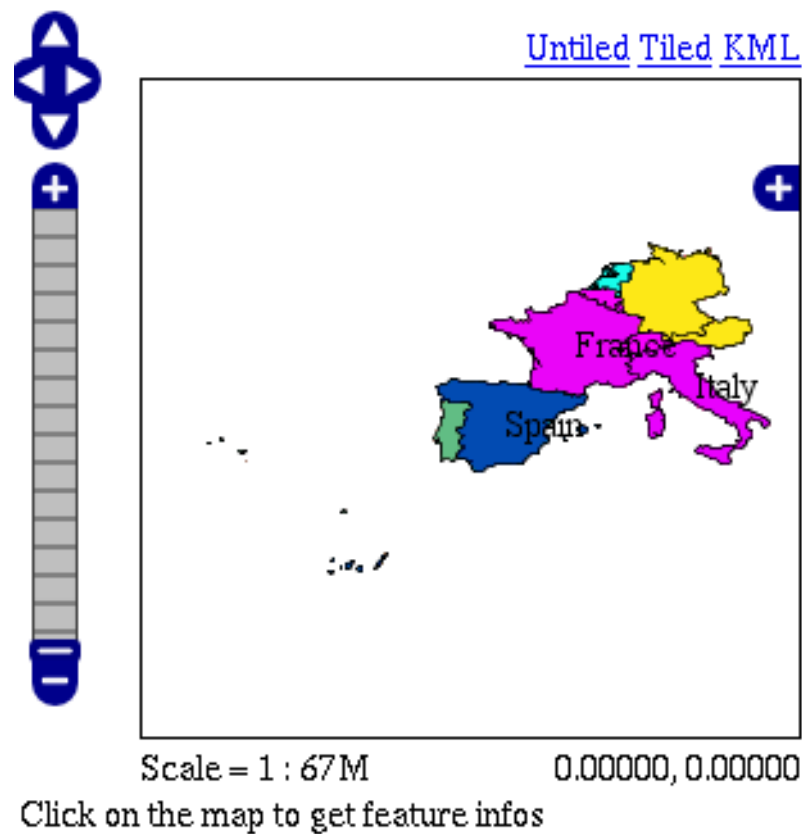
1. Download and unzip **inet\_weu.zip**
2. Configure GeoServer to serve the Shapefile `inet_weu.zip`. (A tutorial is available [Adding a Shapefile](#).)
3. Add the SLD “**inet\_weu.sld** to GeoServer. ( A tutorial is available for [Styling a Map](#))
4. Set the style of the feature type added in step 2 to the style added in step 3

## 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](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:

## 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:

Figure 15.34: *Style*Figure 15.35: *Setup*

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')}
```

1. 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\\_reflect?layers=inet\\_weu&legend=true](http://localhost:8080/geoserver/wms/kml_reflect?layers=inet_weu&legend=true). This should cause Google Earth to open.
1. In Google Earth, adjust the time bar so that it captures a time interval that is approximately 1 year wide
1. 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 15.36: Google Earth



Figure 15.37: Google Earth Time Bar



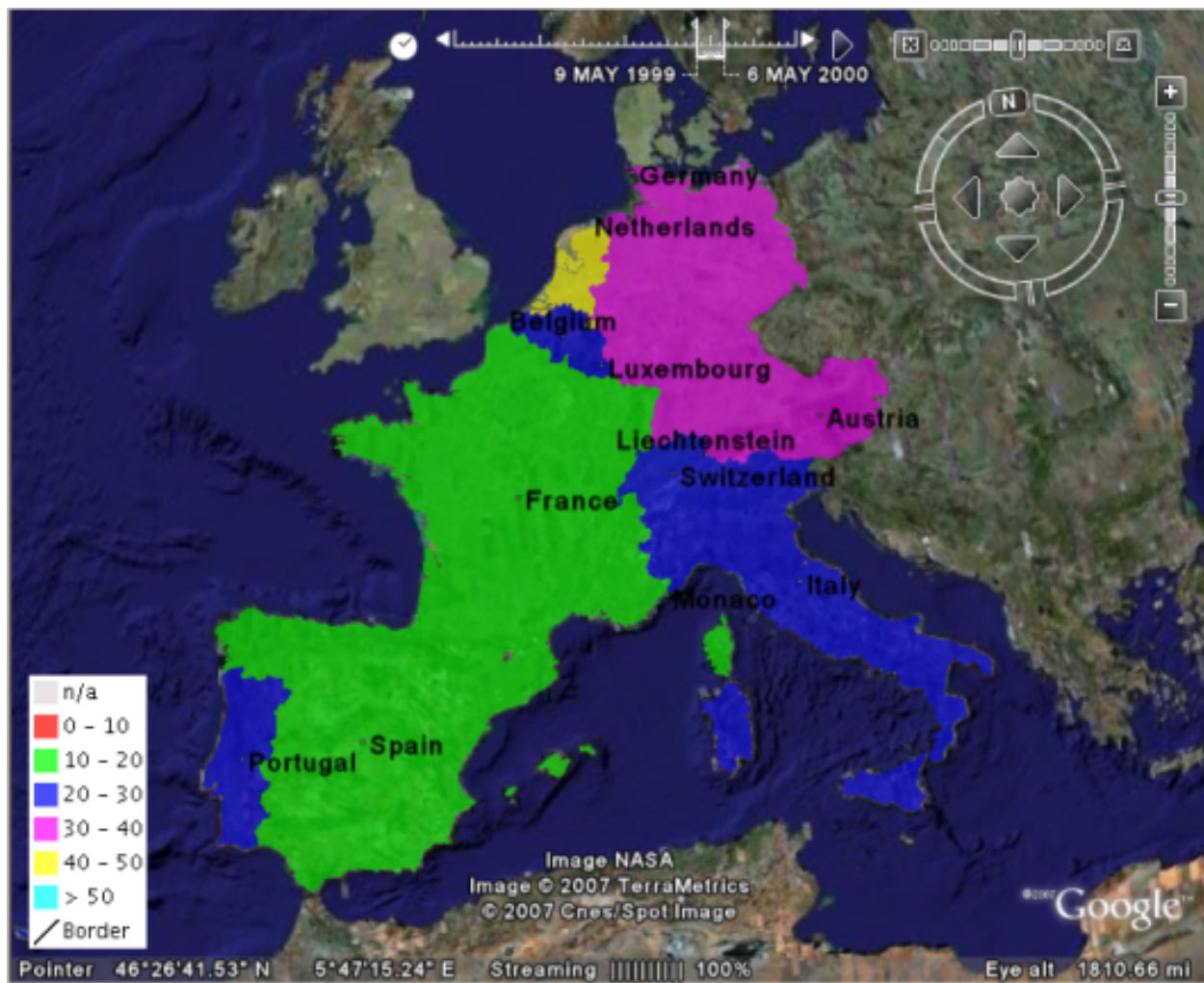


Figure 15.38: *Sliding the Time Bar*

```
${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.

### 15.4.4 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.



Layer name:	Grids Sets:	
nunc:Arc_Sample <a href="#">Seed this layer</a>	EPSG:900913	OpenLayers: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> ]
	EPSG:4326	OpenLayers: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> ] KML: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> , <a href="#">kml</a> ]
nunc:Img_Sample <a href="#">Seed this layer</a>	EPSG:900913	OpenLayers: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> ]
	EPSG:4326	OpenLayers: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> ] KML: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> , <a href="#">kml</a> ]
nunc:Pk50095 <a href="#">Seed this layer</a>	EPSG:900913	OpenLayers: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> ]
	EPSG:4326	OpenLayers: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> ] KML: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> , <a href="#">kml</a> ]
nunc:mosaic <a href="#">Seed this layer</a>	EPSG:900913	OpenLayers: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> ]
	EPSG:4326	OpenLayers: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> ] KML: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> , <a href="#">kml</a> ]
sf:archsites <a href="#">Seed this layer</a>	EPSG:900913	OpenLayers: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> ]
	EPSG:4326	OpenLayers: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> ] KML: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> , <a href="#">kml</a> ]
sf:bugsites <a href="#">Seed this layer</a>	EPSG:900913	OpenLayers: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> ]
	EPSG:4326	OpenLayers: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> ] KML: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> , <a href="#">kml</a> ]
sf:restricted	EPSG:900913	OpenLayers: [ <a href="#">png</a> , <a href="#">gif</a> , <a href="#">png8</a> , <a href="#">jpeg</a> ]

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.

## 15.4.5 Super-Overlays and Extrudes with Building Data

## 15.5 Features

This section delves into greater detail about the various functionality and options possible with KML output and Google Earth.

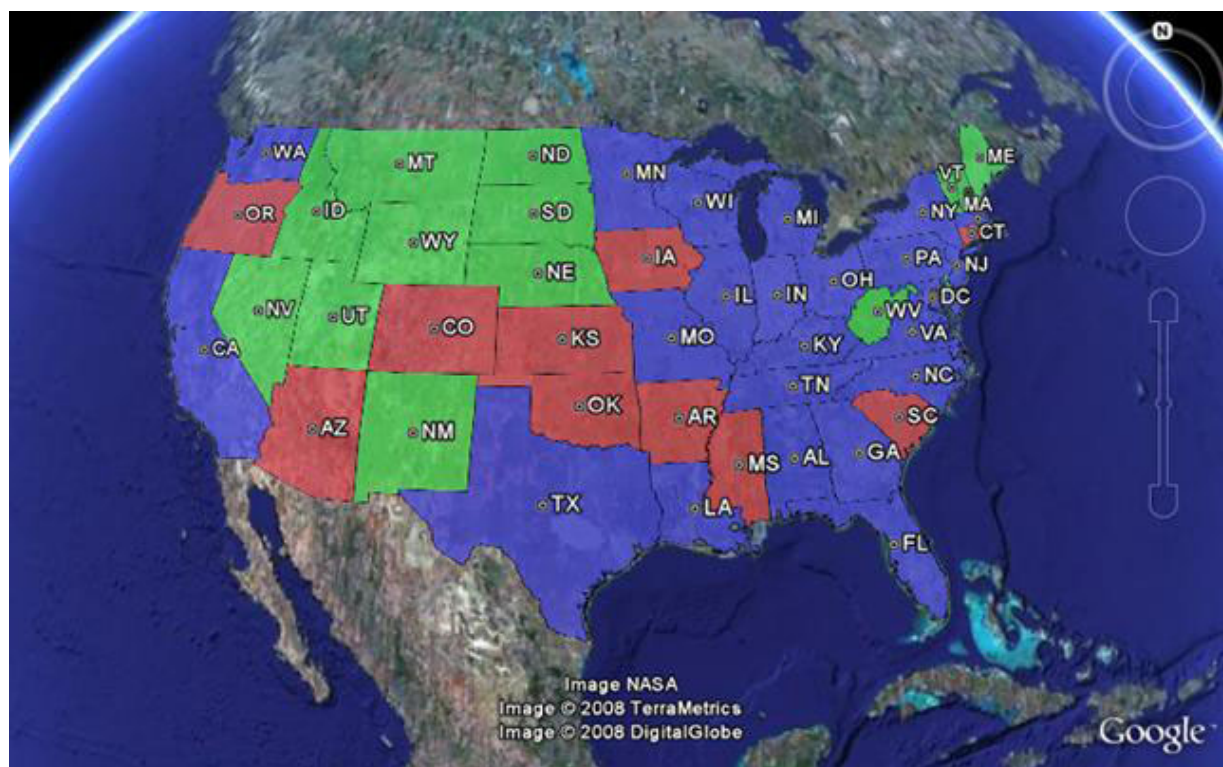
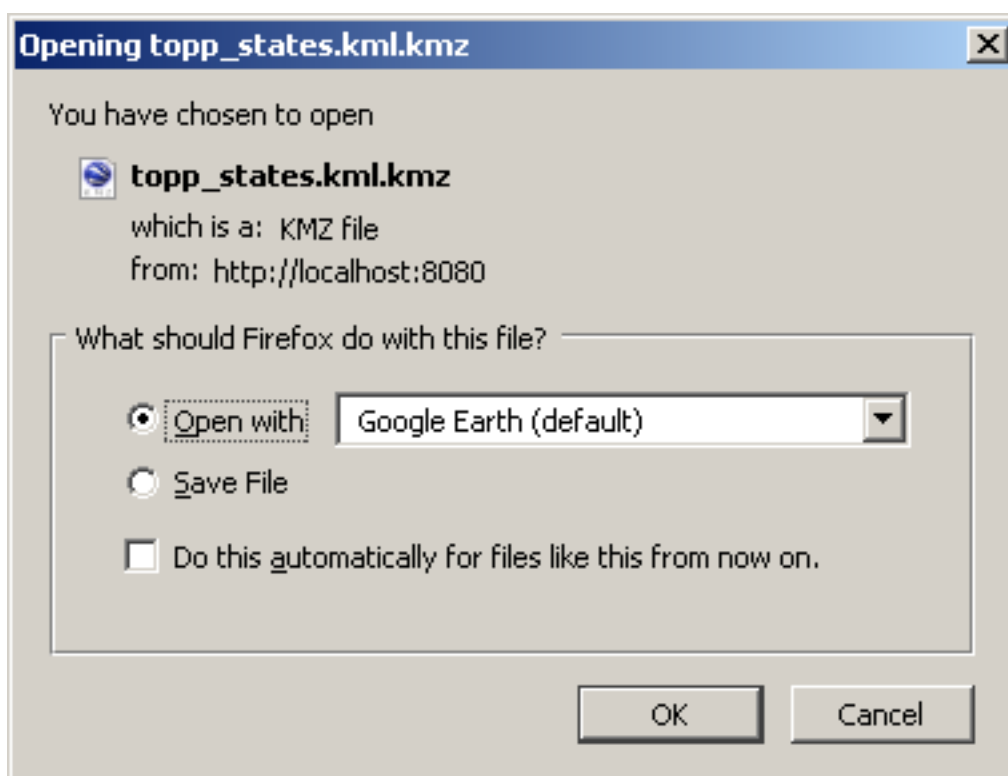
### 15.5.1 KML Reflector

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.
```

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	application/vnd.google-earth.kmz+xml
width	256
height	256
bbox	<layer bounds>
kmattr	true
kmplacemark	false
kmscore	50
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 <code>kmscore</code> . Please see the section on <a href="#">KML Scoring</a> 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 <a href="#">KML Super-Overlays</a> 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 <a href="#">KML Regionation</a> for more information.
hybrid	Displays both raster and vector data at all times.

## 15.5.2 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
```

## 15.5.3 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.

## 15.5.4 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](#)



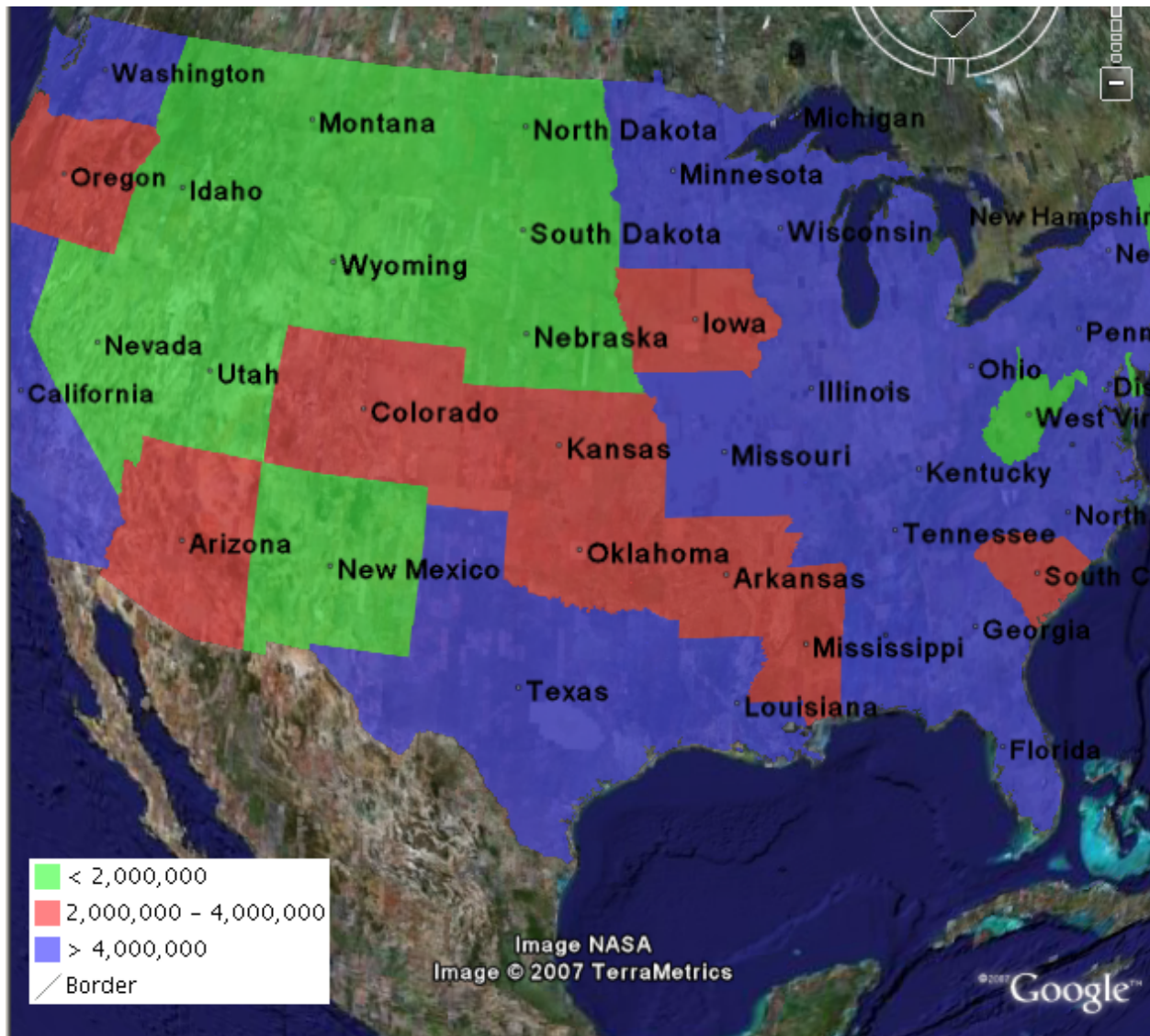
### 15.5.5 KML Legends

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:



### 15.5.6 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”, “dis-joint”), “LIKE” statements, nulls, and more.

## Filter

The 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_reflect?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_reflect?layers=topp:states&FILTER=%3CFilter%3E%3CPropertyIsBet
```

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+150000
```

This query will return all the states in the US with areas between 100,000 and 150,000 km<sup>2</sup>.

## 15.5.7 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 (originally 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.

### 15.5.8 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 <code>random</code> strategy is used. If the data consists of lines or polygons, the <code>geometry</code> strategy is used.
external-sort	Creates a temporary auxiliary database within GeoServer. It takes slightly extra time to build the index upon first request.
native-sort	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.





## 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:

$$\text{maximum number of features} = 10^{(\text{kmscore}/15)}$$

The following table shows the values of this threshold for various values of the `kmscore` parameter:

<b>kmscore</b>	<b>Maximum # of features</b>
0	Force overlay/raster output
10	4
20	21
30	100
40	Approx. 450
50	( <i>default</i> ) 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`.





---

# Extensions

---

Extensions are modules that add various bits of functionality to GeoServer. They need to be installed separately from GeoServer.

This section describes the various extensions available to GeoServer. For information about extensions that add support for additional data formats, such as ArcSDE or SQL Server, see the [Working with Data](#) section.

## 16.1 GeoSearch

### 16.1.1 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](#).

### 16.1.2 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/>.

### 16.1.3 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.

### 16.1.4 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.

### 16.1.5 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.

## 16.2 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:

```

<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:

```

<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:

```

<map name="mymap">
  <area shape="poly" coords="536,100 535,100 534,101 533,101 532,102" onmouseover="onOver('<featureid>')"/>
  <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.

## 16.3 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.

### 16.3.1 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.

### 16.3.2 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).

### 16.3.3 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.

## 16.4 Cross layer filtering

The “querylayer” module adds a few extra filter functions that allow for cross layer filtering, that is, the ability to find in layer A features that have a certain relationship with features in layer B. This can be used, for example, to find all bus stops within a certain distance from a shop, or all coffe shops in a certain city district. Since filter functions are widely supported in GeoServer this cross layer filtering can be applied in SLDs, CQL filters and WFS requests alike.

### 16.4.1 Installing the ‘querylayer’ module

1. Download the ‘querylayer’ extension corresponding to your version of GeoServer.

**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. To check the module is properly installed get the WFS 1.1 capabilities from the GS home page, the Filter\_Capabilities section should contain a reference to a new function named ‘queryCollection’

```

1  ...
2  <ogc:Filter_Capabilities>
3      ...
4      <ogc:ArithmeticOperators>
5          ...
6          <ogc:Functions>
7              <ogc:FunctionNames>
8                  ...
9                  <ogc:FunctionName nArgs="-1">queryCollection</ogc:FunctionName>
10                 <ogc:FunctionName nArgs="-1">querySingle</ogc:FunctionName>
11                 ...
12             </ogc:FunctionNames>
13         </ogc:Functions>
14     </ogc:ArithmeticOperators>
15 </ogc:Scalar_Capabilities>
16 ...
17 </ogc:Filter_Capabilities>
18 ...

```

## 16.4.2 Function reference

Name	Arguments	Description
querySingle	layer:String, attribute:String, filter:String	Queries the specified layer ``applying the specified (E)CQL ``filter and returns the value of attribute from the first feature in the result set. The layer name should be qualified (e.g. topp:states), the filter can be INCLUDE if no filtering is desired
queryCollection	layer:String, attribute:String, filter:String	Queries the specified layer ``applying the specified (E)CQL ``filter and returns the list of the values from attribute out of every single feature in the result set. The layer name should be qualified (e.g. topp:states), the filter can be INCLUDE if no filtering is desired. Will throw an exception if too many results are being collected (see the memory limits section for details)
collectGeometries	geometries: a list of Geometry objects	Turns the list of geometries into a single Geometry object, suitable for being used as the reference geometry in spatial filters. Will throw an exception if too many coordinates are being collected (the results of queryCollection cannot be used as is)

## 16.4.3 Optimizing the module speed

In order to have the module run at full speed on the 2.1.x series it is necessary to add the following parameter as a system variable when starting the Java Virtual Machine:

```
-Dorg.geotools.filter.function.simplify=true
```

This will make sure the functions are evaluated just once per query instead of once per feature matched by the filter. The flag is not necessary on trunk (2.2.x series) and hopefully this behavior will become the default on 2.1.x as well.

## 16.4.4 Memory limits

The query and geometry collection functions are not really performing a database style join, instead they do execute a query against the layer every time they are executed and load the result fully in memory. Both `queryCollection` and `collectGeometries` are thus at risk of filling up the server memory with data if the layer being queried is large, or if the geometries being collected are few but very large. Since this might threaten the server stability there are built in limits to what can be collected:

- at most 1000 features will be collected by `queryCollection`
- at most 37000 coordinates (1MB worth of Coordinate objects) will be collected by `collectGeometries`

Both limits can be overridden by setting appropriate values either as system variable, servlet context variables, or environment variables:

- set `QUERY_LAYER_MAX_FEATURES` to alter the max number of features collected by `queryCollection`
- set `GEOMETRY_COLLECT_MAX_COORDINATES` to alter the max number of coordinates collected by `collectGeometries`

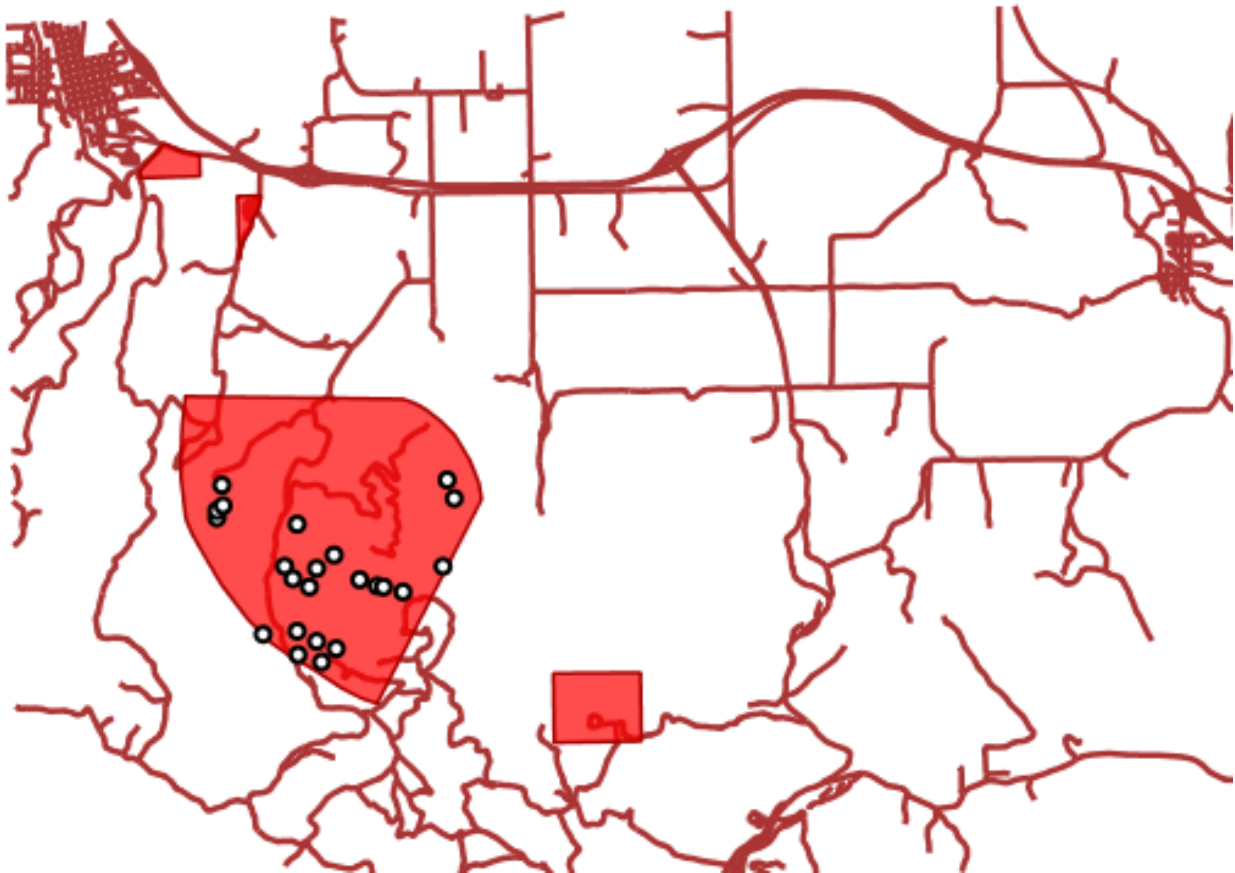
### 16.4.5 WMS Examples

The following examples use the `sf:bugsites`, `sf:roads` and `sf:restricted` demo layers available in the standard GeoServer download.

**Get only the bugsites overlapping the restricted area whose category is "3".** The CQL filter on bugsites is `INTERSECTS(the_geom, querySingle('restricted', 'the_geom', 'cat = 3'))`, the full request is:

<http://localhost:8080/geoserver/wms?LAYERS=sf%3Aroads%2Csf%3Arestricted%2Csf%3Abugsites&STYLES=&FORMAT=application/vnd.openlayers>

and the result looks like:



**Get all bugsides within 200 meters from roads.** The CQL filter looks like `DWITHIN(the_geom, collectGeometries(queryCollection('sf:roads', 'the_geom', 'INCLUDE')), 200, meters)`, the full request is:

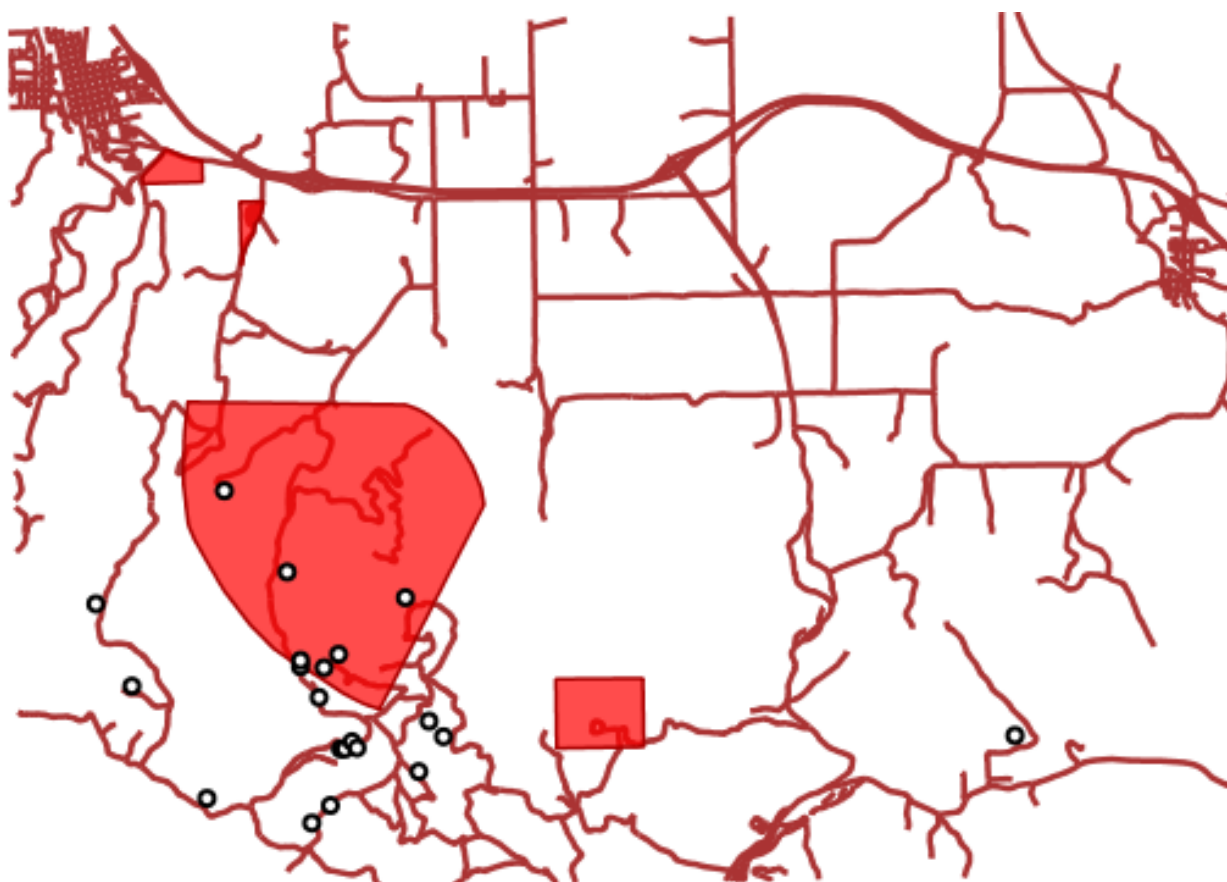
<http://localhost:8080/geoserver/wms?LAYERS=sf%3Aroads%2Csf%3Arestricted%2Csf%3Abugsites&STYLES=&FORMAT=application/vnd.openlayers>

and the result looks like:

### 16.4.6 WFS Examples

The following examples use the `sf:bugsites`, `sf:roads` and `sf:restricted` demo layers available in the standard GeoServer download.

**Get only the bugsites overlapping the restricted area whose category is "3":**





```

1 <wfs:GetFeature xmlns:wfs="http://www.opengis.net/wfs"
2   xmlns:sf="http://www.openplans.org/spearfish"
3   xmlns:ogc="http://www.opengis.net/ogc"
4   service="WFS" version="1.0.0">
5   <wfs:Query typeName="sf:bugsites">
6     <ogc:Filter>
7       <ogc:Intersects>
8         <ogc:PropertyName>the_geom</ogc:PropertyName>
9         <ogc:Function name="querySingle">
10           <ogc:Literal>sf:restricted</ogc:Literal>
11           <ogc:Literal>the_geom</ogc:Literal>
12           <ogc:Literal>cat = 3</ogc:Literal>
13         </ogc:Function>
14       </ogc:Intersects>
15     </ogc:Filter>
16   </wfs:Query>
17 </wfs:GetFeature>

```

Get all bugsites within 200 meters from roads:

```

1 <wfs:GetFeature xmlns:wfs="http://www.opengis.net/wfs"
2   xmlns:sf="http://www.openplans.org/spearfish"
3   xmlns:ogc="http://www.opengis.net/ogc"
4   service="WFS" version="1.0.0">
5   <wfs:Query typeName="sf:bugsites">
6     <ogc:Filter>
7       <ogc:DWithin>
8         <ogc:PropertyName>the_geom</ogc:PropertyName>
9         <ogc:Function name="collectGeometries">
10           <ogc:Function name="queryCollection">
11             <ogc:Literal>sf:roads</ogc:Literal>
12             <ogc:Literal>the_geom</ogc:Literal>
13             <ogc:Literal>INCLUDE</ogc:Literal>
14           </ogc:Function>
15         </ogc:Function>
16         <ogc:Distance units="meter">100</ogc:Distance>
17       </ogc:DWithin>
18     </ogc:Filter>
19   </wfs:Query>
20 </wfs:GetFeature>

```

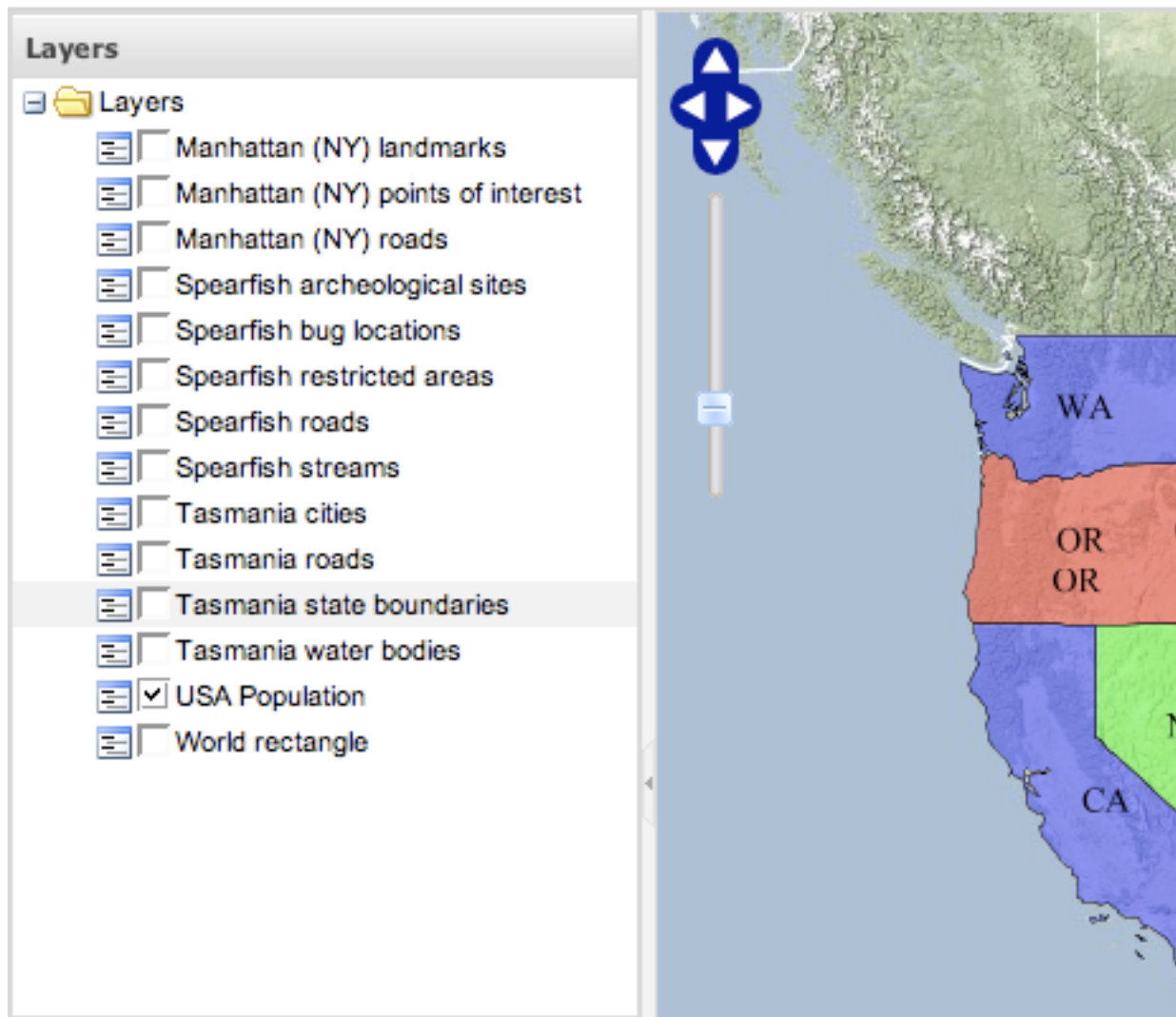
## 16.5 GeoExt Styler

### 16.5.1 Installation

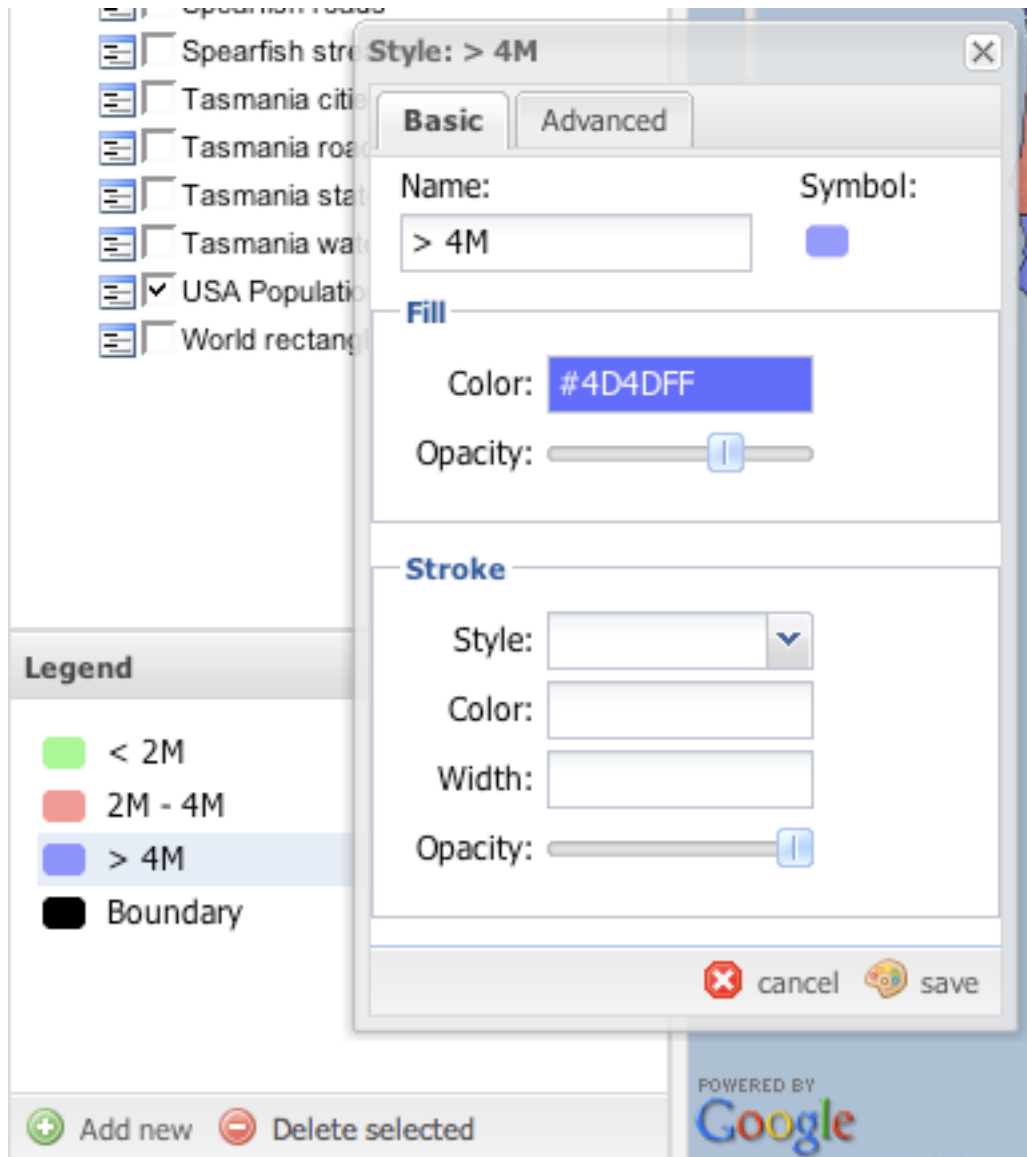
1. Download the REST plugin for your version of GeoServer from the [download page](#) .
2. Unzip the archive into the WEB-INF/lib directory of the GeoServer installation.
3. Restart GeoServer
4. Download the GeoExt Styler extension from [here](#) (it says 1.7.3 but the version number doesn't matter. Soon there will be an updated release)
5. Unzip the archive into the *www/* directory of the GeoServer data directory.

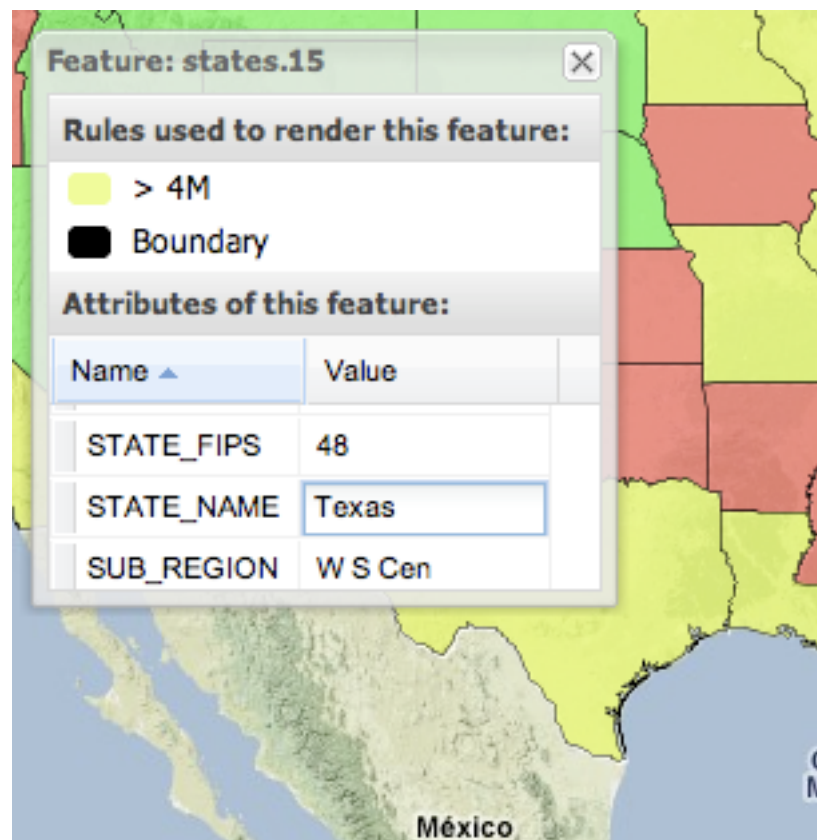
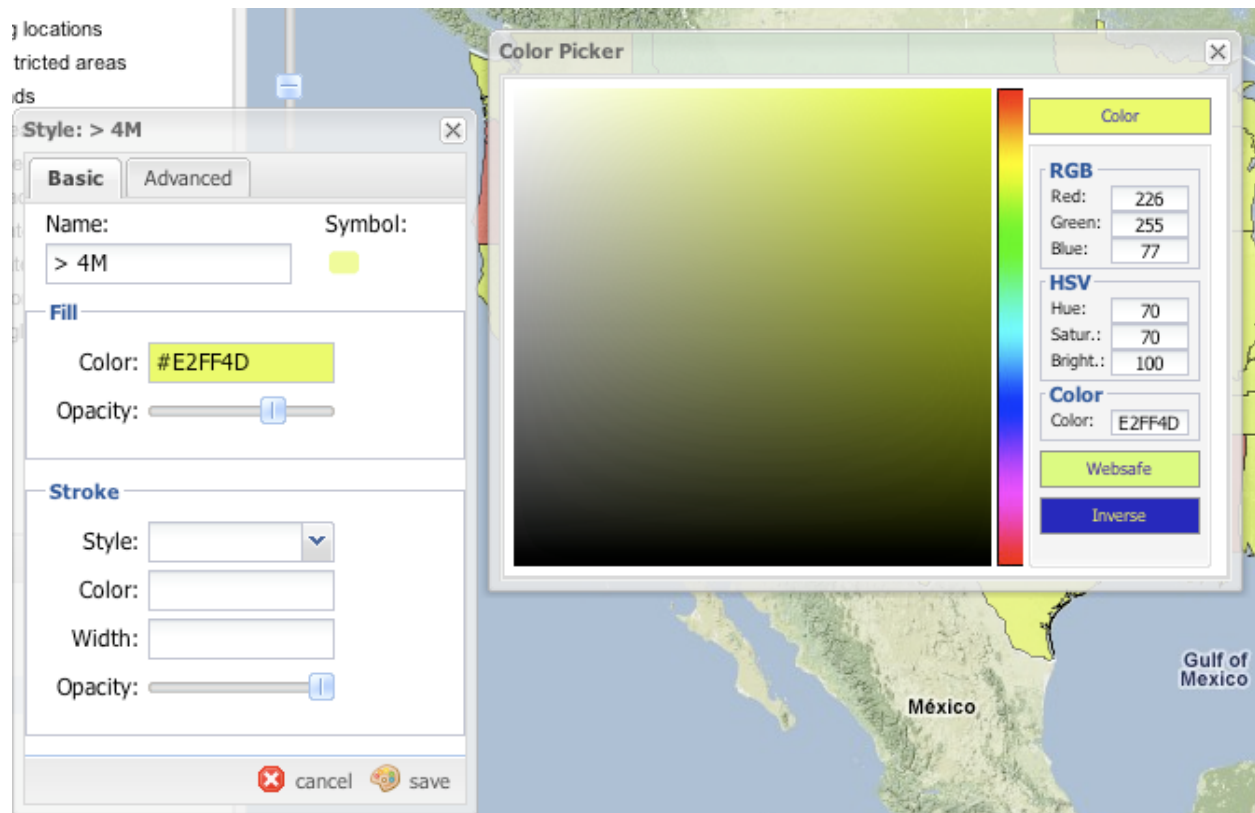
## 16.5.2 Usage

1. Visit <http://localhost:8080/geoserver/www/styler/index.html>
2. Use the “Layers” panel to select a layer to style.



1. In the “Legend” panel select a rule by clicking on it.
2. Change the color by clicking in the color box.
3. Click on a feature to view information about its attributes and which rules applied to it.





## 16.6 WFS Versioning

### 16.6.1 Introduction

One of GeoServer's goals is to help bring the types of collaboration of open source software to the geospatial domain. Just like people across the world form communities and governance structures to build software together, we hope to enable similar things to happen with the creation and maintenance of geospatial data. In the software domain there are a variety of tools - IDE's, version control, bug trackers, etc. that help make this possible. GeoServer hopes to help provide tools that better enable geospatial collaboration.

The Transaction portion of the WFS Standard (also known as WFS-T) specifies an open protocol for inserting, deleting and updating geospatial information. Which is a great start to enable a wide variety of tools, both web-based and desktop, to edit the same database. But it falls short in all but the most controlled environments, as it's too easy to mess things up for everyone. GeoServer's [Security](#) system is one step in this direction, to help people control their environment.

WFS Versioning is a set of extensions to the WFS protocol to keep track of 'versions' of edits. This enables wiki-style editing of geospatial information, by keeping the history of all changes to the data.

### 16.6.2 Protocol

WFS Versioning (WFS-V) is **not** an official OGC standard, and has not yet entered the standard process. In time the GeoServer community hopes to try to standardize it, but first is focused on getting real world implementations. It is designed to be as compatible with WFS-T as possible, reusing elements and extending operations. In the future a REST equivalent may be implemented.

WFS-V adds two new operations and one new action on the Transaction operation.

Operation	Description
GetLog	Returns summaries of the changes that have taken place over a set of constraints.
GetDiff	Retrieves the actual changes that have occurred.
Rollback (optional)	A convenience Transaction element, to revert changes to a previous revision.

For more information on the extensions to WFS see the detailed [draft specification](#).

### 16.6.3 Implementation

GeoServer has completed a first phase implementation that is working in prototype situations. See [this blog post](#) for more information on one of the working prototypes. It has not been used in production, so don't expect it to work perfectly. But the extension is available for download, and we appreciate any help we can get, even just trying it out and reporting bugs.

The implementation currently just works against PostGIS. But there are future plans to have it work seamlessly with Oracle Workspace Manager and ArcSDE Versioning. Doing this may involve adjusting the protocol. On the client side the protocol has been implemented in OpenLayers. And it will work transparently against any WFS-T client, though a non-versioning aware client won't be able to supply commit messages or make use of the advanced operations. But all changes it does make will be versioned.

See the [Phase one implementation proposal](#) section of the wiki for more information on what's been built. The main [Versioning WFS](#) page also has a lot of information on the background and decisions in the implementation, and will be the point of collaboration in the future.

### 16.6.4 Trying it out

After the next round of work on WFS-V we will be publishing some docs to get anyone started. In the meantime advanced users can likely figure things out by looking at some of the older information on the wiki - [trying early WFS-V prototype](#) and the [Versioning](#) section from the 2007 Foss4g workshop.

## 16.7 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.

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>.

### 16.7.1 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.

### 16.7.2 WPS Operations

**Note:** For the official WPS specification, please go to <http://www.opengeospatial.org/standards/wps>.

WPS defines three main operations for the publishing of geospatial processes. These operations are modeled on similar operations in WFS and WMS. They are named:

- GetCapabilities
- DescribeProcess
- Execute

## Service Capabilities

WCS  
[1.0.0](#)  
[1.1.1](#)  
 WFS  
[1.0.0](#)  
[1.1.0](#)  
 WMS  
[1.1.1](#)  
[1.3.0](#)  
 WPS  
[1.0.0](#)

---

Figure 16.1: A link for the WPS capabilities document will display if installed properly

### GetCapabilities

The **GetCapabilities** operation requests the WPS server to provide details of service offerings. This information includes server metadata and metadata describing all processes implemented. The response from the service is an XML document called the **capabilities document**.

To make a GetCapabilities request, use the following URL:

```
http://localhost:8080/geoserver/ows?
  service=WPS&
  version=1.0.0&
  request=GetCapabilities
```

This URL assumes that GeoServer is located at `http://localhost:8080/geoserver/`.

The required parameters, as in all GetCapabilities requests, are **service** (`service=WPS`), **version** (`version=1.0.0`), and **request** (`request=GetCapabilities`).

### DescribeProcess

The **DescribeProcess** operation makes a request to the WPS server for a full description of a process known to the WPS.

An example GET request (again, assuming a GeoServer at `http://localhost:8080/geoserver/`) using the process `JTS:buffer`, would look like this:

```
http://localhost:8080/geoserver/ows?
  service=WPS&
  version=1.0.0&
```

```
request=DescribeProcess&
identifier=JTS:buffer
```

Here, the important parameter here is the `identifier=JTS:buffer`, as this defines what process to describe. Multiple processes can be requested, separated by commas (for example, `identifier=JTS:buffer,gs:Clip`), but at least one process must be specified.

**Warning:** 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 good to be aware of the specification.

The response to this request contains the following information:

<b>Title</b>	"Buffers a geometry using a certain distance"
<b>Inputs</b>	<b>distance:</b> "The distance (same unit of measure as the geometry)" ( <i>double, mandatory</i> ) <b>quadrant segments:</b> "Number of quadrant segments. Use > 0 for round joins, 0 for flat joins, < 0 for mitred joins" ( <i>integer, optional</i> ) <b>capstyle:</b> "The buffer cap style, round, flat, square" ( <i>selection, optional</i> )
<b>Output formats</b>	One of GML 3.1.1, GML 2.1.2, or WKT

**Note:** The specific processes available in GeoServer are subject to change.

## Execute

The **Execute** operation makes a request to the WPS server to perform the actual process.

The inputs required for this request depend on the process being executed. For more information about WPS processes in GeoServer, please see the section on [WPS Processes](#).

This operation is cumbersome to view as a GET request, so below is an example of a POST request. The specific process takes as an input a point at the origin (described in WKT as `POINT(0 0)`) and runs a buffer operation (`JTS:buffer`) of 10 units with single quadrant segments and a flat style, and outputs 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" xmlns:wps="http://www.opengis.net/wps/2007" xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:gml="http://www.opengis.net/gml/3.1" xmlns:geos="http://www.geogebra.org/m">
  <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>DataInputs>
  <wps:OutputFormats>
    <ows:Identifier>gml311</ows:Identifier>
  </wps:OutputFormats>
</wps:Execute>
```



```

<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 response from such a request would be (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 [WPS Request Builder](#).

### 16.7.3 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.

To see WPS requests in action, you can use the built-in [WPS Request Builder](#).

## 16.7.4 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.

## GeoServer Demos

Collection of GeoServer demo applications

- [Demo requests](#) Example requests for GeoServer (using the TestServlet).
- [SRS List](#) List of all SRS known to GeoServer
- [WCS request builder](#) Step by step WCS GetCoverage request builder
- [WPS request builder](#) Step by step WPS request builder

Figure 16.2: 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.

## WPS request builder

Step by step WPS request builder.

### Choose process

**Execute process**

**Generate XML from process inputs/outputs**

Figure 16.3: *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.

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

## WPS request builder

Step by step WPS request builder.

### Choose process

---

gs:Bounds

Computes the overlall bounds of the input features ([WPS DescribeProcess](#))

### Process inputs

---

#### features\* - FeatureCollection

The feature collection whose bounds will be computed

VECTOR\_LAYER    topp:states

### Process outputs

---

#### bounds\* - ReferencedEnvelope

The feature collection bounds

☒ Generate

Execute process

Generate XML from process inputs/outputs

Figure 16.4: WPS request builder form to determine the bounds of *topp:states*

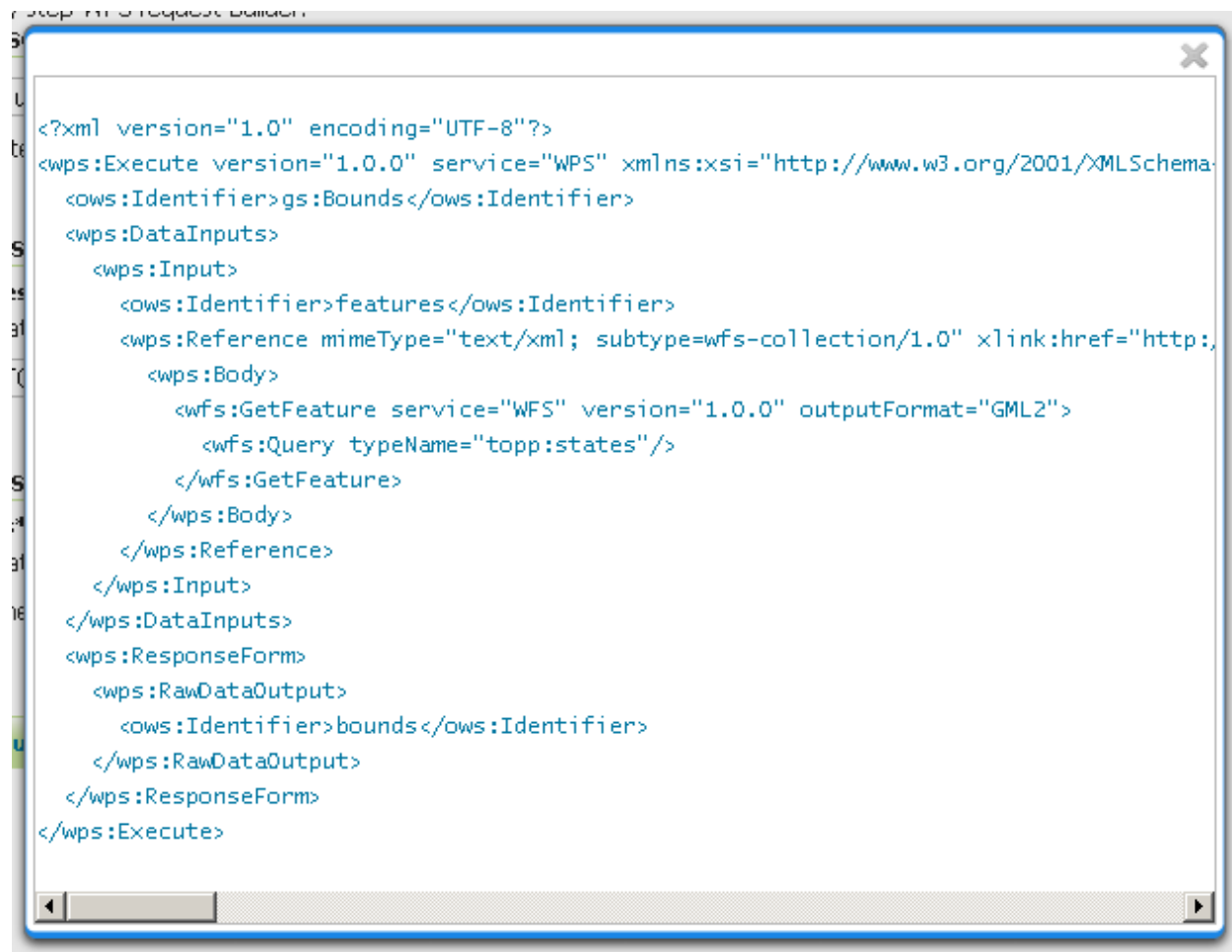


Figure 16.5: Raw WPS POST request for the above process

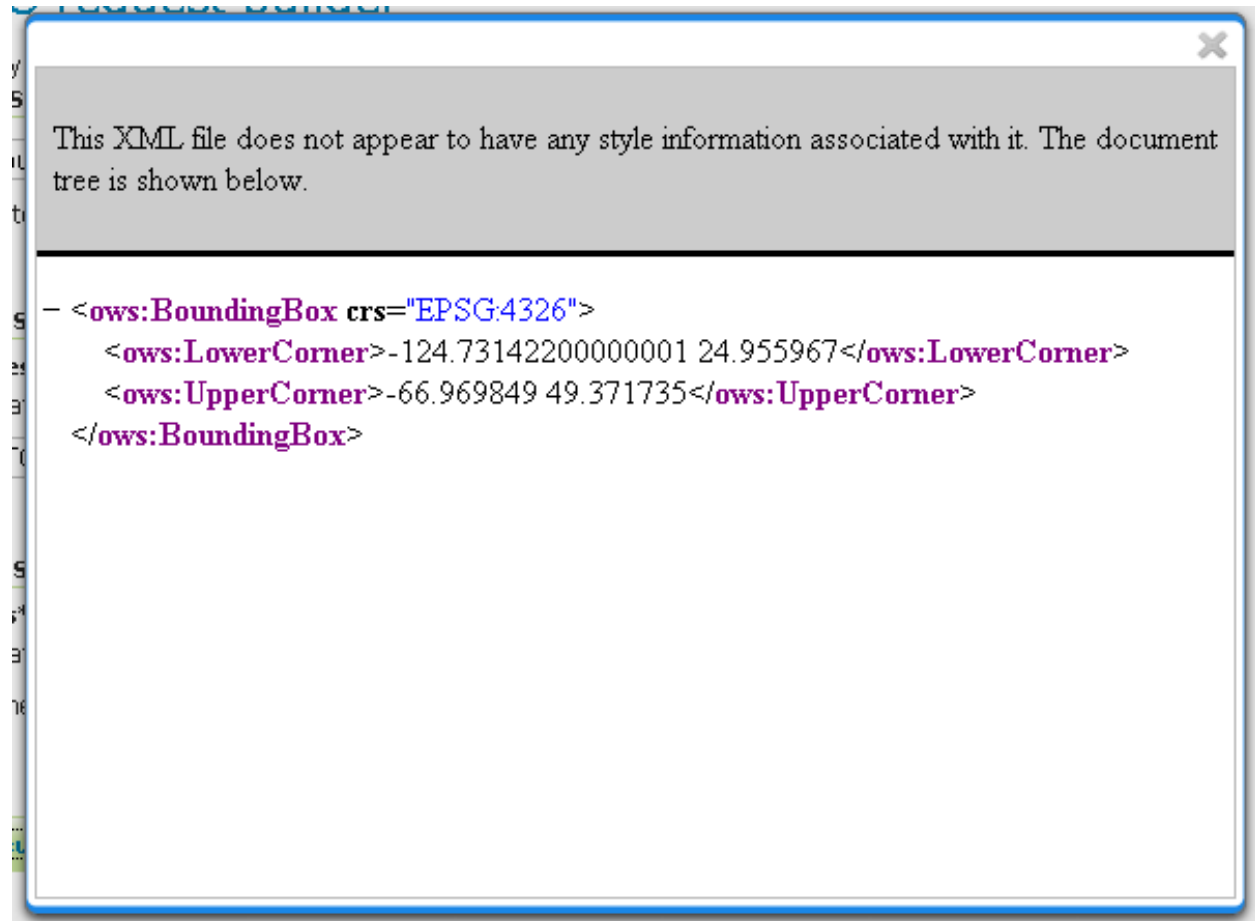


Figure 16.6: WPS server response

---

# Tutorials

---

## 17.1 Freemarker Templates

### 17.1.1 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 you for various level of customization. Given a templated output, a template name (`template.ftl`) and a feature type (`myFeatureType`), GeoServer will perform the following lookups:

- Look into `GEOSERVER_DATA_DIR/workspaces/<workspace>/<datastore>/myfeatureType/template.ftl` to see if there is a type specific template
- Look into `GEOSERVER_DATA_DIR/templates/<workspace>/template.ftl` to see if there is a workspace-specific template
- Look into `GEOSERVER_DATA_DIR/templates/template.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
  - 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
  - type (string): attribute type, the fully qualified Java class name
  - isGeometry (boolean): true if the attribute is geometric, false otherwise
  - value: the attribute value (as a string)
- type (map)
  - name (string): the type name (same as typeName)
  - 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)

## 17.2 GeoRSS

GeoServer supports [GeoRSS](#) as an output format allowing you to serve features as an RSS feed.

### 17.2.1 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=topp: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:

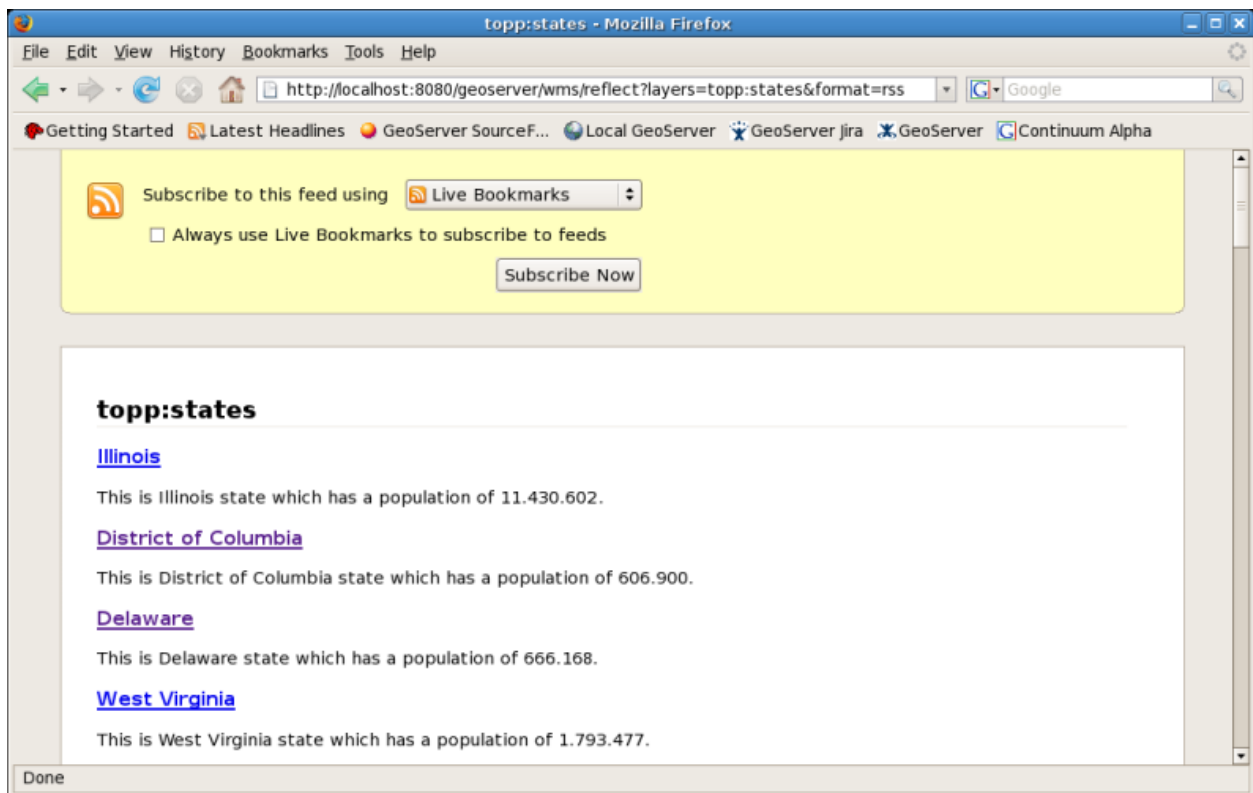


Figure 17.1: *topp:states* rss feed

### 17.2.2 Ajax Map Mashups

**Note:** Internet visible geoServer required for Ajax Mashups. Your GeoServer instance must be visible from the internet, IE; localhost will not work.

### 17.2.3 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/xhtml1-strict.dtd">
<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</title>
    <script src="http://maps.google.com/maps?file=api&v=2.x&key=<INSERT MAPS API KEY HERE>"></script>

    <script type="text/javascript">
      //
        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("&lt;INSERT GEOSERVER URL HERE&gt;/geoserver/wms/reflect?layer=...");
            map.addOverlay(geoXml);
          }
        }
      //]]&gt;
    &lt;/script&gt;

  &lt;/head&gt;
  &lt;body onload="load()" onunload="GUnload()"&gt;
    &lt;div id="map" style="width: 800px; height: 600px"&gt;&lt;/div&gt;
  &lt;/body&gt;
&lt;/html&gt;</pre></div><div data-bbox="133 517 450 533" data-label="List-Group"><ol><li>3. Visit <code>gmaps.html</code> in your web browser.</li></ol></div><div data-bbox="112 539 889 571" data-label="Text"><p><b>Note:</b> 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</p></div><div data-bbox="112 595 297 613" data-label="Section-Header"><h2>17.2.4 Yahoo Maps</h2></div><div data-bbox="112 627 740 644" data-label="Text"><p>How to create a Yahoo! Maps mashup with a GeoRSS overlay produced by GeoServer.</p></div><div data-bbox="133 650 938 704" data-label="List-Group"><ol><li>1. Obtain a &lt;Yahoo Maps Application ID <a href="http://search.yahooapis.com/webservices/register_application">http://search.yahooapis.com/webservices/register_application</a>&gt; from Yahoo.</li><li>2. Create an html file called <code>ymaps.html</code>:</li></ol></div><div data-bbox="154 721 1000 900" data-label="Text"><pre>&lt;html&gt;
  &lt;head&gt;
    &lt;title&gt;Yahoo! Maps GeoRSS Overlay Example&lt;/title&gt;
    &lt;script src="http://api.maps.yahoo.com/ajaxymap?v=3.0&amp;appid=&lt;INSERT APPLICATION ID HERE&gt;" type="text/javascript"&gt;&lt;/script&gt;

    function StartYMap() {
      var map = new YMap(document.getElementById('ymap'));
      map.addPanControl();
      map.addZoomShort();

      function doStart(eventObj) {
        var defaultEventObject = eventObj;
      }
    }
  &lt;/head&gt;
  &lt;body&gt;
    &lt;div id="ymap"&gt;&lt;/div&gt;
  &lt;/body&gt;
&lt;/html&gt;</pre></div><div data-bbox="112 931 147 948" data-label="Page-Footer"><hr/>528</div><div data-bbox="707 931 889 949" data-label="Page-Footer">Chapter 17. Tutorials</div>
```

```

        //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

### 17.2.5 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:

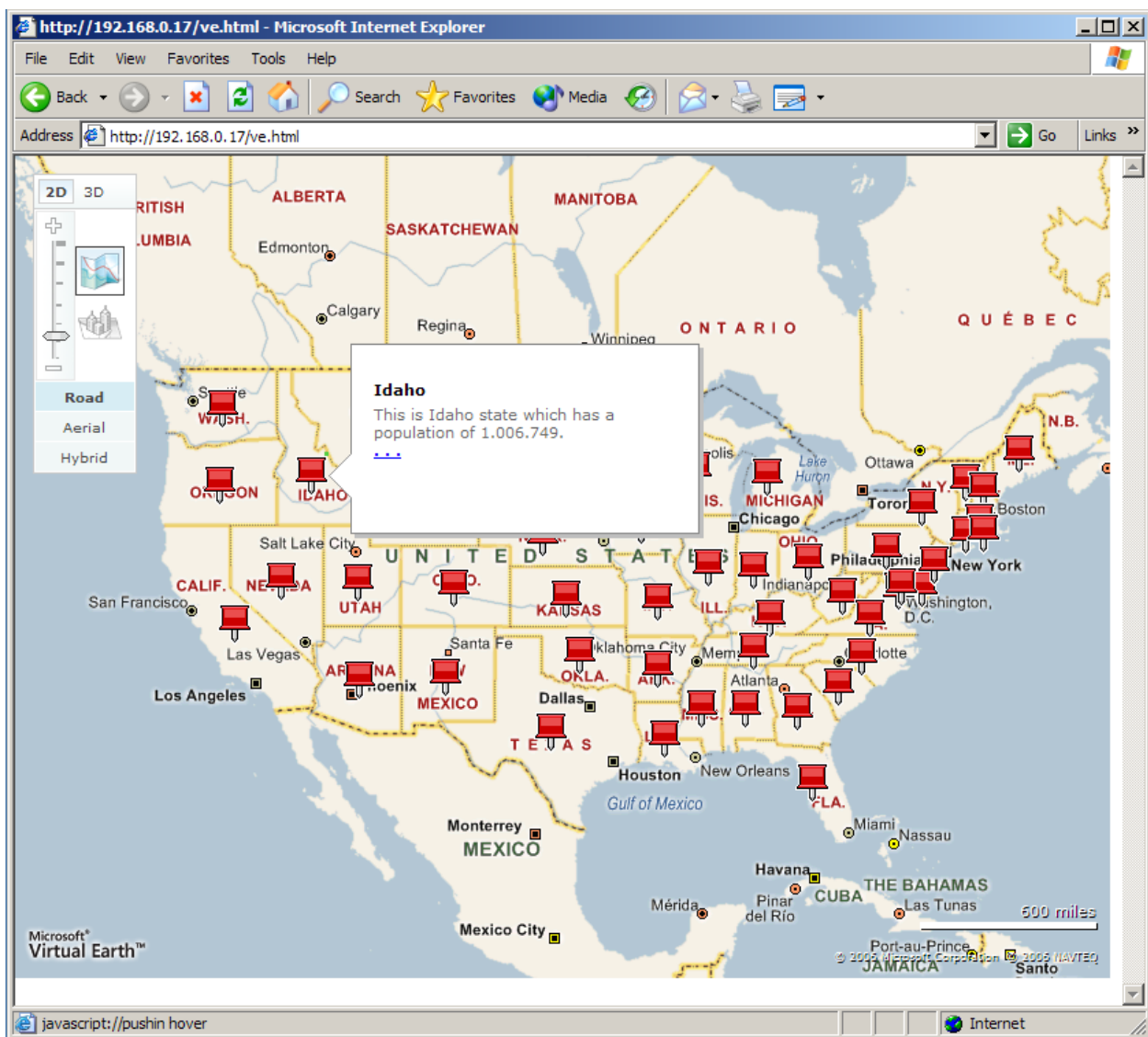


Figure 17.2: *Virtual Earth*

## 17.3 GetFeatureInfo Templates

This tutorial describes how to use the GeoServer template system to create custom HTML GetFeatureInfo responses.

### 17.3.1 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).

### 17.3.2 Standard Templates

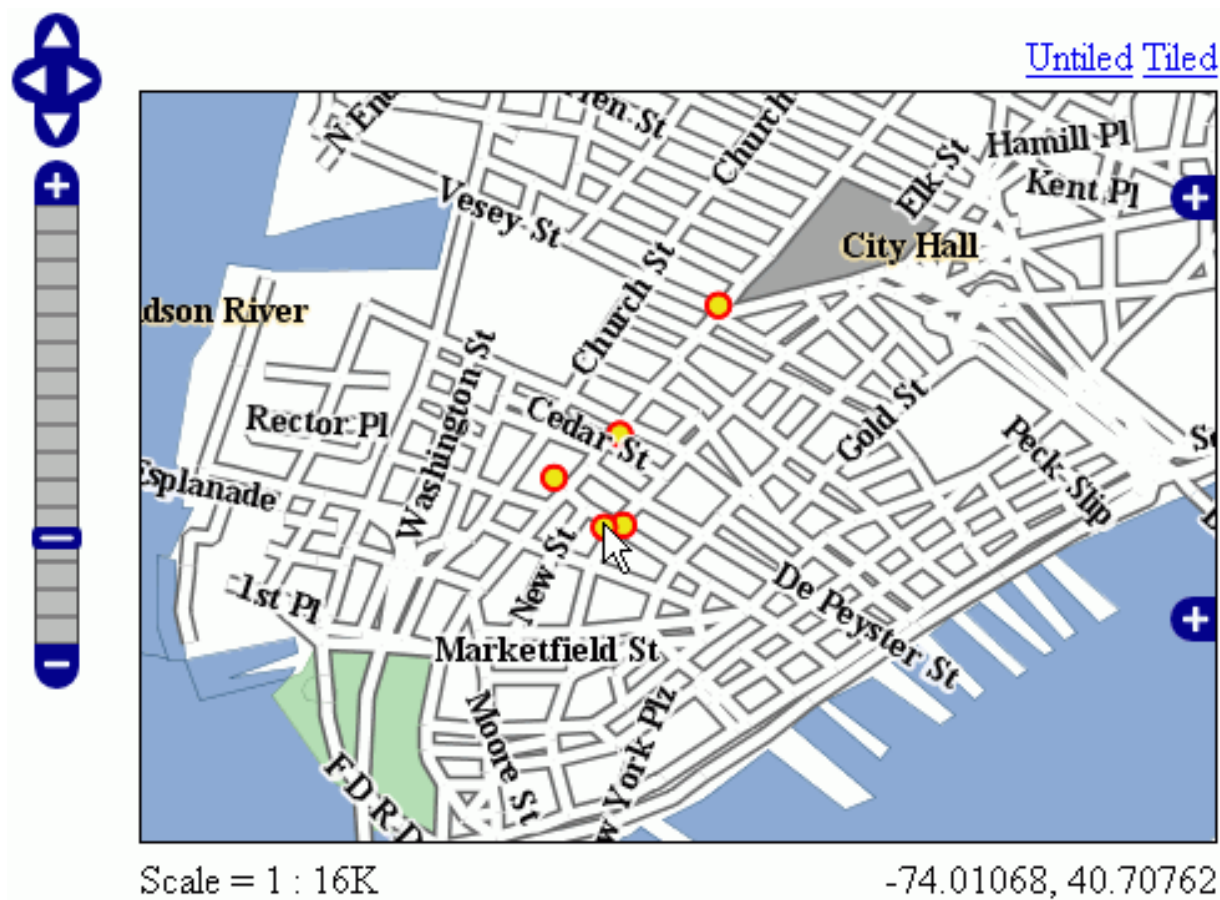
The following assumes you're already up to speed with Freemarker templates. If you're not, read the [Freemarker Templates](#) tutorial, and the [KML Placemark Templates](#) page, which has simple examples.

The default output is generated by the standard templates, which are three:

- header.ftl
- content.ftl
- footer.ftl

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;
```

**TIGER\_ROADS**

CFCC	NAME
A41	Wall St

**POI**

NAME	THUMBNAIL	MAINPAGE
stock	pics/22037829-Ti.jpg	pics/22037829-L.jpg

Figure 17.3: Default Output

```

    }
    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.

### 17.3.3 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>
```

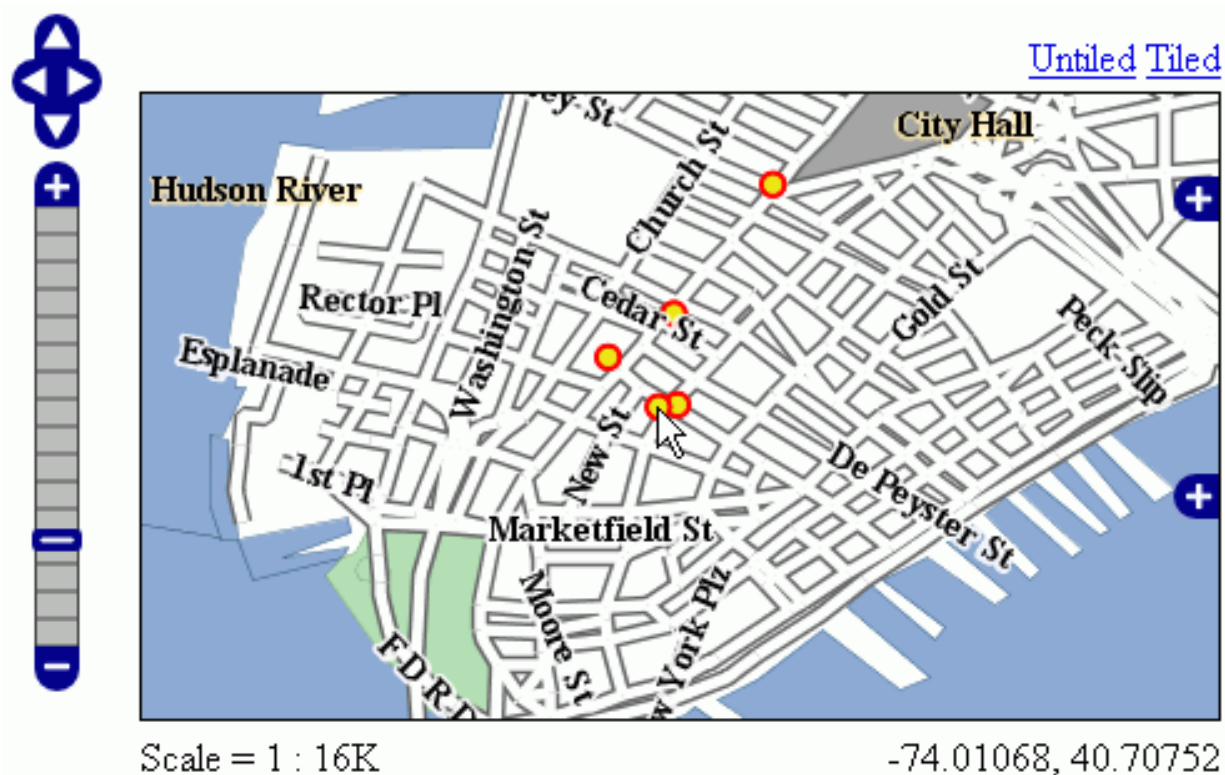
With this template in place, the output would be:

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/>
    
  </li>
</#list>
</ul>
```





**Type: tiger\_roads** (id: *tiger\_roads.6063*):

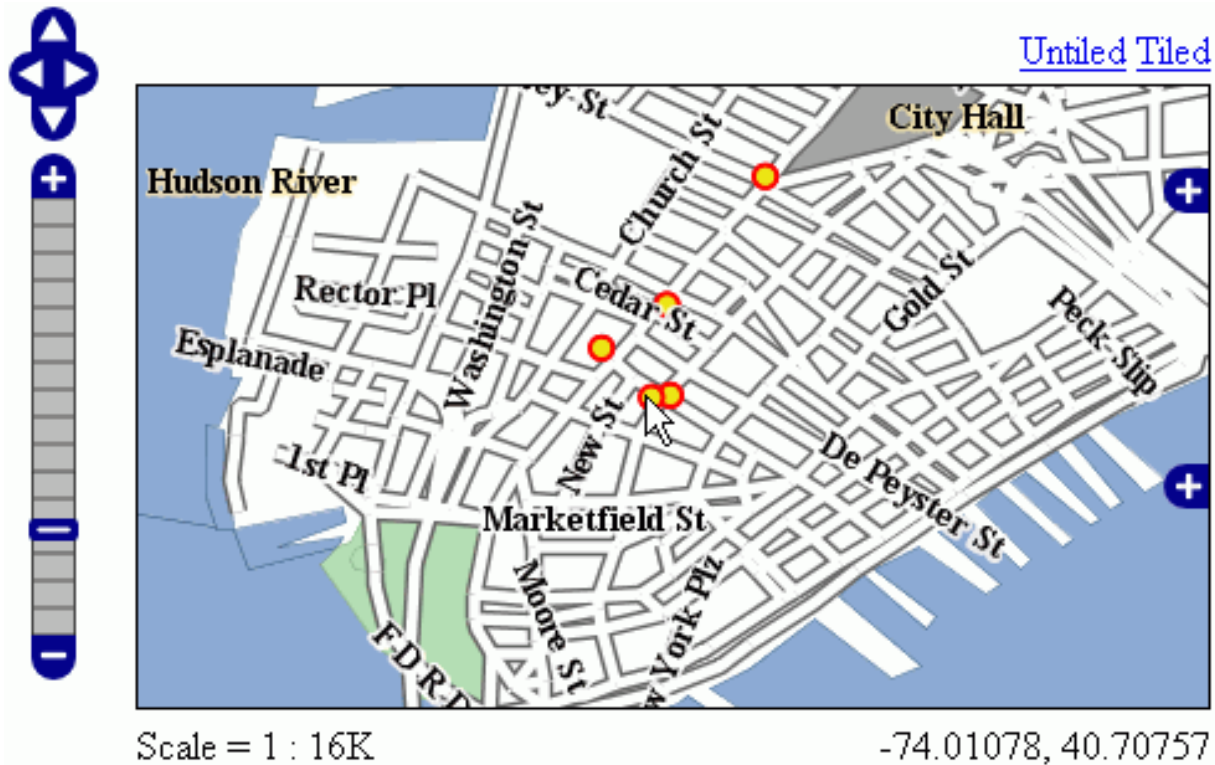
- ◊ CFCC: A41
- ◊ NAME: Broad St

**Type: poi** (id: *poi.2*):

- ◊ NAME: stock
- ◊ THUMBNAIL: pics/22037829-Ti.jpg
- ◊ MAINPAGE: pics/22037829-L.jpg

Figure 17.4: Bulleted List Output

With this additional template, the output is:



- ◆ **Type: tiger\_roads** (id: *tiger\_roads.6066*):
  - CFCC: A41
  - NAME: Wall St
- ◆ **Point of interest, "stock":**



Figure 17.5: Output with Thumbnail Image

As you can see, roads are still using the generic template, whilst POI is using its own custom template.

### 17.3.4 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}`.

## 17.4 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.

### 17.4.1 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.

### 17.4.2 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 17.6: *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 `$GEOSERVER_DATA_DIR/palettes` directory. The sample file can be either in GIF or PNG format. If the file is named `mypalette.gif` or `mypalette.png`, the user will be able to refer it appending `palette=mypalette` to the GetMap request. GeoServer will load the palette from the file and use it.
3. Provide a palette file. The process is just as before, but this time only the palette, in `.PAL` format (Microsoft Palette Format, can be generated both by Paint Shop Pro and IrfanView), will be stored into `$GEOSERVER_DATA_DIR/palettes`.

### 17.4.3 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

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



Figure 17.7: The standard PNG full color output





Figure 17.8: The PNG8 output



Figure 17.9: PNG + internet safe palette



Figure 17.10: PNG + 'custom palette <<http://geoserver.org/download/attachments/1278244/nyp.pal?version=1>>' \_



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,  $GT + \text{sizeInKbytes} * 8 / \text{speedInKbits}$ .

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.

#### 17.4.4 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

#### 17.4.5 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.

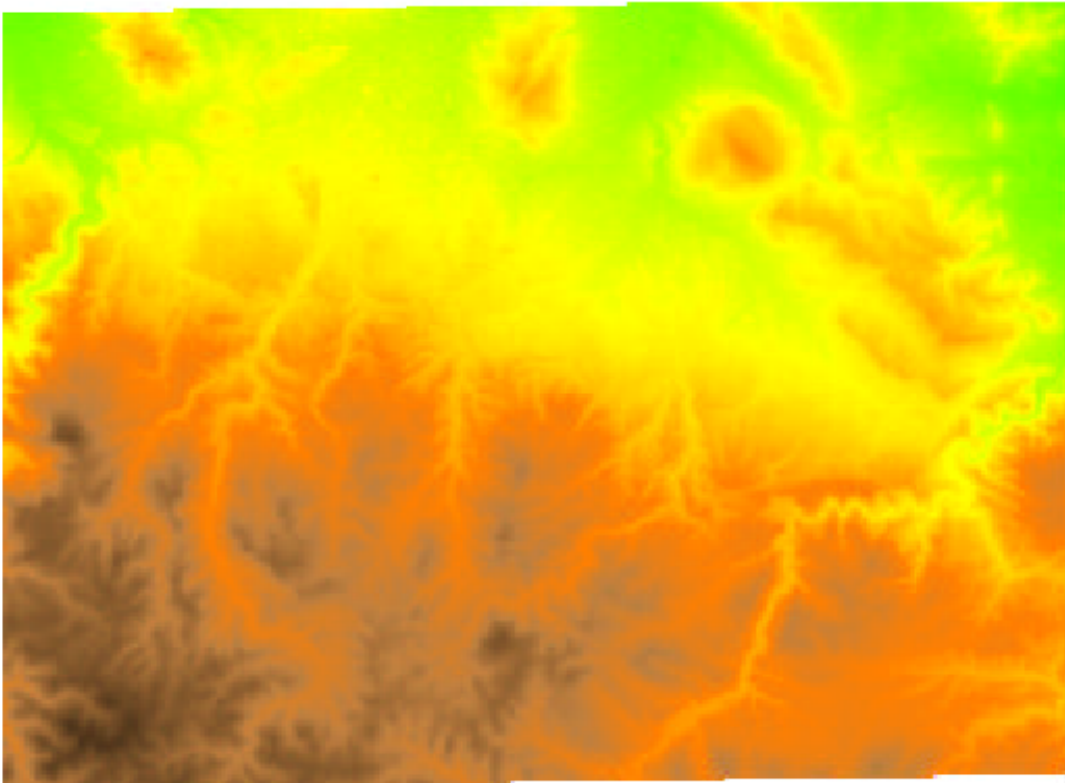


Figure 17.11: *The standard PNG full color output.*

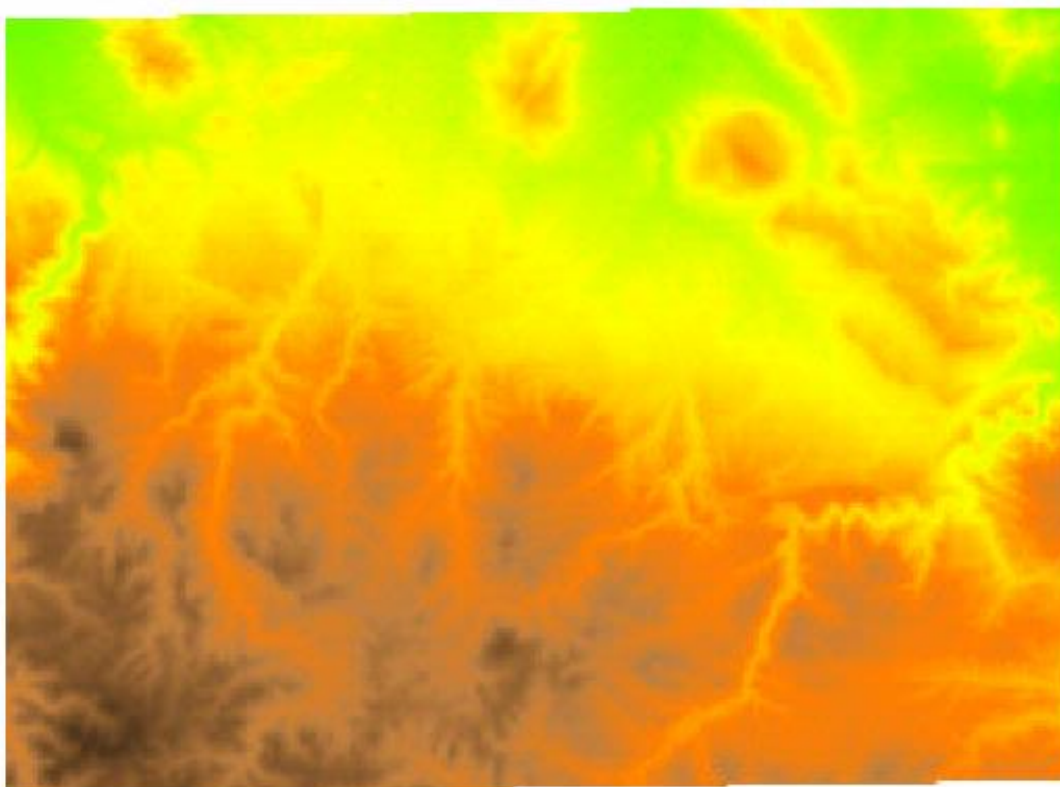


Figure 17.12: *JPEG output*

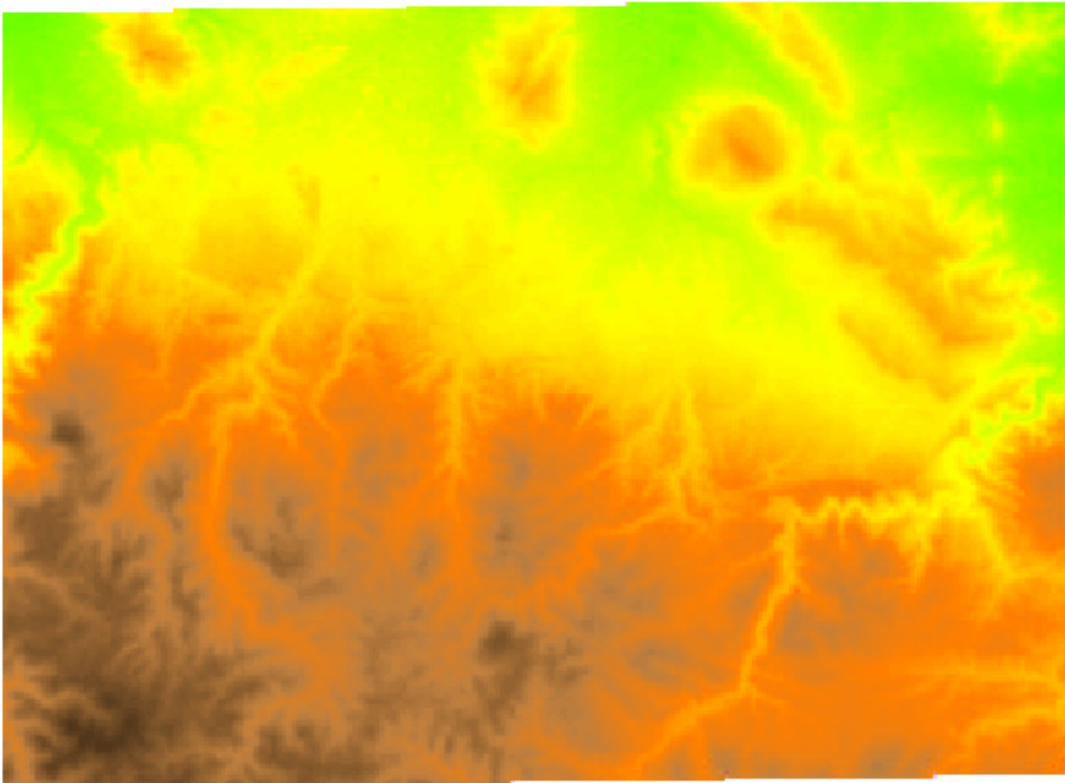


Figure 17.13: *The PNG8 output.*

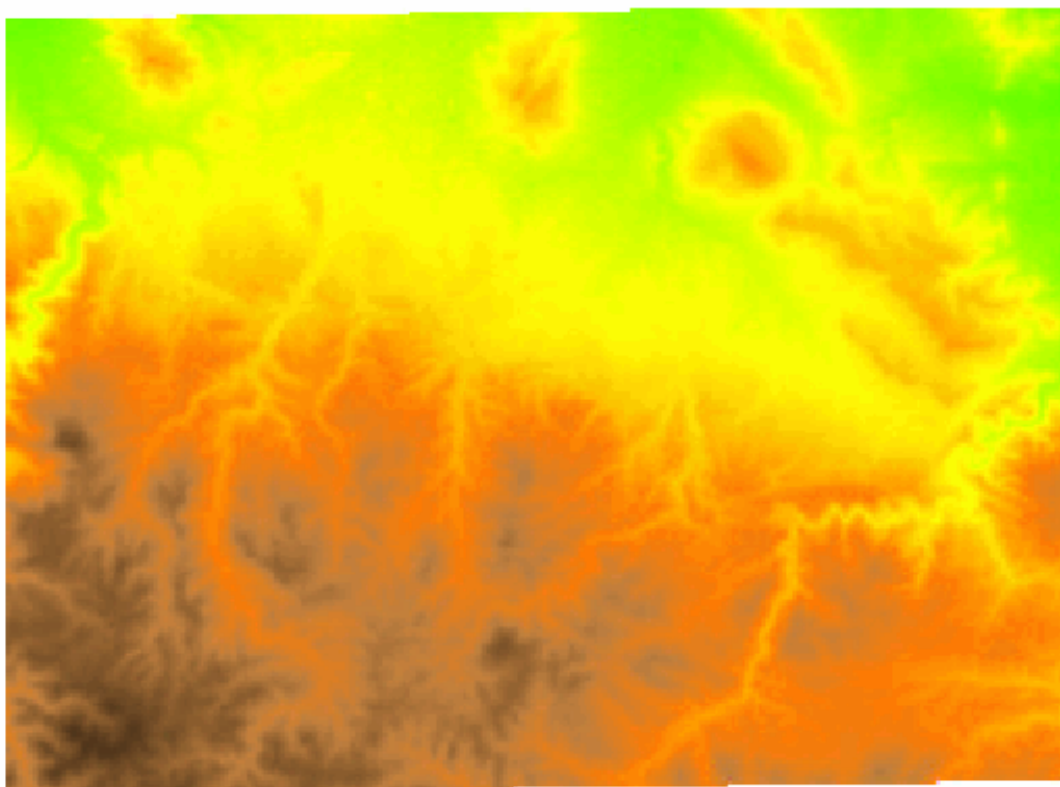


Figure 17.14: *PNG + custom palette (using the png8 output as the palette).*

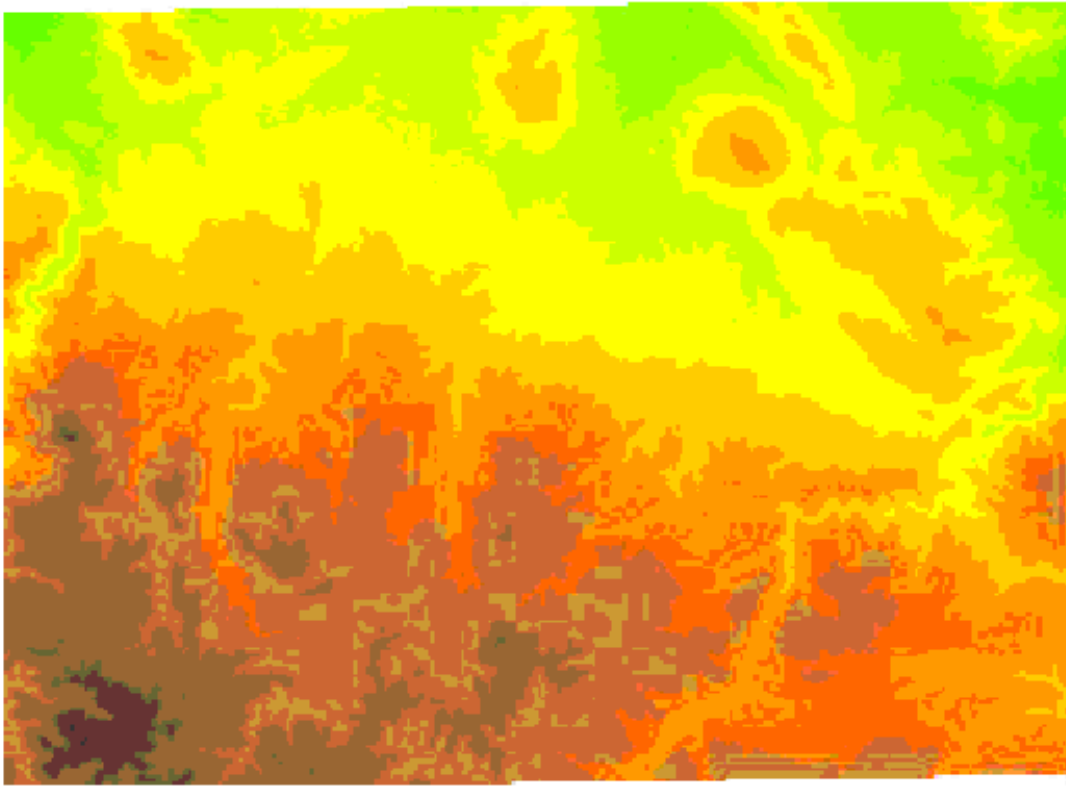


Figure 17.15: *PNG + internet safe palette.*

## 17.5 Serving Static Files

### 17.5.1 Introduction

Let's say you've just setup your data and styles in Geoserver, and you've created a nice front end with a pure javascript library like OpenLayers or MapBuilder. You're ready to tell the world about your new shiny app, there is only a catch... where do you put your static files (html files, a few icons, some javascript) so that they are served on the web?

So far you did not have a quick solution, and had to use one of the following approaches:

1. Roll your own extra web app to be deployed along with Geoserver (in the same container). This requires some java webapp setup knowledge.
2. Unpack geoserver, modify the webapp contents, repack it (ugly, making Geoserver upgrades inconvenient)
3. Use separate web server (Apache, IIS) to serve the pages (which requires some knowledge on its own).

If your application needed to make ajax calls back to Geoserver (WFS-T requires that) you would stumble into another roadblock: ajax calls are sandboxed so that you can call back only the same server that provided the page making the call. This meant that option #3 was out of the question, and an approach using some proxying (mod\_proxy or similar) was required.

### 17.5.2 Directly from the Data Directory

With GeoServer you can put your own static files in the `www` subfolder of the data directory, and have them served at `http://myhost:8080/geoserver/www`. This means you can put in your html, images and javascript (even a full installation of MapBuilder) and have Geoserver provide them on the web: no need for unpacking, creating a new webapp, or fiddling with another web server, and no problems with ajax callback.

Now, this is handy, but has its own limitations:

- we cannot serve files whose MIME type does not get recognized (if you get an HTTP 415 error, this is because we cannot spot your file MIME type);
- the solution is pure java and does not make use of eventual accelerators such as the [Tomcat APR library](#), this means if you have tons of static files to be served at high speed, you probably want to switch back to solution #1 or #3 to get optimal performance.

## 17.6 WMS Reflector

### 17.6.1 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=800`

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.

## 17.6.2 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:restricted
```

### 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 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 uses 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:restricted&bbox=-103.929,44.375,-103.633,44.500
```

## 17.6.3 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.

```

```

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:

```

```

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" >

</a>
```

(The a-tags are on separate lines for clarity, they will in fact result in a space in front and after the image).

### 17.6.4 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:states" >
</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.

## 17.7 CQL and ECQL

CQL (OGC Common Query Language) is a query language created by OGC for the [Catalogue WebServices specification](#). Unlike the OGC Filter specification, CQL is plain text, human readable, and thus well suited for manual construction as opposed to machine generation. However CQL has some serious limitations, for example it cannot encode id filters and requires an attribute to be on the left side of any comparison operator. ECQL removes such limitations making for 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 dynamic symbolizers. When the documentation refers to CQL you can rest assured ECQL syntax can be used as well (and if not, please report that as a bug).

This tutorial introduces the language by example. If you need a full reference instead have a look at the [ECQL BNF definition](#) on the GeoTools site.

### 17.7.1 Getting started

All the following examples are going to use the `topp:states` sample layer shipped with GeoServer, and will use the `CQL_FILTER` vendor parameter to show how the CQL filters alter the map appearance. The easiest way to follow the tutorial is to open your GeoServer map preview, click on the *options* button at the top of the map preview, in order to open the advanced options toolbar, and enter the filter in the CQL box.

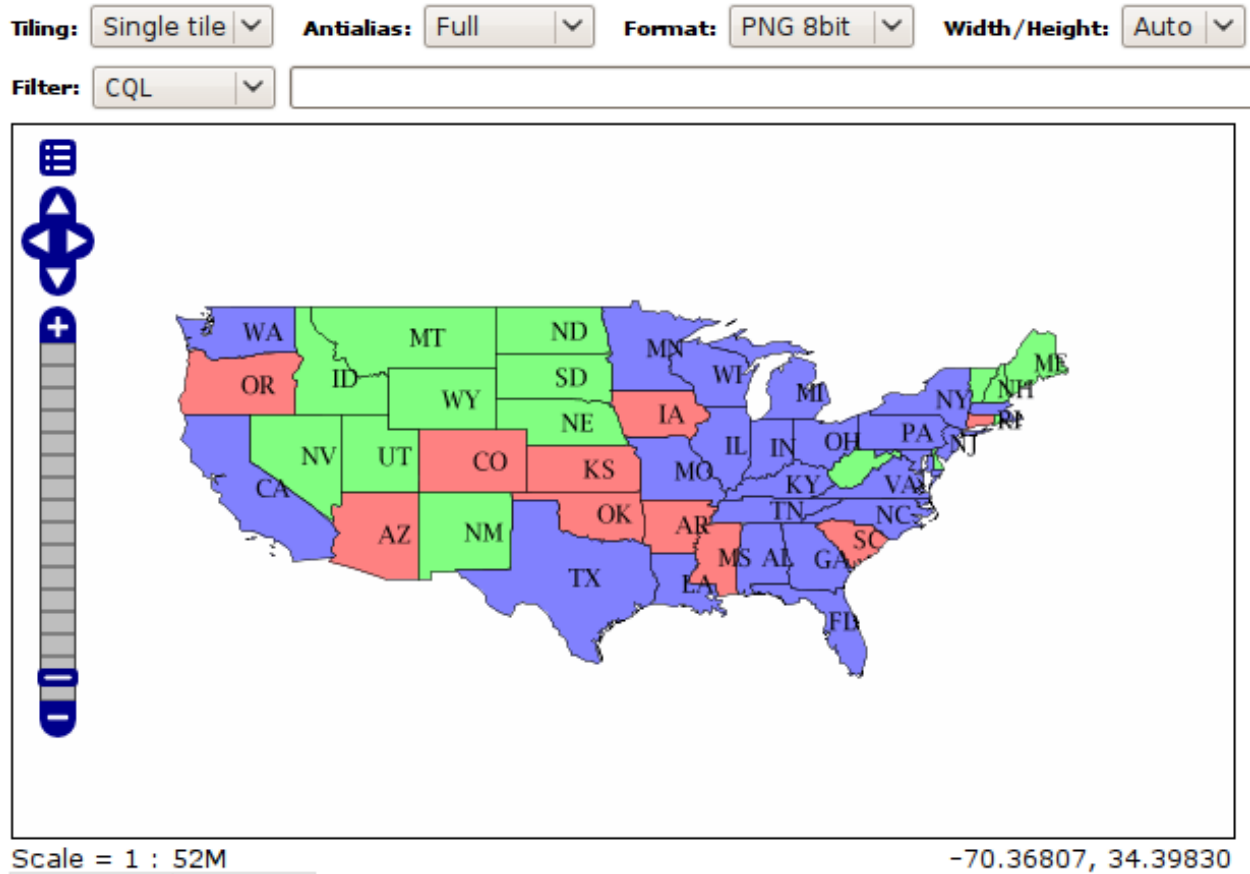


Figure 17.16: *topp:states* preview with advanced toolbar open.

The attributes we'll be using in the filters are those included in the layer itself. This is an example of attribute names and values for the state of Colorado:

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

### 17.7.2 Simple comparisons

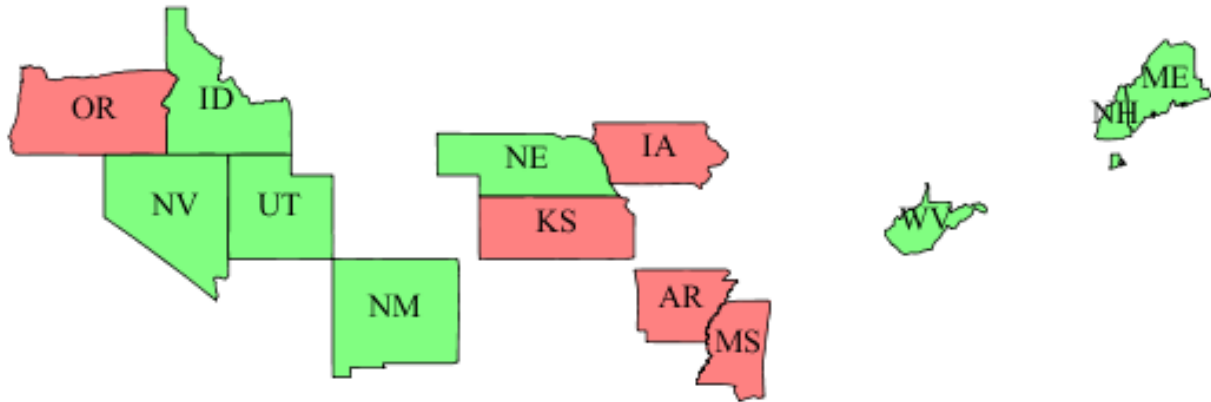
Let's get started with the simplest example. In CQL basic arithmetic and comparisons do look exactly like plain text. The filter `PERSONS > 15000000` will extract only states that do have more than 15 million inhabitants:



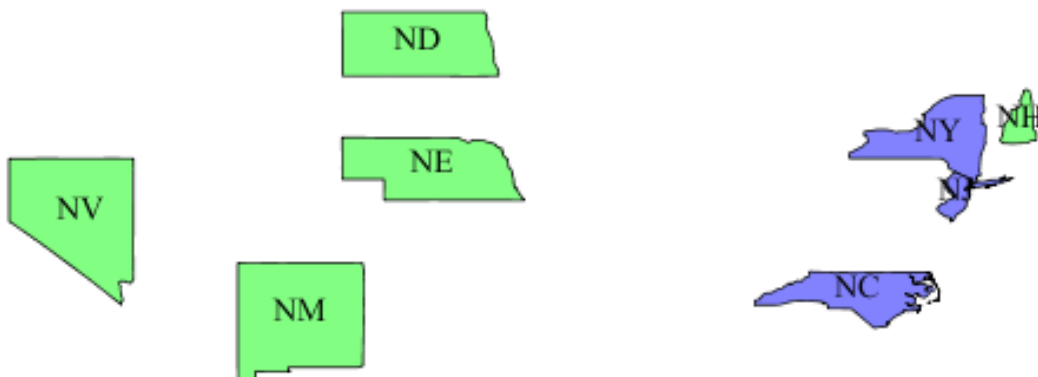
Figure 17.17: *PERSONS > 15000000*

To check a range of values a between filter can be used instead: `PERSONS BETWEEN 1000000 AND 3000000`:

Comparing with text is similar. In order to get only the state of California, the filter will be `STATE_NAME`

Figure 17.18: *PERSONS BETWEEN 1000000 AND 3000000*

= 'California'. More complex text comparisons are available using `LIKE` comparisons. `STATE_NAME LIKE 'N%'` will extract all states starting with an N.

Figure 17.19: *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):

It is also possible to make simple math expressions using the `+`, `-`, `*`, `/` operators. The following filter `UNEMPLOY / (EMPLOYED + UNEMPLOY) > 0.07` selects all states whose unemployment ratio is above 7% (remember the sample data is very old, don't draw any conclusion from the results)

### 17.7.3 Id and list comparisons

If we want to extract only the states with a certain feature id we'll use the `IN` filter without specifying any attribute, as in `IN ('states.1', 'states.12')`:



Figure 17.20: *MALE > FEMALE*

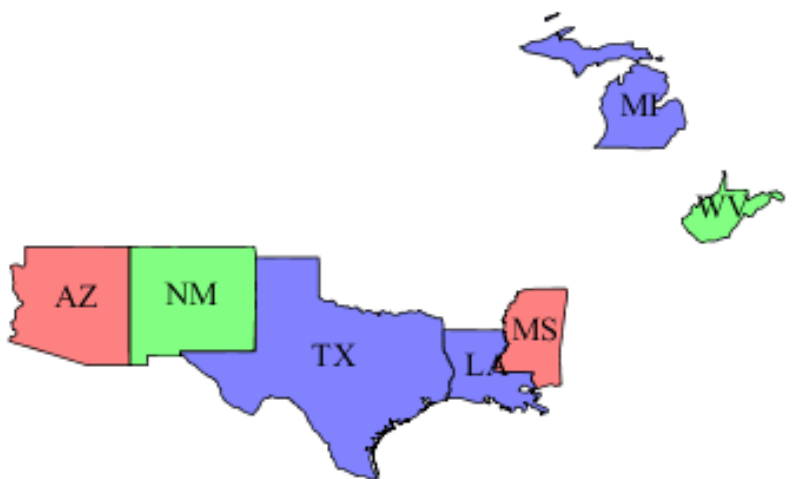


Figure 17.21:  $UNEMPLOY / (EMPLOYED + UNEMPLOY) > 0.07$



Figure 17.22: `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` filter specifying an attribute name, like in `STATE_NAME IN ('New York', 'California', 'Montana', 'Texas')`:

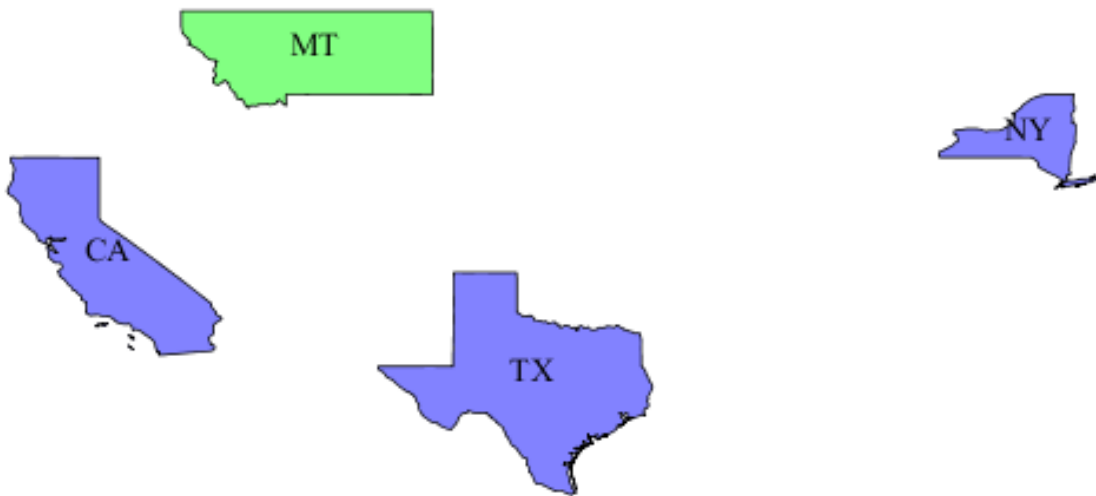


Figure 17.23: `STATE_NAME IN ('New York', 'California', 'Montana', 'Texas')`

### 17.7.4 Calling filter functions

CQL/ECQL can call any of the [filter functions](#) available in GeoServer.

For example, say we want to find all states whose name contains an “m”, regardless of whether it’s a capital one, or not. We can call the `strToLowerCase` to turn all the state names to lowercase and then use a like comparison: `strToLowerCase(STATE_NAME) like '%m%'`:



Figure 17.24: `strToLowerCase(STATE_NAME)` like `'%m%'`

### 17.7.5 Geometric filters

CQL provides a full set of geometric filter capabilities. Say, for example, you want to display only the states that do cross the `(-90,40,-60,45)` bounding box. The filter will be `BBOX(the_geom, -90, 40, -60, 45)`



Figure 17.25: `BBOX(the_geom, -90, 40, -60, 45)`

Conversely we can filter out all of the states that are overlapping that bounding box with the following filter `DISJOINT(the_geom, POLYGON((-90 40, -90 45, -60 45, -60 40, -90 40)))`:



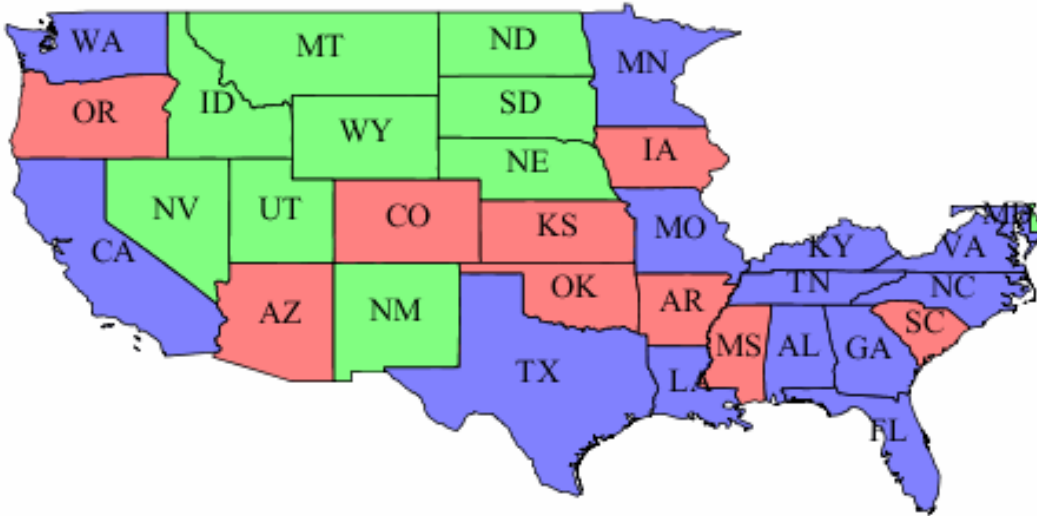


Figure 17.26: `DISJOINT(the_geom, POLYGON((-90 40, -90 45, -60 45, -60 40, -90 40)))`

## 17.8 Using the ImageMosaic plugin

### 17.8.1 Introduction

This tutorial describes the process of creating a new coverage using the new ImageMosaic plugin. The ImageMosaic plugin is authored by [Simone Giannecchini](#) of [GeoSolutions](#), 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 (see next sections)
3. All the granules must share the same spatial resolution and set of overviews.

**Note:** About point 3, in the original version of the ImageMosaic plugin this assumption was entirely true since we made an assumption to work with real tiles coming from a set of adjacent images. Lately we have been doing a substantial refactoring, so this condition could be removed, but doing so would take some more work and a few additional options in the configuration file.

To be more specific, if we can't assume that all the granules share the same spatial layout and overviews set we would not be able to assess the raster dimensions (width and height) the spatial dimensions (grid-to-world and envelope) and the overviews set to the final mosaic coverage, unless we specify them somehow or we default to something. As long as we can assume that the various granules share the same spatial elements as well as the same overviews set we can inherit the first definition for the final mosaic. This limitation can be overcome with more work.

### **17.8.2 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 are needed:

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 shapefile 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.

Later on we will describe the process of creating an index for a set of granules.

### **17.8.3 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 mosaic creation and usually do not require manual editing. The table below describes the various elements in this configuration file.

Parameter	Mandatory	Description
<i>Envelope2D</i>	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 <i>Envelope2D=432500.25,81999.75 439250.25,84999.75</i>
<i>Level-Num</i>	Y	Represents the number of reduced resolution layers that we currently have for the granules of this mosaic.
<i>Levels</i>	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 all the granules.
<i>Name</i>	Y	Represents the name for this mosaic.
<i>Expand-ToRGB</i>	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 granules do not share the same color map hence a straight composition that would retain such a color map cannot be performed.
<i>AbsolutePath</i>	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.
<i>Location-Attribute</i>	N	The name of the attribute path in the shapefile index. Default value is <i>location</i> .

#### 17.8.4 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 tree of directories starting from the provided input one.

#### 17.8.5 Configuring a Coverage in Geoserver

This is a process very similar to creating a FeatureType. More specifically, one has to perform the steps highlighted 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 17.27: ImageMosaic in the list of raster data stores

1. 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*

1. Click on the layer you wish to configure and you will be presented with the Coverage Editor:

#### *Coverage Editor*

1. 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.
2. Click on the Save button.

Once you complete the preceding operations it is possible to access the OpenLayers map preview of the created mosaic.

**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:

## Add Raster Data Source

Description

ImageMosaic

Image mosaicking plugin

### Basic Store Info

#### Workspace

cite ▼

#### Data Source Name

#### Description

☒ Enabled

### Connection Parameters

#### URL

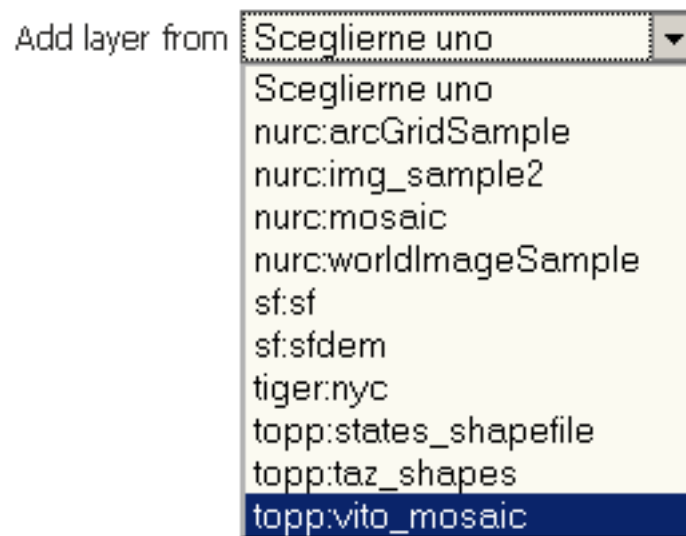
file:data/example.extension

Save

Cancel

Figure 17.28: *Configuring an ImageMosaic data store*

## New Layer chooser





#### 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 33 N...

SRS handling

Reproject native to declared

### Bounding Boxes

Native Bounding Box:

Min X	Min Y	Max X	Max Y
500.000	5,678,999	540.000	5,719,999

[Compute from data](#)

Lat/Lon Bounding Box:

Min X	Min Y	Max X	Max Y
3	51,261	3,578	51,631

[Compute from native bounds](#)

Parameter	Description
<i>MaxAllowedTiles</i>	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.
<i>Background-Values</i>	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.
<i>OutputTransparentColor</i>	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*

<i>InputTransparentColor</i>	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*

<i>AllowMultithreading</i>	If true enable tiles multithreading loading. This allows to perform parallelized loading of the granules that compose the mosaic.
<i>USE_JAI_IMAGE_READ</i>	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.
<i>SUGGESTED_TILE_SIZE</i>	Controls the tile size of the input granules as well as the tile size of the output mosaic. Consists of two positive integers separated by a comma, like 512,512.

**Note:** Deferred loading consumes less memory since it uses a streaming approach to load in memory only 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.

## 17.8.6 Configuration examples

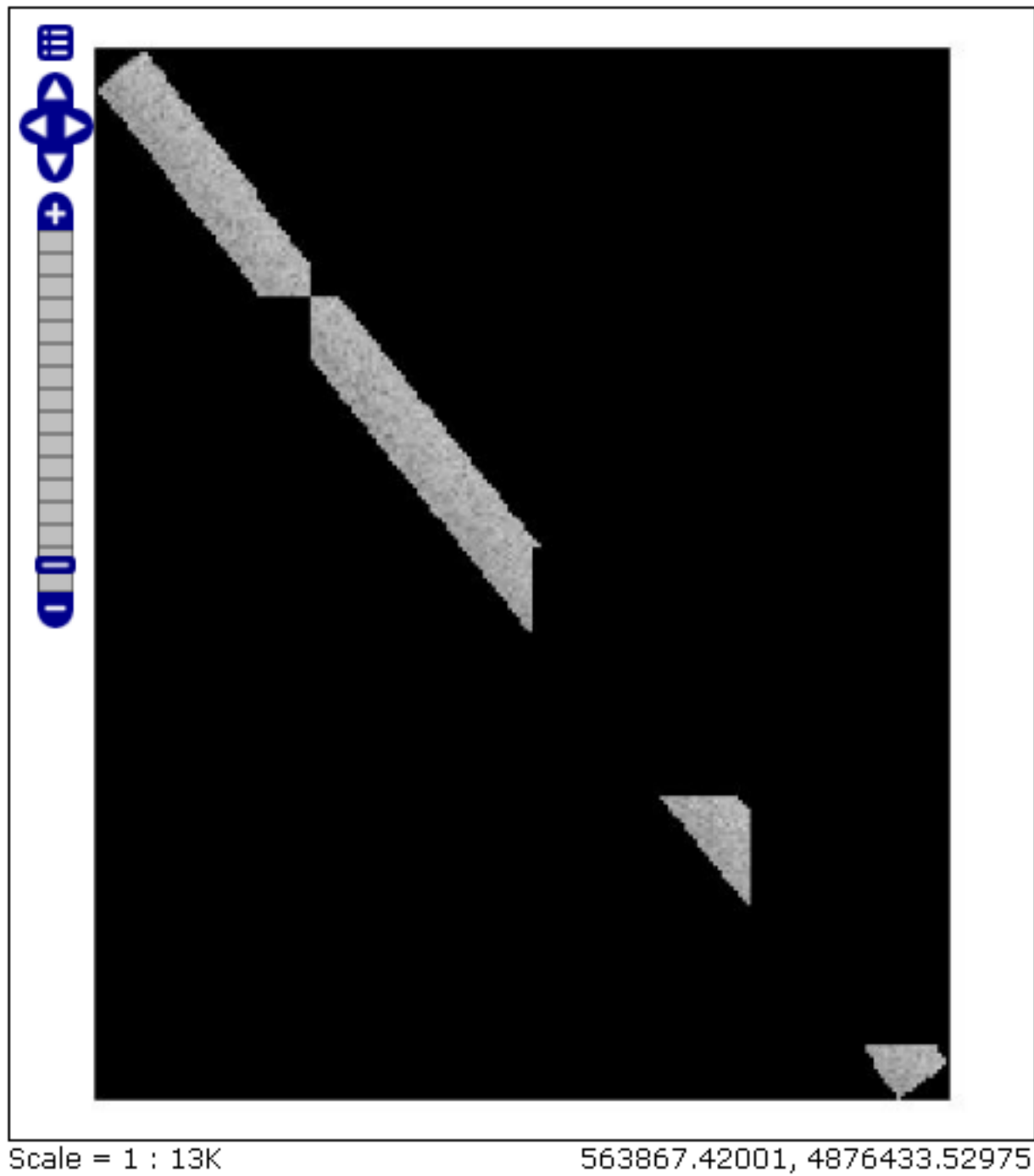
Now we are going to provide a few examples of mosaic configurations to demonstrate how we can make use of the ImageMosaic parameters.

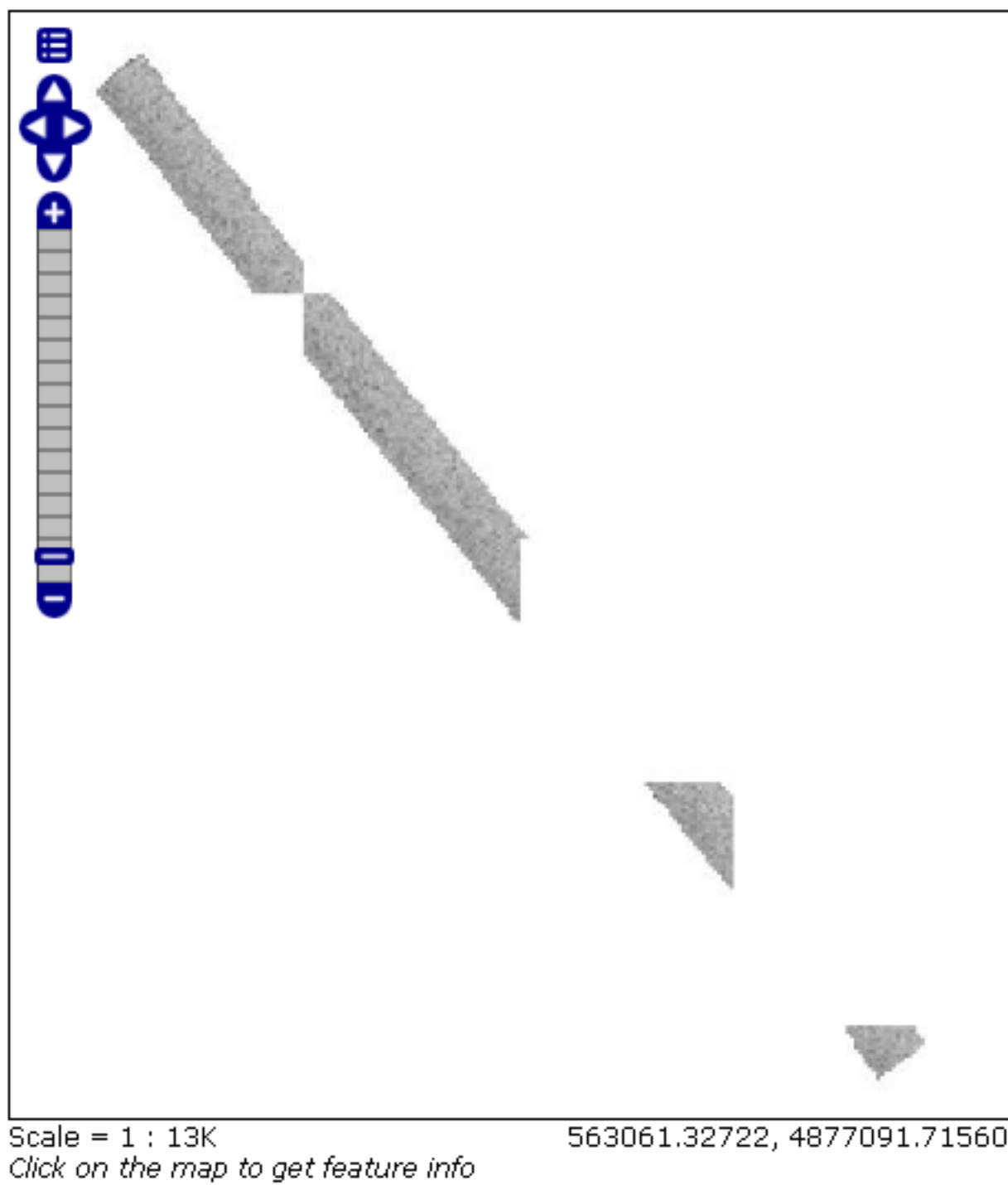
### DEM/Bathymetric mosaic configuration (raw data)

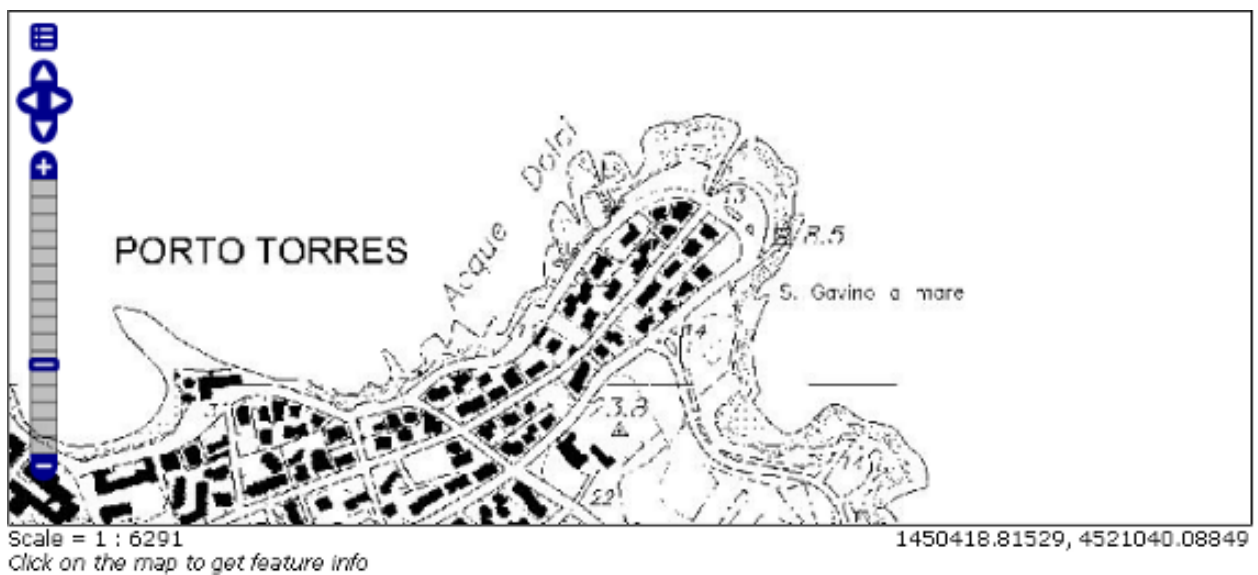
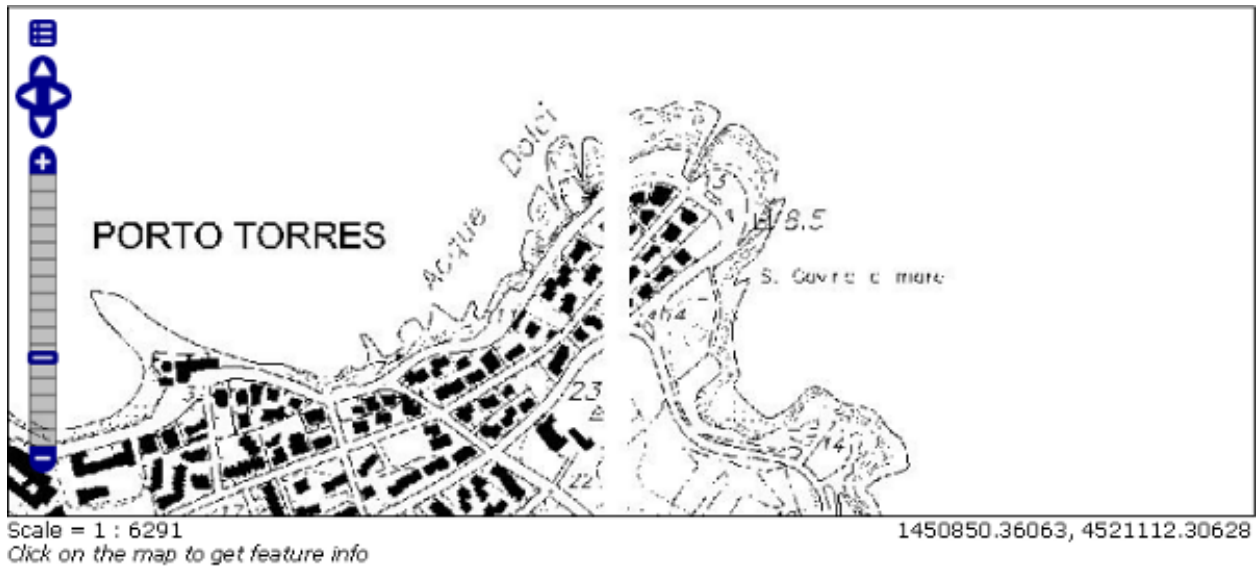
Such a mosaic can be use to serve large amount of data which represents altitude or depth and therefore does not specify colors directly while it reather needs an SLD to generate pictures. In our case we have a DEM dataset which consists of a set of raw geotiff files.

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/StyledLayerDescriptor.xsd">
  <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:
1. Allow multithreaded granules loading. By setting the 'AllowMultiThreading' parameter to true GeoServer will load the granules in parallel using 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'.

### Default Title

#### Default Style

#### Additional Styles

Available Styles		Selected Styles
<div>burg capitals cite_lakes concat dem flags giant_polygon grass green line</div>	<div>➔ ➞</div>	<div></div>

#### Default WMS Path

### WMS Attribution

#### Attribution Text

#### Attribution Link

#### Logo URL

#### Logo Content Type

#### Logo Image Width

#### Logo Image Height

[Auto-detect image size and type](#)

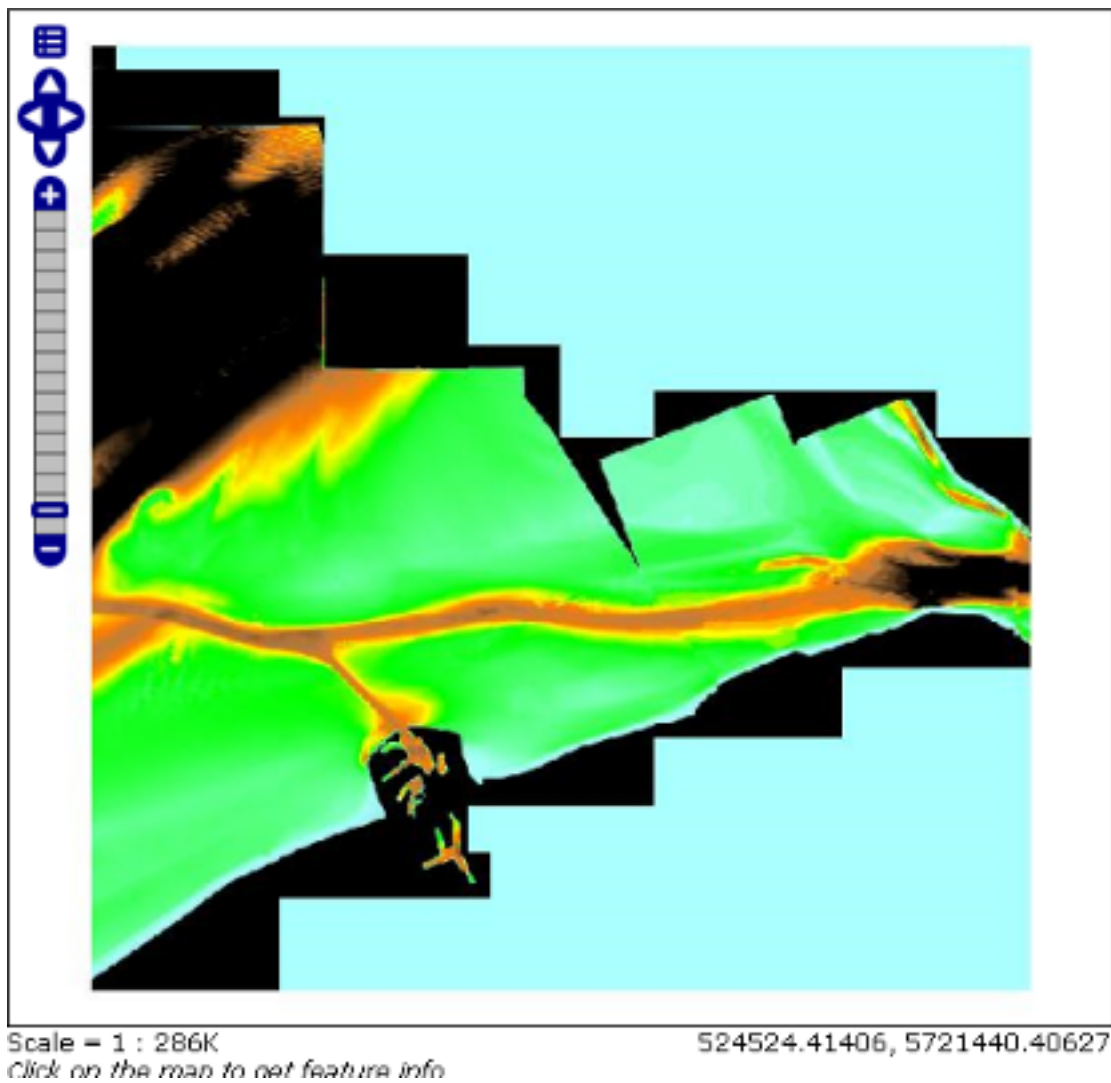


Figure 17.29: Basic configuration

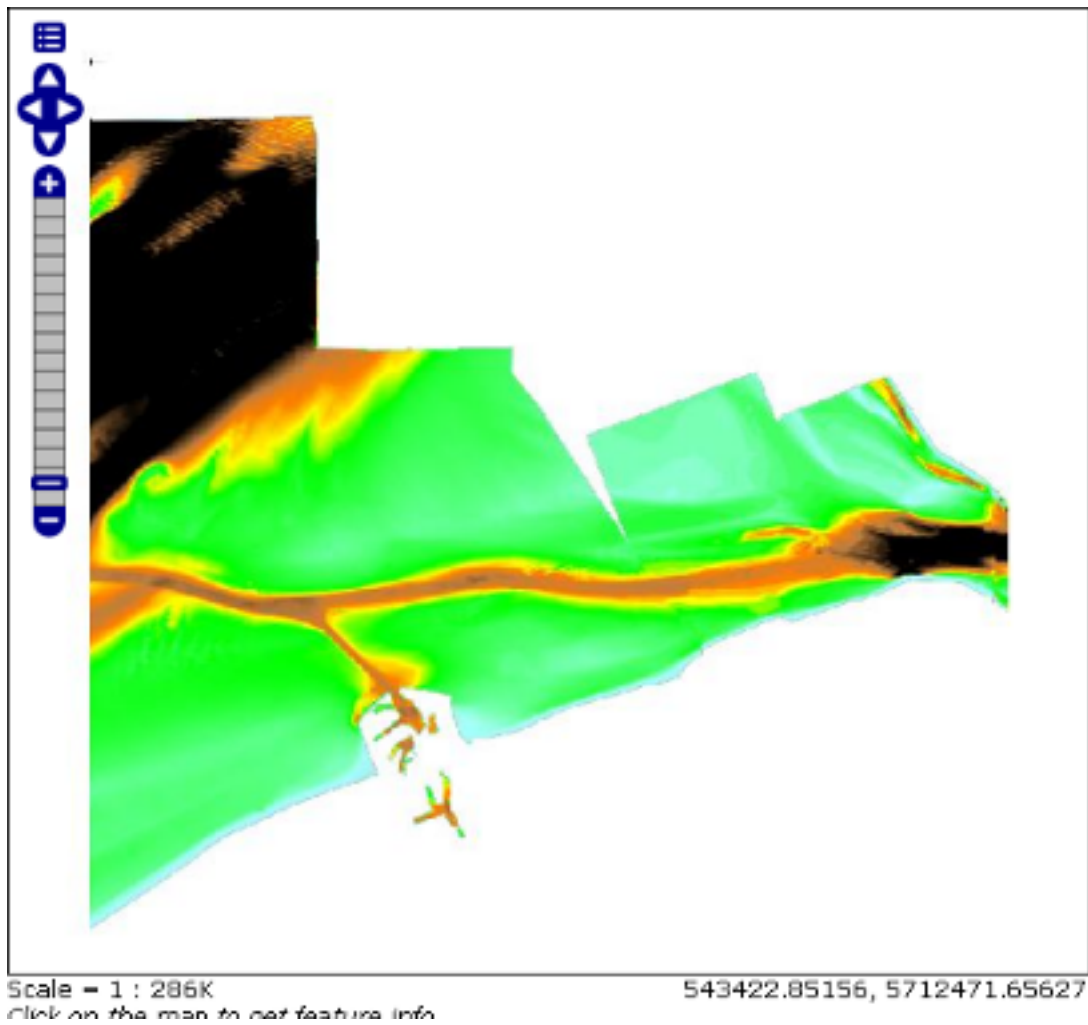


Figure 17.30: *Advanced configuration*

6. AllowMultiThreading: true.
7. USE\_JAI\_IMAGEREAD: true.
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/StyledLayerDescriptor.xsd">
  <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 entirely cover its bounding box. The areas within the BBOX that are not covered with data will default to a value of 0 on each band. Since this mosaic is RGB we can 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.





Figure 17.31: Basic configuration

The results is the following:

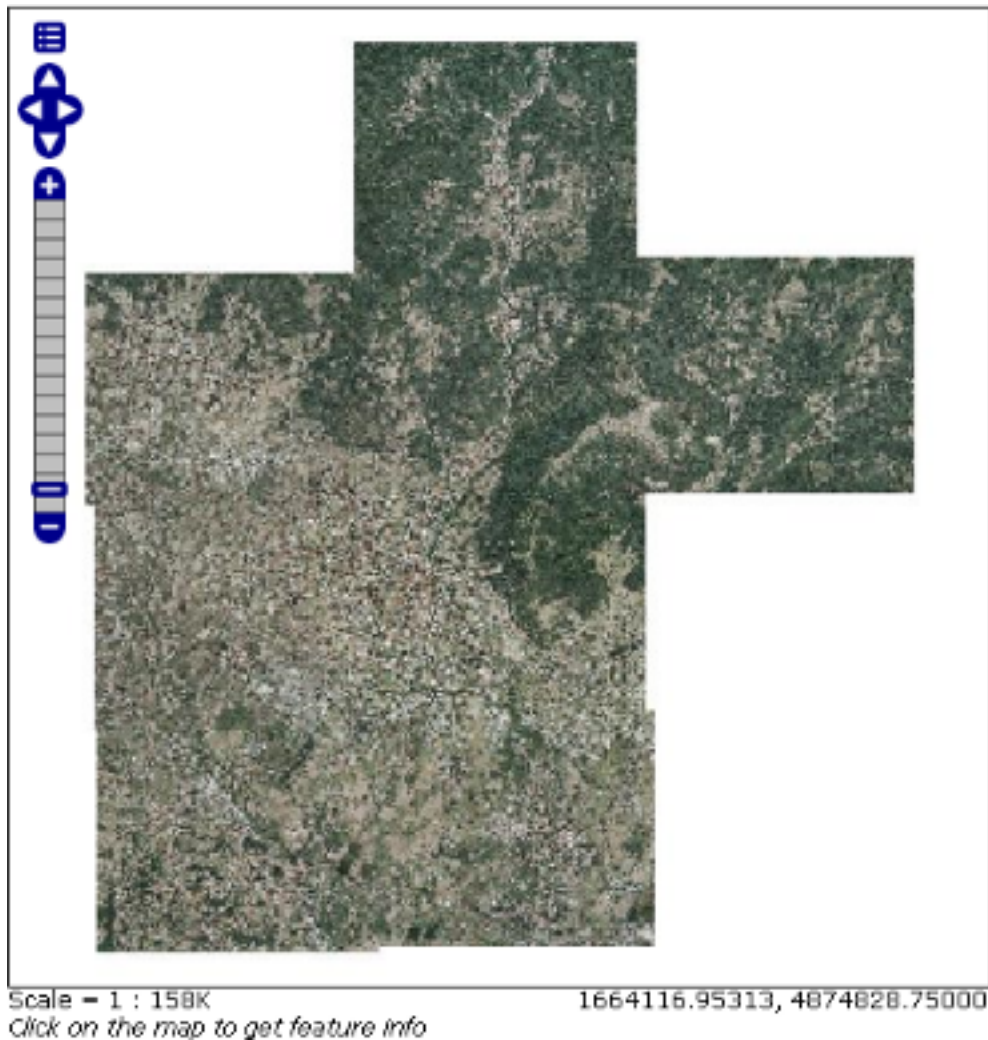


Figure 17.32: *Advanced configuration*

### 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 'InputTransparentColor' parameter at to make the collar areas disappear during the superimposition process, as instance, in this case, by using the '#FFFFFF' 'InputTransparentColor'.

This is the result:

The final configuration parameters are the followings:

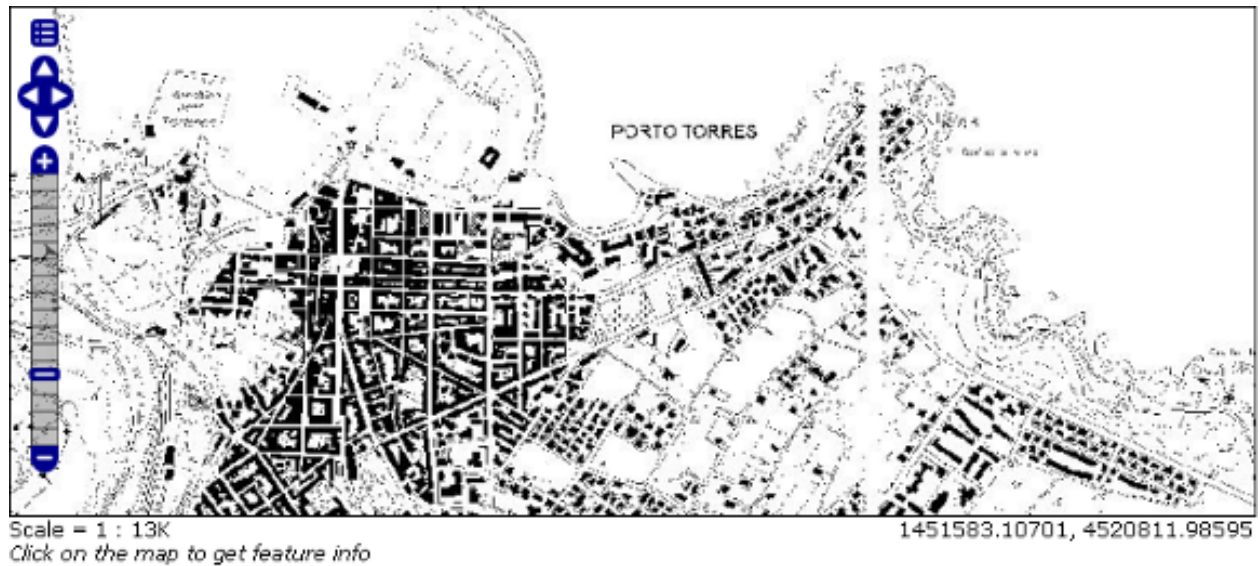


Figure 17.33: Basic configuration

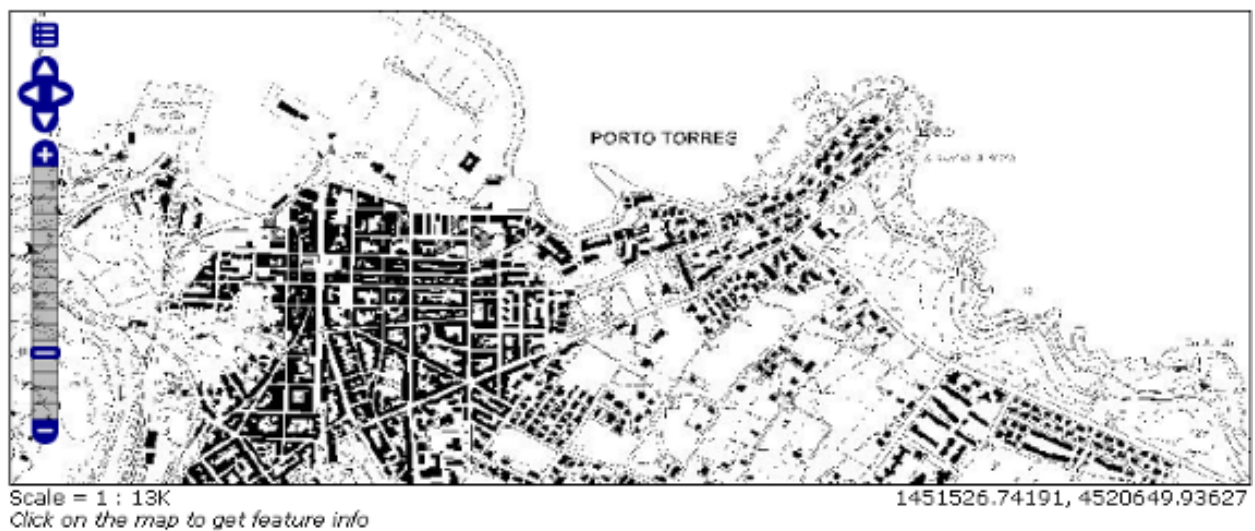


Figure 17.34: Advanced configuration

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.

## 17.9 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.

### 17.9.1 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:

**ImagePyramid**  
Image pyramidal plugin

**Basic Store Info**

**Workspace \***

cite ▼

**Data Source Name \***

bm\_pyramid

**Description**

☒ Enabled

**Connection Parameters**

**URL \***

/home/aaime/devel/pyramid\_tutorial/pyramid

Figure 17.35: 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:

## New Layer chooser

Add layer from

Here is a list of resources contained in the store 'bm\_pyramid'. Click on the layer you wish to configure

Results 0 to 0 (out of 0 items)

Published	Layer name	
	bmpyramid	<a href="#">Publish</a>

Results 0 to 0 (out of 0 items)

Figure 17.36: Choosing the coverage for publishing

Publish the layer, and then setup the layer parameter `USE_JAI_IMAGEREAD` to `false` to get better scalability:

### Coverage Parameters

#### AllowMultithreading

#### BackgroundValues

#### InputTransparentColor

#### MaxAllowedTiles

#### OutputTransparentColor

#### SUGGESTED\_TILE\_SIZE

#### USE\_JAI\_IMAGEREAD

Figure 17.37: Tuning the pyramid parameters

Submit and go to the preview, the pyramid should be ready to use:

### 17.9.2 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

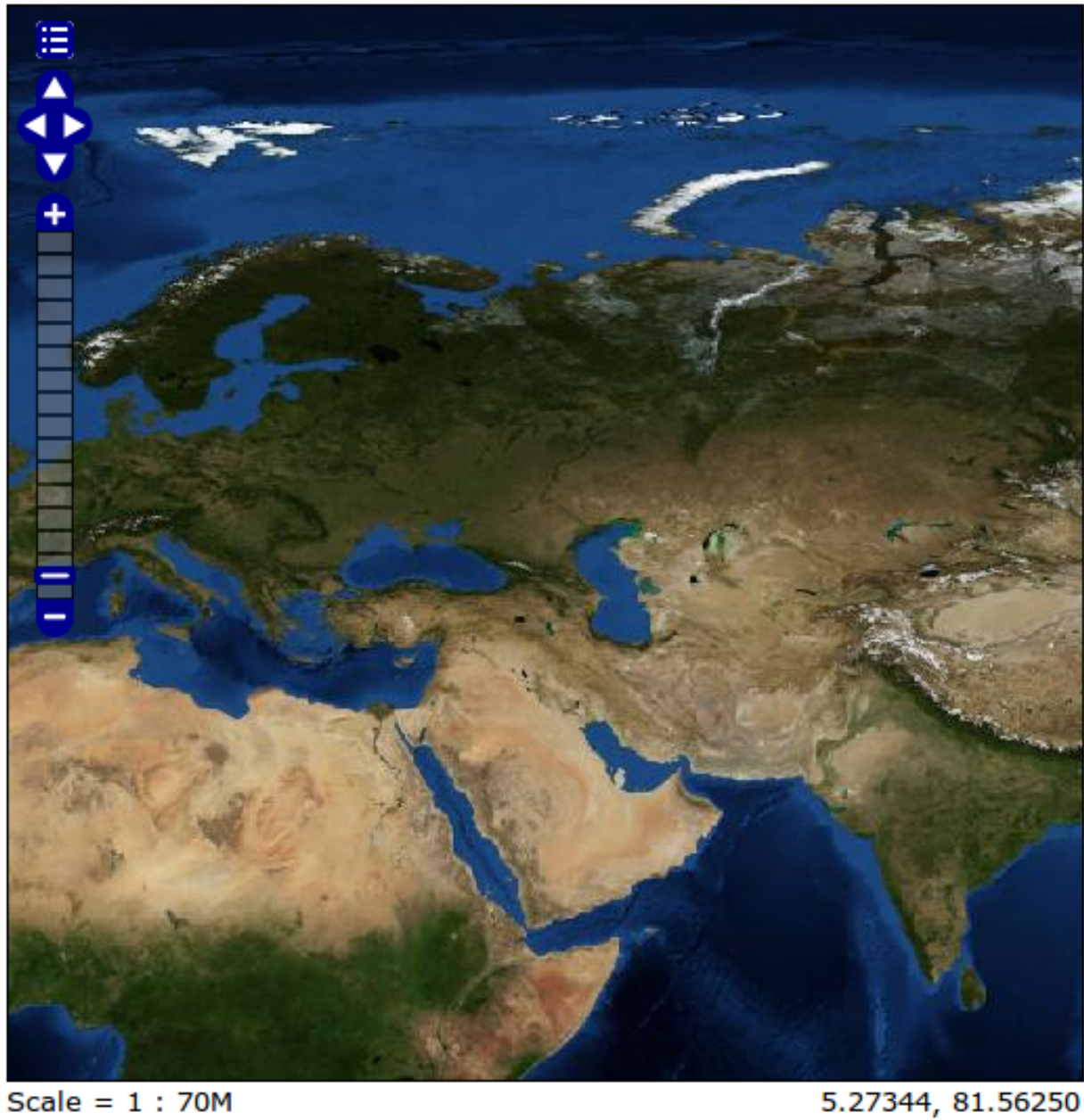


Figure 17.38: *Previewing the pyramid*

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
```

## 17.10 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.

### 17.10.1 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.

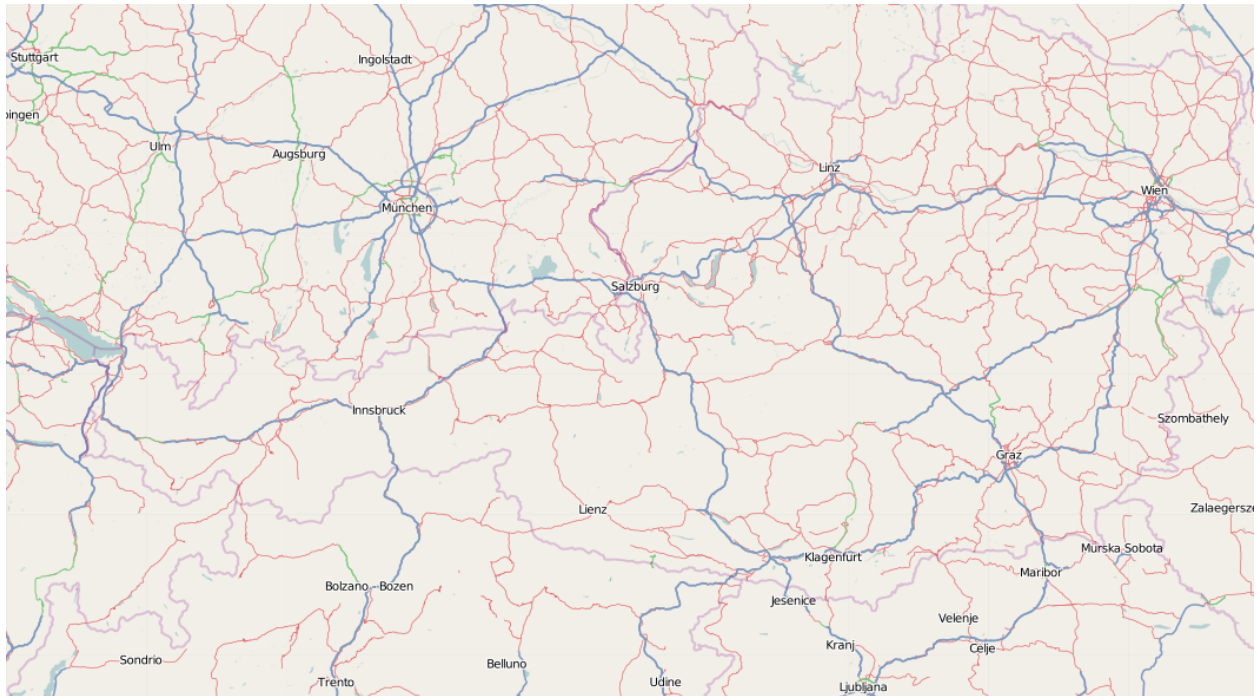
The full documentation is available here: <http://docs.codehaus.org/display/GEOTDOC/Image+Mosaicing+Pyramidal+JD>

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)

### 17.10.2 Getting Started

We use postgis/postgres as database engine, a database named “gis” and start with an image from open-streetmap. We also need this utility [http://www.gdal.org/gdal\\_retile.html](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, let's 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
```

### 17.10.3 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](http://star.pst.qub.ac.uk/idl/Image_Tiling.html)

### 17.10.4 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 =  $\log(\text{pixelsize of image}) / \log(2) - \log(\text{pixelsize of tile}) / \log(2)$ .**

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. Take 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](http://www.gdal.org/gdal_retile.html) from the gdal project.

The executable 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

### 17.10.5 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.

1. 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"/>
    <tileTableNameAttribute name="TileTable" />
    <spatialTableNameAttribute 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.

1. 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 :-)

### 17.10.6 Using the java ddl generation utility

The full documentation is here: <http://docs.codehaus.org/display/GEOTDOC/Using+the+java+ddl+generation+utility>

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}.jar
```

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}.j
```

Explanation of parameters

parameter	description
ddl	create ddl statements
-config	the file name of our <code>osm.postgis.xml</code> 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
-spatialTNPref	A prefix for tablenamees 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.

### 17.10.7 Executing the DDL scripts

For user “postgres”, database “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.

### 17.10.8 Importing the image data

The full documentation is here: <http://docs.codehaus.org/display/GEOTDOC/Using+the+java+import+utility>

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`.

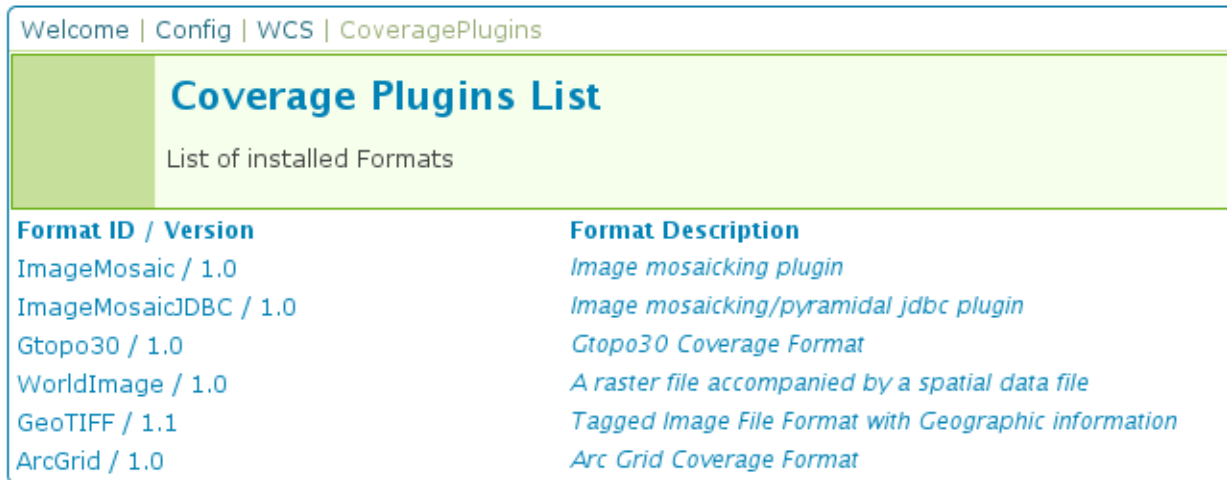
Change to the working directory and execute:

```
java -jar <your_geoserver_install_dir>/webapps/geoserver/WEB-INF/lib/gt-imagemosaic-jdbc-{version}.j
```

This statement imports your tiles including all pyramids into your database.

### 17.10.9 Configuring GeoServer

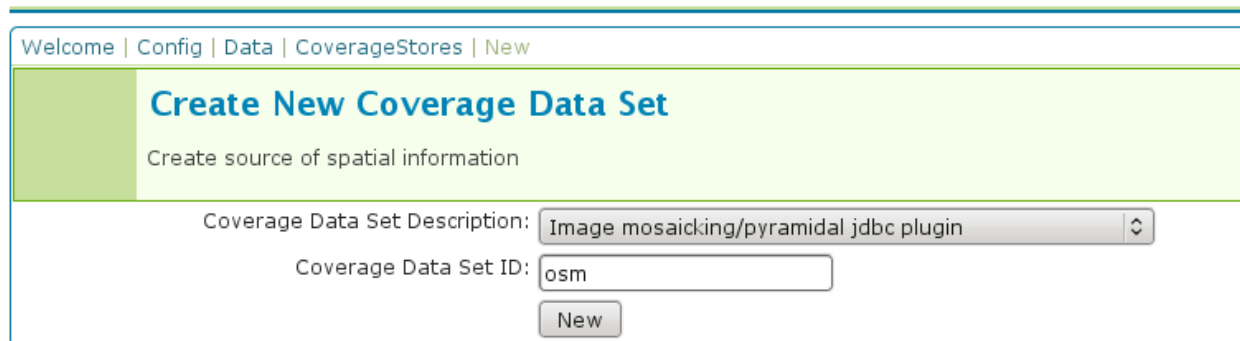
Start GeoServer and log in. Under *Config* → *WCS* → *CoveragePlugins* you should see



Format ID / Version	Format Description
ImageMosaic / 1.0	Image mosaicking plugin
ImageMosaicJDBC / 1.0	Image mosaicking/pyramidal jdbc plugin
Gtopo30 / 1.0	Gtopo30 Coverage Format
WorldImage / 1.0	A raster file accompanied by a spatial data file
GeoTIFF / 1.1	Tagged Image File Format with Geographic information
ArcGrid / 1.0	Arc Grid Coverage Format

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

**My GeoServer**



Coverage Data Set Description:

Coverage Data Set ID:

Press **New** and fill in the formular

## my Geoserver

Welcome | Config | Data | CoverageStores | Edit

## Coverage Data Set Editor

Edit a source of spatial information

Coverage Data Set ID: **osm**

Enabled: ☒

Namespace:

Type:

\* URL:

Description:

\* = required field

Press **Submit**.

Press **Apply**, then **Save** to save your changes.

Next select *Config*→*Data*→*Coverages*→*New* and select “osm”.

## My GeoServer

Welcome | Config | Data | Coverages | New

## Create New Coverage

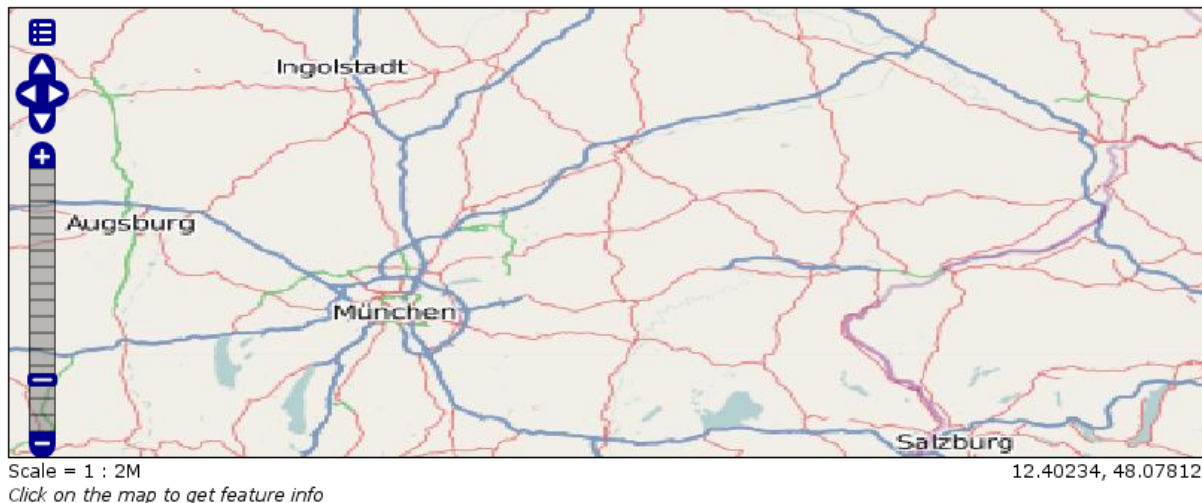
Create a new Coverage from an available Format

Coverage Data Set ID:

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 postgis (at the time of this writing). So I used EPSG:4326. For the purpose of this tutorial, this is ok.

### 17.10.10 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.

## 17.11 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.

### 17.11.1 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.codehaus.org/display/GEOTDOC/Feature-Pregeneralized>

This tutorial will show two possible scenarios, explaining step by step what to do for using this module in GeoServer.



### 17.11.2 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/
```

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

### 17.11.3 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" featur
    <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

Welcome | Config | Data | DataStores | Edit Logout

## Feature Data Set Editor

Edit a source of spatial information

Feature Data Set ID: **GenStreams1**

Enabled: ☒

Namespace: **topp**

Description:

\* RepositoryClassName: **org.geoserver.data.gen.DSFinderRepository**

\* GeneralizationInfosProviderClassName: **org.geoserver.data.gen.info.GeneralizationInfosProviderImpl**

GeneralizationInfosProviderParam: **file:data/streams/geninfo\_shapefile.xml**

\* = required field

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**.

Name: **GenStreams**

Alias:

Style: **line**

Additional Styles: 

burg  
capitals  
cite\_lakes  
dem  
giant\_polygon  
grass  
green  
line

>> <<

SRS: **26713**  [SRS Help](#) - [SRS List](#)

SRS WKT: PROJCS["NAD27 / UTM zone 13N", GEOGCS["NAD27", DATUM["North American Datum 1927", SPHEROID["Clarke 1866", 6378206.4, 294.9786982138982, AUTHORITY["EPSG","7008"]], TOWGS84[-4.2, 135.4, 181.9, 0.0, 0.0, 0.0, 0.0], AUTHORITY["EPSG","6267"]], PRIMEM["Greenwich", 0.0, AUTHORITY["EPSG","8901"]], UNIT["degree", 0.017453292519943295], AXIS["Geodetic longitude", EAST], AXIS["Geodetic latitude", NORTH], AUTHORITY["EPSG","4267"]], PROJECTION["Transverse Mercator", AUTHORITY["EPSG","9807"]], PARAMETER["central\_meridian", -105.0], 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","26713"]]

Native SRS WKT: PROJCS["NAD\_1927\_UTM\_Zone\_13N", GEOGCS["GCS\_North\_American\_1927", DATUM["D\_North\_American\_1927", SPHEROID["Clarke\_1866", 6378206.4, 294.9786982]], PRIMEM["Greenwich", 0.0], UNIT["degree", 0.017453292519943295], AXIS["Longitude", EAST], AXIS["Latitude", NORTH]], PROJECTION["Transverse\_Mercator"] PARAMETER["central\_meridian", -105.0], 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["x", EAST], AXIS["y", NORTH]]

SRS handling: **Force declared SRS (native will be ignored)**

Title: **GenStreams\_Type**

Bounding Box:

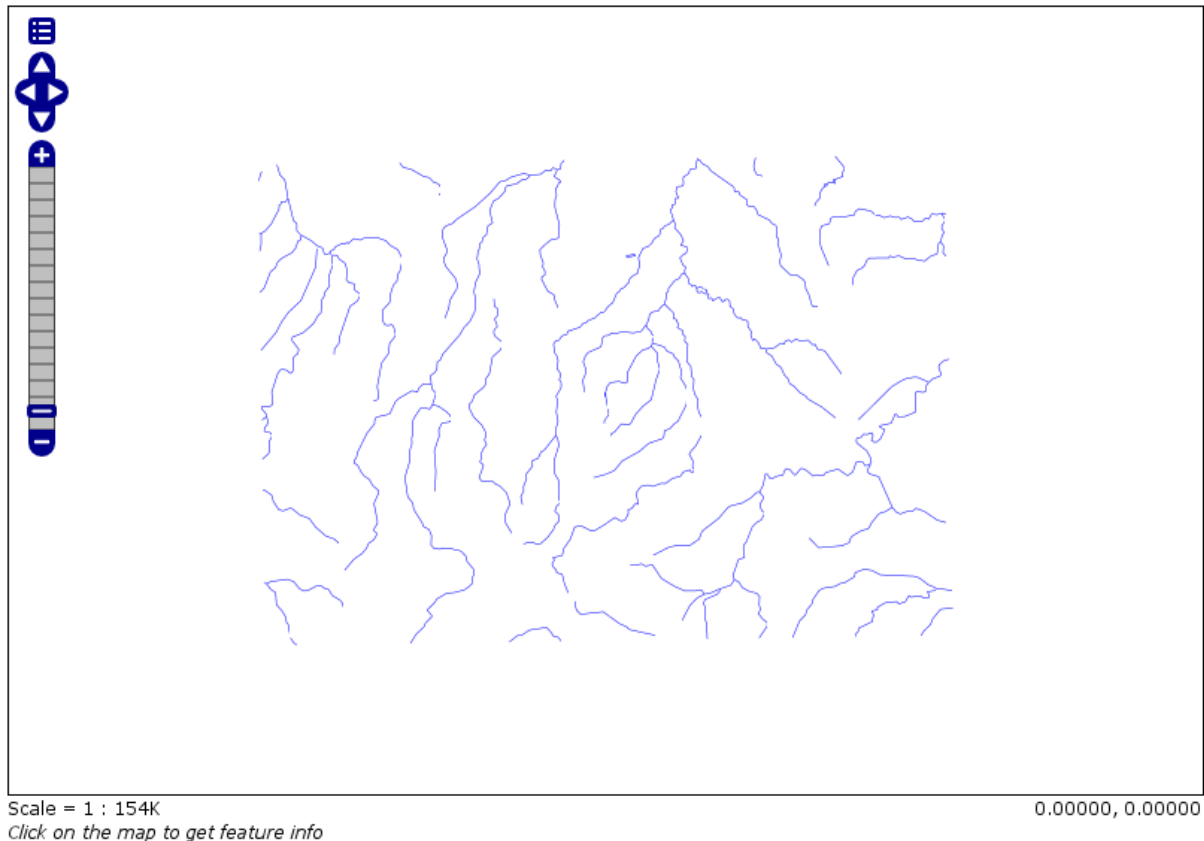
Data min X:	589443.06	Data min Y:	4913935.46
Data max X:	609526.75	Data max Y:	4928059.15

Min Long: **-103.877326154750** Min Lat: **44.3702348938932**

Max Long: **-103.622320736390** Max Lat: **44.50011993037069**

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
```

### 17.11.4 Public Deployment

First we have to configure all our shape files

Welcome | Config | Data | DataStores | New Logout

### Create New Feature Data Set

Create source of spatial information

Feature Data Set Description: Shapefile

Feature Data Set ID: Streams\_0

New

The **Feature Data Set ID** for the other shape files is

1. Streams\_5
2. Streams\_10
3. Streams\_20
4. Streams\_50

The screenshot shows the 'Feature Data Set Editor' web interface. At the top, there is a navigation bar with links: 'Welcome', 'Config', 'Data', 'DataStores', and 'Edit'. Below this is a green header bar with the title 'Feature Data Set Editor' and the subtitle 'Edit a source of spatial information'. The main form area contains the following fields and controls:

- Feature Data Set ID:** A text field containing 'Streams\_0'.
- Enabled:** A checkbox that is checked.
- Namespace:** A dropdown menu showing 'topp'.
- Description:** A large empty text area.
- \* url:** A text field containing 'file:data/streams/0/streams.shp'. The asterisk indicates it is a required field.
- create spatial index:** A dropdown menu.
- charset:** A text field containing 'ISO-8859-1'.
- memory mapped buffer:** A dropdown menu.
- Buttons:** 'Submit' and 'Reset' buttons at the bottom right.

At the bottom left, a legend states: '\* = required field'.

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

Name: **streams**

Alias:

Style:  [Create new SLD](#)

Additional Styles:

burg	
capitals	
cite_lakes	
dem	
giant_polygon	
grass	
green	
line	

SRS:  [Lookup SRS](#) [SRS Help](#) - [SRS List](#)

SRS WKT: PROJCS["NAD27 / UTM zone 13N", GEOGCS["NAD27", DATUM["North American Datum 1927", SPHEROID["Clarke 1866", 6378206.4, 294.9786982138982, AUTHORITY["EPSG","7008"]], TOWGS84[-4.2, 135.4, 181.9, 0.0, 0.0, 0.0, 0.0], AUTHORITY["EPSG","6267"]], PRIMEM["Greenwich", 0.0, AUTHORITY["EPSG","8901"]], UNIT["degree", 0.017453292519943295], AXIS["Geodetic longitude", EAST], AXIS["Geodetic latitude", NORTH], AUTHORITY["EPSG","4267"]], PROJECTION["Transverse Mercator", AUTHORITY["EPSG","9807"]], PARAMETER["central\_meridian", -105.0], 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","26713"]]

Native SRS WKT: PROJCS["NAD 1927 UTM Zone 13N", GEOGCS["GCS North American 1927", DATUM["D North American 1927", SPHEROID["Clarke 1866", 6378206.4, 294.9786982]], PRIMEM["Greenwich", 0.0], UNIT["degree", 0.017453292519943295], AXIS["Longitude", EAST], AXIS["Latitude", NORTH]], PROJECTION["Transverse Mercator"] PARAMETER["central\_meridian", -105.0], 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["x", EAST], AXIS["y", NORTH]]

SRS handling:

Title:

Bounding Box: [Generate](#)

Data min X:	589443.06	Data min Y:	4913935.46
Data max X:	609526.75	Data max Y:	4928059.15
Min Long:	-103.8773261547504	Min Lat:	44.3702348938932
Max Long:	-103.6223207363905	Max Lat:	44.50011993037069

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.

Now 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"
    <Generalization dataSourceNameSpace="topp" dataSourceName="Streams_20" distance="20"
    <Generalization dataSourceNameSpace="topp" dataSourceName="Streams_50" distance="50"
  </GeneralizationInfo>
</GeneralizationInfos>
```

Save this file as `geninfo_shapefile2.xml` into `GEOSERVER_DATA_DIR/data/streams`.

Create the pregeneralized datastore

Welcome | Config | Data | DataStores | New Logout

---

## Create New Feature Data Set

Create source of spatial information

Feature Data Set Description:

Feature Data Set ID:

Now we use the **CatalogRepository** class to find our needed data stores

Welcome | Config | Data | DataStores | Edit Logout

---

## Feature Data Set Editor

Edit a source of spatial information

Feature Data Set ID: [GenStreams2](#)

Enabled: ☒

Namespace:

Description:

\* RepositoryClassName:

\* GeneralizationInfosProviderClassName:

GeneralizationInfosProviderParam:

\* = required field

Last step

Name: **GenStreams2**

Alias:

Style: **line**

Additional Styles: **burg**, **capitals**, **cite\_lakes**, **dem**, **giant\_polygon**, **grass**, **green**, **line**

SRS: **26713**  [SRS Help - SRS List](#)

SRS WKT: PROJCS["NAD27 / UTM zone 13N", GEOGCS["NAD27", DATUM["North American Datum 1927", SPHEROID["Clarke 1866", 6378206.4, 294.9786982138982, AUTHORITY["EPSG","7008"]], TOWGS84[-4.2, 135.4, 181.9, 0.0, 0.0, 0.0], AUTHORITY["EPSG","6267"]], PRIMEM["Greenwich", 0.0, AUTHORITY["EPSG","8901"]], UNIT["degree", 0.017453292519943295], AXIS["Geodetic longitude", EAST], AXIS["Geodetic latitude", NORTH], AUTHORITY["EPSG","4267"]], PROJECTION["Transverse Mercator", AUTHORITY["EPSG","9807"]], PARAMETER["central\_meridian", -105.0], 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","26713"]]

Native SRS WKT: PROJCS["NAD\_1927\_UTM\_Zone\_13N", GEOGCS["GCS\_North\_American\_1927", DATUM["D\_North\_American\_1927", SPHEROID["Clarke\_1866", 6378206.4, 294.9786982]], PRIMEM["Greenwich", 0.0], UNIT["degree", 0.017453292519943295], AXIS["Longitude", EAST], AXIS["Latitude", NORTH], PROJECTION["Transverse Mercator"], PARAMETER["central\_meridian", -105.0], 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["x", EAST], AXIS["y", NORTH]]

SRS handling: **Force declared SRS (native will be ignored)**

Title: **GenStreams2\_Type**

Bounding Box:

Data min X:	589443.06	Data min Y:	4913935.46
Data max X:	609526.75	Data max Y:	4928059.15
Min Long:	-103.8773261547504	Min Lat:	44.3702348938932
Max Long:	-103.6223207363905	Max Lat:	44.50011993037069

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
```

### 17.11.5 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.



## 17.12 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

### 17.12.1 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).

For more information about the possible parameters and their values refer to the [DBCP documentation](#).

### 17.12.2 GeoServer setup

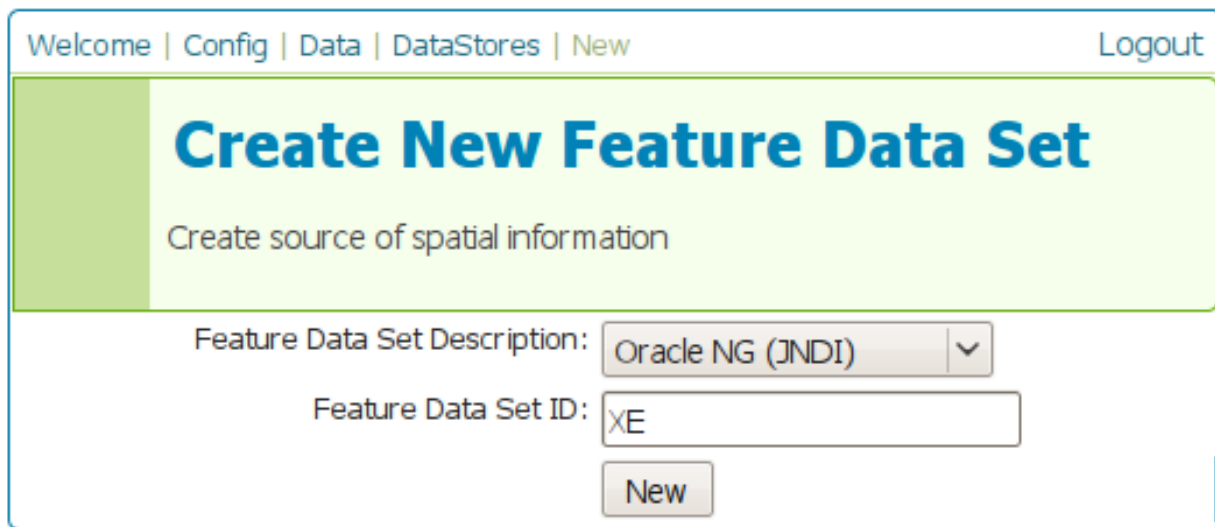
To allow a web application reference to a JNDI resource its `web.xml` file must be modified so that the reference is explicit. Following the above example, we have to modify

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:



Welcome | Config | Data | DataStores | New Logout

## Create New Feature Data Set

Create source of spatial information

Feature Data Set Description: Oracle NG (JNDI) ▼

Feature Data Set ID: XE

New

Figure 17.39: 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.

## Feature Data Set Editor

Edit a source of spatial information

Feature Data Set ID: [XE](#)

Enabled: ☒

Namespace:

Description:

\* jndiReferenceName:

schema:

\* = required field

Figure 17.40: Configuring the JNDI connection



---

# 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.

## 18.1 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.

### 18.1.1 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

This avoid a single user to make too many requests in parallel:

```
user=<count>
```

Where `<count>` is the maximum number of parallel requests a single user can execute in parallel. The user tracking mechanism is cookie based, so it will work fine for browsers but not as much for other kinds of clients. An IP based mechanism is not provided at the time, but it would have its own fallacies as well, as it would limit all the users sitting behind a single router to `<count>` requests (imagine the effect on a big public administration).

## 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.

### 18.1.2 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
```

## 18.2 GeoServer CSS Module

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:

### 18.2.1 Installing the GeoServer CSS Module

The CSS module is built nightly and published to the [nightly build server](#). The installation process is similar to other GeoServer plugins:

1. Download the file named like `geoserver-2.0.2-SNAPSHOT-css-plugin.zip`. 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/` directory in the GeoServer webapp. For example, if you have installed the GeoServer binary to `/opt/geoserver-2.1.0/`, you should place the CSS extension's JAR files in `/opt/geoserver-2.1.0/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 document useful in getting started styling layers with CSS: [Tutorial: Converting an SLD to CSS](#).

### 18.2.2 Tutorial: Converting an SLD to CSS

This tutorial will walk through installing the CSS plugin for GeoServer and using it to style the states data that is included with GeoServer.

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 GeoServer 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.

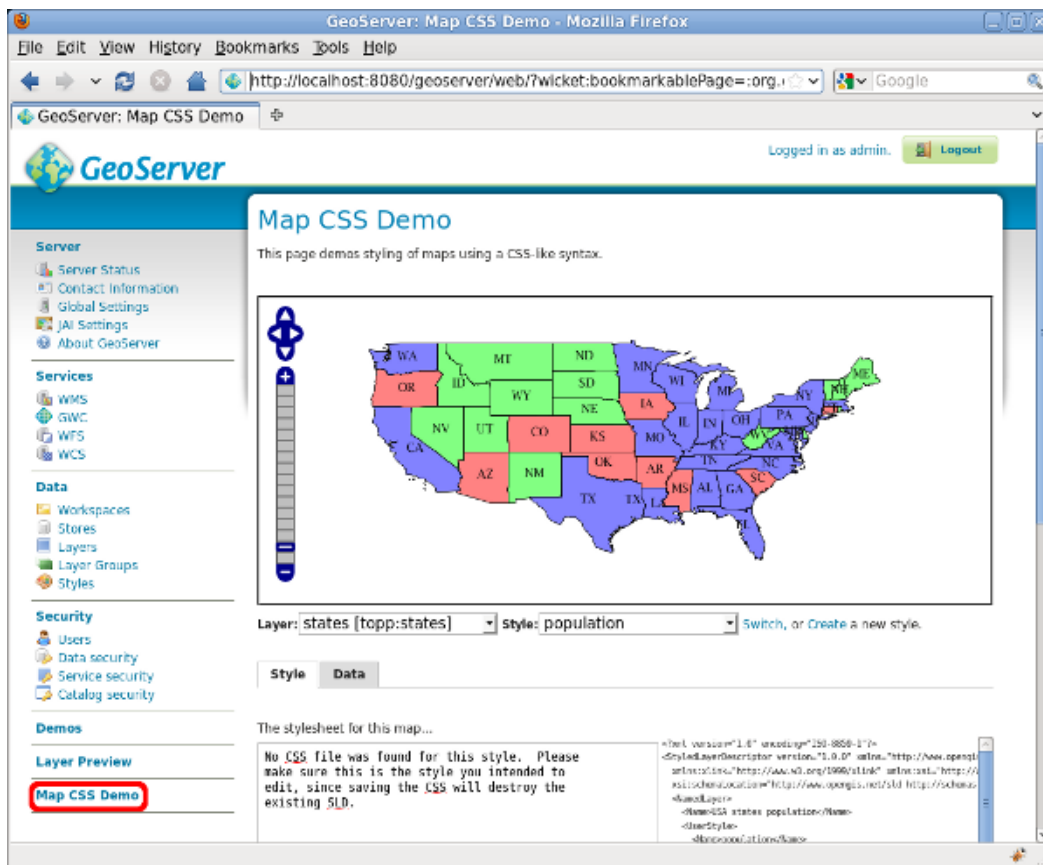


Figure 18.1: The CSS demo page can be used to switch between layers and styles. Note the sidebar link, highlighted in red.

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:



```

<?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;
  stroke: black;
  mark: symbol(square);
}
```

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 `filters` and `properties` 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];
  font-family: "Times New Roman";
  font-style: normal;
  font-size: 14;
}
```

**Note:** You may have noticed this snippet doesn't include the LabelPlacement fields from the SLD. The CSS module doesn't yet support this option.

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];
  font-family: "Times New Roman";
  font-style: normal;
  font-size: 14;
}
```

Press the **Submit** button at the bottom of the CSS form to 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 specifies 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 [properties](#) 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];
  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];
  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];
  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];
  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.

### 18.2.3 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:

Op- era- tor	Meaning
=	The property must be exactly <i>equal</i> 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 <code>_</code> to indicate a single unspecified character and <code>%</code> 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 wiki](#).



## 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 GeoServer 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;
}
```

## 18.2.4 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.

## 18.2.5 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: mark(circle), mark(circle), mark(circle), mark(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: mark(circle), mark(circle), mark(circle), mark(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.

## 18.2.6 Property Listing

This page lists the supported rendering properties. See `values` for more information about the value types for each.

### Point Symbolology

Property	Type	Meaning	Accepts Expression?
<code>mark</code>	<code>url,symbol</code>	The image or well-known shape to render for points	yes
<code>mark-geometry</code>	<code>expression</code>	An expression to use for the geometry when rendering features	yes
<code>mark-size</code>	<code>length</code>	The width to assume for the provided image. The height will be adjusted to preserve the source aspect ratio.	yes
<code>mark-rotation</code>	<code>angle</code>	A rotation to be applied (clockwise) to the mark image.	yes

### Line Symbolology

Property	Type	Meaning	Accepts Expression?
<code>stroke</code>	<code>color, url, symbol</code>	The color, graphic, or well-known shape to use to stroke lines or outlines	yes
<code>stroke-geometry</code>	<code>expression</code>	An expression to use for the geometry when rendering features.	yes
<code>stroke-mime-type</code>	<code>string</code>	The mime-type of the external graphic provided. This is <b>required</b> when using external graphics	yes
<code>stroke-opacity</code>	<code>percentage</code>	A value in the range of 0 (fully transparent) to 1.0 (fully opaque)	yes
<code>stroke-width</code>	<code>length</code>	The width to use for stroking the line.	yes
<code>stroke-size</code>	<code>length</code>	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
<code>stroke-rotation</code>	<code>angle</code>	A rotation to be applied (clockwise) to the stroke image. See also the <code>stroke-repeat</code> property.	yes
<code>stroke-linecap</code>	<code>keyword: butt, square, round</code>	The style to apply to the ends of lines drawn	yes
<code>stroke-linejoin</code>	<code>keyword: miter, round, bevel</code>	The style to apply to the “elbows” where segments of multi-line features meet.	yes
<code>stroke-dasharray</code>	<code>list of lengths</code>	The lengths of segments to use in a dashed line.	no
<code>stroke-dashoffset</code>	<code>length</code>	How far to offset the dash pattern from the ends of the lines.	yes
<code>stroke-repeat</code>	<code>keyword: repeat, stipple</code>	How to use the provided graphic to paint the line. If <code>repeat</code> , then the graphic is repeatedly painted along the length of the line (rotated appropriately to match the line’s direction). If <code>stipple</code> , then the line is treated as a polygon to be filled.	yes

## 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	The mime-type of the external graphic provided. This is <i>required</i> when using external graphics	yes
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



## 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
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 <i>required</i> 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 <a href="#">spaceAround</a> 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 <a href="#">group</a> 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 <a href="#">maxDisplacement</a> vendor parameter.	no
-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 <a href="#">repeat</a> 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 <a href="#">allGroup</a> 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-overrun	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 <a href="#">followLine</a> 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 <a href="#">maxAngleDelta</a> vendor parameter.	no
622 -gt-label-auto-wrap	length	Labels will be wrapped to multiple lines if they exceed this length in pixels. This is equivalent to the <a href="#">autoWrap</a> vendor parameter.	no
-gt-label-forced-flip	one of: true or false	By default, the renderer will flip labels whose normal	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 GeoServer 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

## 18.2.7 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 `multivalues`.

### 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.

## Colors

Color values are relatively important to styling, so there are multiple ways to specify them.

Format	Interpretation
<code>#RRGGBB</code>	A hexadecimal-encoded color value, with two digits each for red, green, and blue.
<code>#RGB</code>	A hexadecimal-encoded color value, with one digit each for red, green, and blue. This is equivalent to the two-digit-per-channel encoding with each digit duplicated.
<code>rgb(r, g, b)</code>	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.
<i>Simple name</i>	The simple English name of the color. A full list of the supported colors is available at <a href="http://www.w3.org/TR/SVG/types.html#ColorKeywords">http://www.w3.org/TR/SVG/types.html#ColorKeywords</a>

## 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/);  
}
```

## 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");
}
```

### 18.2.8 Styled Marks in GeoServer 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
- `: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 *n*th stacked point marker on a feature.
- `:nth-stroke(n)` specifies that a rule applies to the symbol used for the *n*th stacked stroke pattern on a feature.
- `:nth-fill(n)` specifies that a rule applies to the symbol used for the *n*th stacked fill pattern on a feature.
- `:nth-symbol(n)` specifies that a rule applies to the symbol used for the *n*th 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;
  }

```

## 18.3 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
2. Binary Interleaved by Line(BIL) - image/bil

### 18.3.1 Installing the DDS/BIL extension

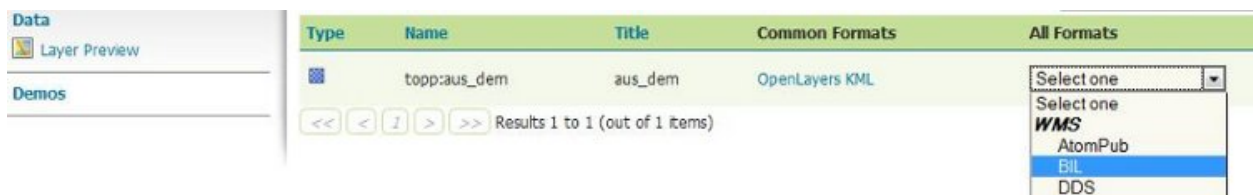
1. Download the DDS/BIL extension from the [nightly GeoServer community module builds](#).

**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.

### 18.3.2 Checking if the extension is enabled

Once the extension is installed, the provided mime-types should appear in the layer preview dropbox as shown:



### 18.3.3 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.

## 18.4 Monitoring

The monitoring extension provides a request monitor for GeoServer. It captures information about each request a GeoServer instance handles and produces reports based on the request data.

### 18.4.1 Installing the Monitoring Extension

Monitoring is a community extension, and thus is not found on the standard GeoServer release download pages. Community extensions are only available via [Nightly builds](#) or by compiling from source.

1. Download the proper “monitoring” extension linked from the [GeoServer nightly builds 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.

### Verifying the Installation

There are two ways to verify that the monitoring extension has been properly installed.

- 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.

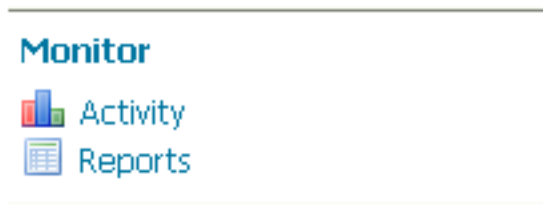


Figure 18.2: Monitoring section in the web admin interface

- 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.

### 18.4.2 Monitoring Overview

The following diagram outlines the architecture of the monitoring extension:

As a request is processed the monitor inserts itself at particular points in the request life cycle to capture necessary data. That data is persisted to an external database:

ID	STATUS	PATH	START_TIME	...
1	FINISHED	/wms	2008-25-10 08:21	...
2	FAILED	/wfs	2008-25-10 08:22	...
3	RUNNING	/wcs	2008-25-10 08:25	...

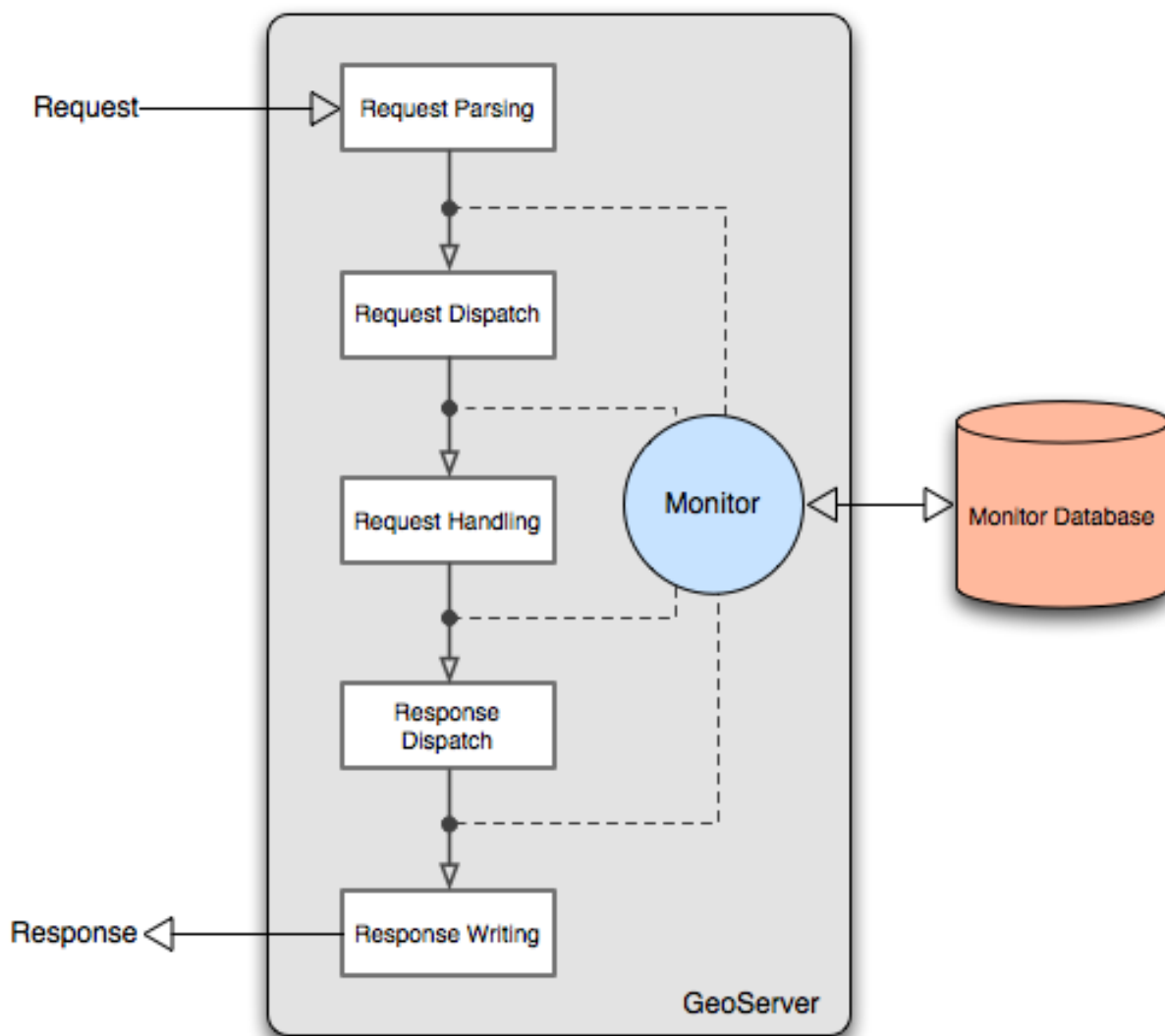


Figure 18.3: *Monitor extension architecture*

The request data is made available through various reports:

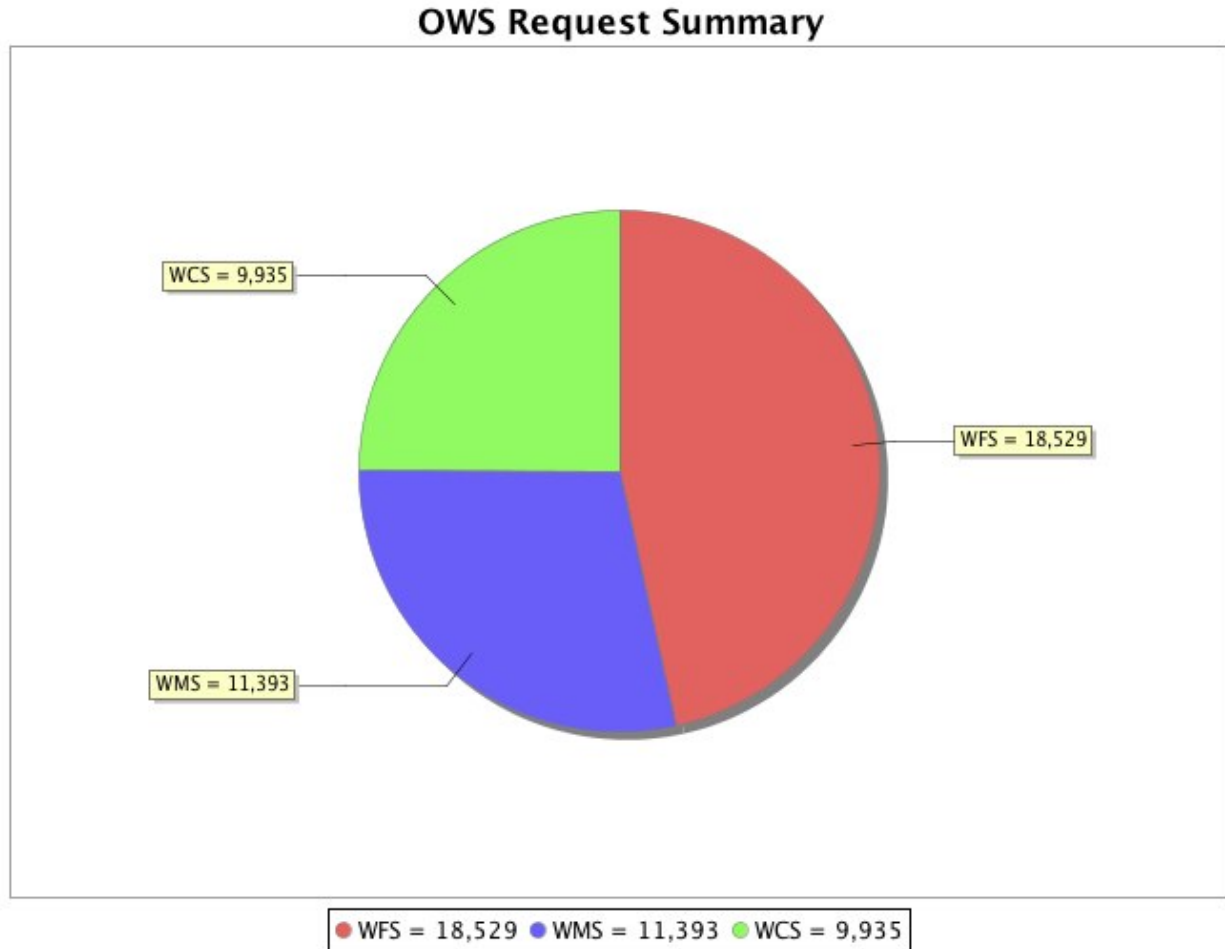


Figure 18.4: *Request report*

### 18.4.3 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 function of these files will be discussed below.



## Monitor Mode

The monitoring extension supports different “monitoring modes” that control how request data is captured and stored. Currently three modes are supported:

- **history** (*Default*) - Only historical request information is available. No live information is maintained.
- **live** - Only information about live requests is maintained.
- **mixed** - A combination of live and history. This mode is experimental.

The mode is set in the `monitor.properties` file.

## History Mode

History mode persists information about all requests in an external database. It does not provide any real time information. This mode is appropriate in cases where a user is most interested in analyzing request history over a given time period.

## Live Mode

Live mode only maintains short lived information about requests that are currently executing. It also maintains a small buffer of recent requests. No external database is used with this mode and no information is persisted for the long term.

This mode is most appropriate in cases where a user only cares about what a server is doing in real time and is not interested about request history.

## Mixed Mode

Mixed mode combines both live and history mode in that it maintains both real time information and persists all request data to the monitoring database. This mode however is experimental and comes with more overhead than the other two modes. This is because mixed mode must perform numerous database transactions over the life cycle of a single request (in order to maintain live information), whereas history mode only has to perform a single database transaction for a request.

This mode is most appropriate when both real time request information and request history are desired. This mode is also most appropriate in a clustered server environment in which a user is interested in viewing real time request information about multiple nodes in a cluster.

## Monitor Database

By default monitored request data is stored in an embedded H2 database located in the `monitoring` directory. 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:

```
driver=org.postgresql.Driver
url=jdbc:postgresql://192.168.1.124:5432/monitoring
username=bob
password=foobar
```

## Request Filters

By default not all requests are monitored. Those requests excluded include any web admin requests or any [Monitor HTTP 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:

```
/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](#).

### 18.4.4 Monitor HTTP API

The monitor extension provides an API for retrieving request information via a simple set of HTTP calls.

The most simple of all calls would be to retrieve information about all requests:

```
GET http://localhost:8080/geoserver/rest/monitor/requests.html
```

This would return an HTML document containing information about all requests. The general structure of a query for a set of requests is:

```
GET http://<host>:<port>/geoserver/rest/monitor/requests.<format>
```

Where `format` is the representation of the returned result and is one of:

- `html` - Representation as an HTML table.
- `csv` - Representation as a Comma Separated Value table.

A query for a single request 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` is as described above.

**Note:** An alternative to specifying the returned representation with the `format` extension is to use the `http Accept` header and specify one of the MIME types:

- `text/html`
- `application/csv`

See the [HTTP specification](#) for more information about the `Accept` header.

## API Reference

There are numerous parameters available that can be used to filter what request information is returned and how it is structured. This section contains a comprehensive list of all parameters. See the examples section for a set of examples of applying these parameters.

### count

Specifies how many records should be returned.

Syntax	Example
<code>count=&lt;integer&gt;</code>	<code>requests.html?count=100</code>

### offset

Specifies where in the result set records should be returned from.

Syntax	Example
<code>offset=&lt;integer&gt;</code>	<code>requests.html?count=100&amp;offset=500</code>

### live

Specifies that only live (currently executing) requests be returned.

Syntax	Example
<code>live=&lt;yes   no   true   false&gt;</code>	<code>requests.html?live=yes</code>

This parameter relies on a [Monitor Mode](#) being used that maintains real time request information (either **live** or **mixed**).

## from

Specifies an inclusive lower bound on the timestamp for the start of a request.

Syntax	Example
from=<timestamp>	requests.html?from=2010-07-23T16:16:44

## to

Specifies an inclusive upper bound on the timestamp for the start of a request.

Syntax	Example
to=<timestamp>	requests.html?to=2010-07-24T00:00:00

## order

Specifies which attribute of a request to sort by.

Syntax	Example
order=<attribute>[;<ASC   DESC>]	requests.html?order=path requests.html?order=startTime:DESC requests.html?order=totalTime:ASC

## Examples

### All requests as HTML

```
GET http://localhost:8080/geoserver/rest/monitor/requests.html
```

### All requests as CSV

```
GET http://localhost:8080/geoserver/rest/monitor/requests.csv
```

### Requests over a time period

```
GET http://localhost:8080/geoserver/rest/monitor/requests.html?from=2010-06-20&to=2010-07-20
```

### Requests 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
```

## 18.5 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.

### 18.5.1 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.

### 18.5.2 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.

### 18.5.3 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.

## 18.6 Python

The Python extension allows users to extend GeoServer dynamically by writing Python scripts via [jython](#), the Java implementation of Python.

### 18.6.1 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.

#### Verifying the Installation

To verify the extension has been installed properly start the GeoServer instance and navigate to the data directory. Upon named `python` will be created.

### 18.6.2 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.

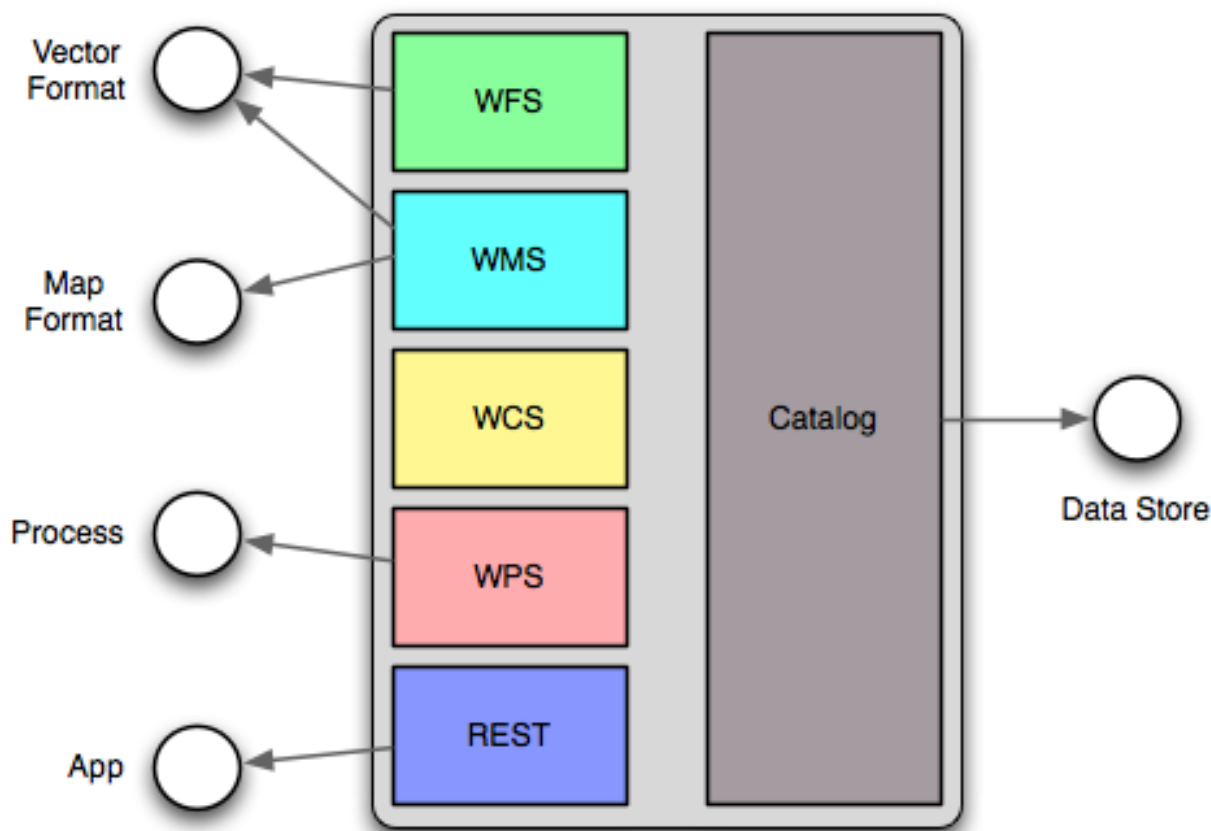


Figure 18.5: Python scripting extension hooks

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 corresponds 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.

## 18.6.3 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.

## 18.7 Spatialite

**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.

[Spatialite](#) is the spatial extension of the popular [SQLite](#) embedded relational database.

### 18.7.1 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.

### 18.7.2 Notes about shared 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 cease to function.

If the libraries are not installed on your system in a “default” location you must set the `LD_LIBRARY_PATH` (or equivalent) environment variable in order to load them at runtime.

**Note:** On Windows systems it is easiest to place the GEOS and PROJ dll's in the `C:\WINDOWS\system32` directory.

### 18.7.3 Adding a SpatiaLite database

Once the extension is properly installed `SpatiaLite` will show up as an option when creating a new data store.

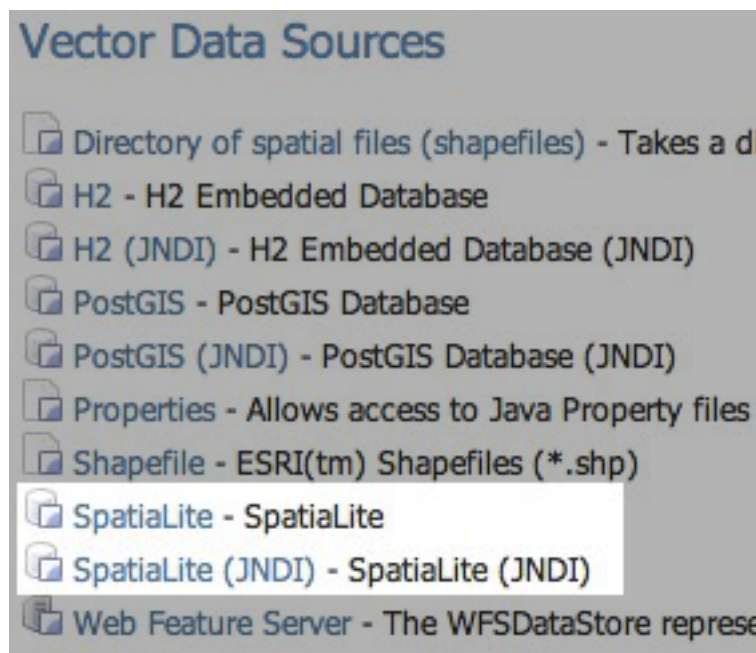


Figure 18.6: *SpatiaLite in the list of vector data sources*

## New Vector Data Source

Add a new vector data source

SpatialLite

SpatialLite

### Basic Store Info

Workspace \*

topp

Data Source Name \*

Description

☒ Enabled

### Connection Parameters

database

schema

passwd

Namespace \*

<http://www.openplans.org/topp>

☐ Expose primary keys

max connections

10

min connections

1

fetch size

1000

Connection timeout

20

Primary key metadata table

user

Save

Cancel

Figure 18.7: Configuring a SpatialLite data store

### 18.7.4 Configuring a SpatiaLite data store

database	The name of the database to connect to. See <a href="#">notes</a> 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	Connection pool configuration parameters. See the <a href="#">Database Connection Pooling</a> section for details.
min connections	

The *database* parameter may be specified as an absolute path or a relative one. When specified as a relative path the database will be created in the `spatialite` directory, located directly under the root of the GeoServer data directory.