**Introduction**

GeoServer is an open source software server written in Java that allows users to share and edit geospatial data that is published in Open Geospatial Consortium (OGC)-compliant services, including Web Feature Services (WFS) and Web Map Services (WMS) (<http://geoserver.org/display/GEOS/Welcome>). OGC services published as part of the National Geothermal Data System (NGDS, <http://geothermaldata.org/>) can use a variety of software servers and platforms, and GeoServer is an ideal candidate.

This document is meant to assist in simple GeoServer set-up using OpenGeo Suite as well as deployment of NGDS data services using Geoserver, with some additional configuration considerations for use of application schemas in Geoserver. It is assumed that schema-valid data tables already exists in an object-relational database management system (Postgres) and/or an open-source software program adding support to that system (PostGIS). (For more information on schemas, visit <http://schemas.usgin.org/models/>.)

**Installation and Set-Up**

Install GeoServer and Postgres/PostGIS together by getting OpenGeo Suite. Download the OpenGeo Suite, community or paid editions, from OpenGeo (<http://opengeo.org/>). The OpenGeo Suite not only includes Postgres and PostGIS but also includes GeoServer, OpenLayers and other packages, configured to work together. Also install the pgAdmin III administration tool. This is an incredibly useful tool as it provides the interface for creating new databases, tables, views, and users, execute SQL statements, modify user permissions, and much more. Although these tasks can still be accomplished via the psql command shell, pgAdmin III makes life much easier.

2.3.1 GeoServer Installation

GeoServer is built on the Java framework and therefore supported on many different platforms, such as Linux, Mac OS X and Windows. The two primary ways to install GeoServer on Windows are: 1) by launching the Windows exe executable file, or 2) by deploying the web archive (war) file. Java applications such as GeoServer need to be run through a java servlet engine. The Windows exe installer will install the Jetty servlet engine and create a Windows Service for automatic startup; the war file must be deployed in a previously existing servlet engine.

The latest version of GeoServer can be found at <http://geoserver.org/display/GEOS/Download>. It is recommended to install the latest stable version. No GeoServer extensions/plugins are required. Installing GeoServer using the Windows exe installer is a straight-forward process: download the latest stable version of GeoServer and launch the executable file which starts a GeoServer set-up wizard. Once GeoServer is installed and the service running, point your web browser to http://localhost:8080/geoserver/ to confirm the installation. Log in using the username and password set during installation. The default username and password is admin/geoserver. After installation, be sure to resolve any notices about security related issues that are displayed on the your local GeoServer homepage. Then, click Contact Information to specify the service provider information that will serve as your metadata contact information in service requests.

**Deploying Services**

The following outlines how to use OpenGeo Suite on Windows platform to deploy WMS/WFS services, using a connection to PostGIS database tables as an example.

1. Using the OpenGeo Suite Dashboard, click Start and choose GeoServer to open the GeoServer interface.
2. Under Workspaces, click Add new workspace. The workspace specifies a namespace, so each data type will need its own separate workspace.
   1. Choose a name to identify your workspace (i.e., ThermalSprings)
   2. Fill in the namespace, given in the first line of an NGDS schema, or XSD, (<http://schemas.usgin.org/models/>)
   3. Click Submit
3. Under Stores, click Add new Store. The Store will house the connection parameters to the data table in PostGIS database.
   1. If using PostGIS, choose this from the Vector Data Sources list. Choose the workspace you’ve just created from the drop-down list. The Data Source Name\* will be the name of the data table as it appears exactly in your PostGIS database. (Using pgAdmin III administration tool is especially helpful here. When you understand the file structure and exactly where the data tables are in your PostGIS database, you will be better able to input the correct parameters to connect to that database in the Store page on Gerserver. It will also allow you to see the exact name and capitalization of a given data table.)
   2. Be sure to click the box next to “Enabled”
   3. Once connection parameters are filled in, click Save.
4. Under Layers, click Add a new resource.
   1. From the drop-down list, choose the workspace:Store you’ve just created.
   2. If connected to PostGIS database, this should give all the available tables in the specified folder. Navigate through the list to the data table (i.e., azthermalsprings) and click Publish to the right of that data table name.
   3. This takes you to Edit Layer. Keep the lower-case name.
   4. In the Data tab, under Bounding Boxes, choose ‘Compute from data’ under ‘Native Bounding Box; choose ‘Compute from native bounds’ under ‘Lat/Lon Bounding Box’. This will pull locational bounding boxes from the data table.
   5. Click Save at bottom.
5. Under Layers, click Add a new resource.
   1. Choose the workspace:Store you created previously.
   2. When the available data tables pop up, there is an bolded option at the top labeled ‘Configure new SQL view...’ Click this.
      1. We’ve found that when connecting to a PostGIS database, GeoServer will not recognize the correct SRID (Spatial Reference System Identifier). This needs to be indicated manually by creating a new SQL view of the layer. The SQL view will serve as the final layer for your published service.
   3. In the Create new SQL view page, fill in the ‘View Name’ that will serve as the final layer name, as indicated by the appropriate schema.
   4. In ‘SQL statement’ use a SQL statement that will call all the fields from the desired data table as such:

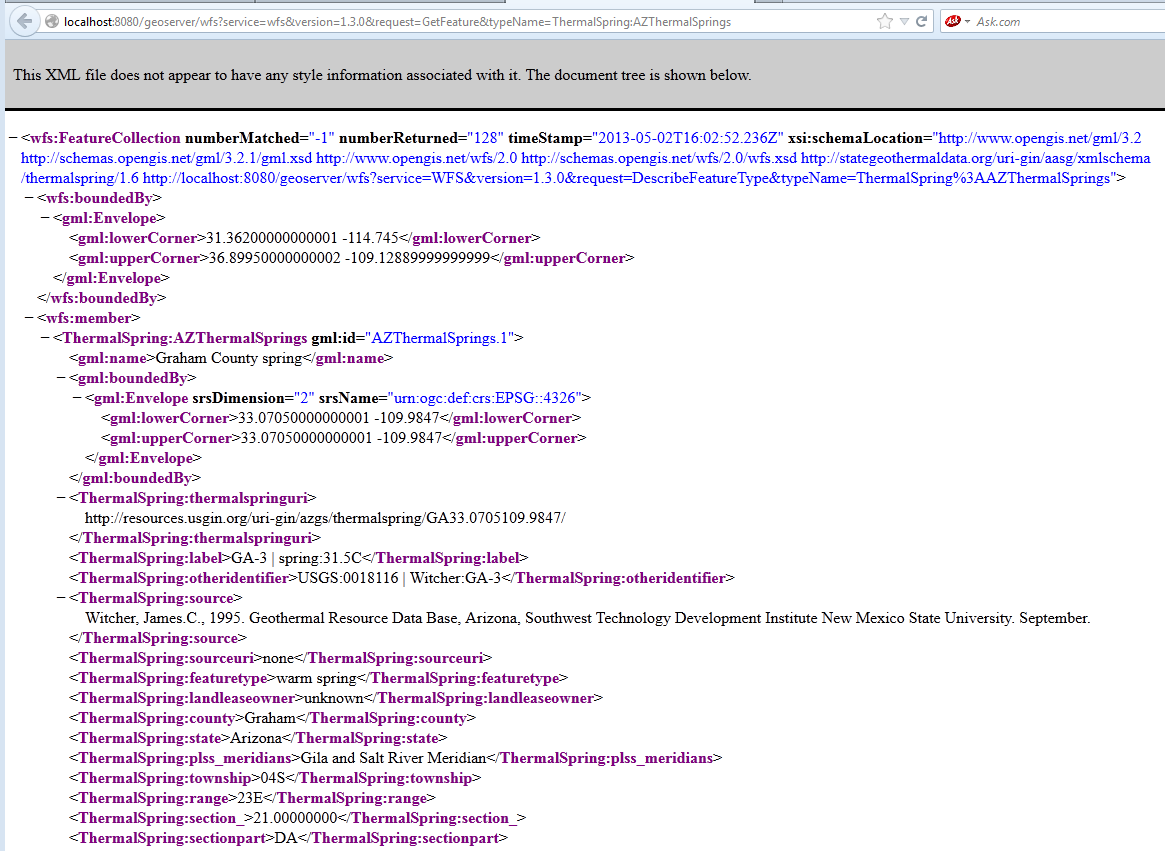
select \* from azthermalsprings

Click Refresh.

* 1. When the fields appear, check objectid as the Identifier. For the shape field, type is indicated as Geometry. The SRID box must be filled in with the correct EPSG code (i.e., 4326). Click Save.
  2. This takes you to Edit Layer.
  3. In the Data tab, under Bounding Boxes, choose ‘Compute from data’ under ‘Native Bounding Box; choose ‘Compute from native bounds’ under ‘Lat/Lon Bounding Box’. This will pull locational bounding boxes from the data table.
  4. In the publishing tab, under WMS Settings, a Style (or symbology) may be chosen.
     1. To add a Style: under Styles from GeoServer home, click Add a new style.
     2. Choose a name and the workspace for this style (i.e., ThermalSprings)
     3. To use the SLDs for the NGDS project, scroll down to SLD file and click Browse. Browse to your downloaded SLD copies, avaialbe at <http://repository.stategeothermaldata.org/repository/resource/50ec3aefb656b70647f32e38bc1b7479/>
     4. Click Upload. Click Submit.
  5. Click Save.

1. Perform a GetFeature request to confirm data return. Construct a GetFeature URL, where the typeName=workspace:layer. For example:

[http://localhost:8080/geoserver/wfs?service=wfs&version=1.3.0&request=GetFeature&typeName=ThermalSpring:ThermalSpring](http://localhost:8080/geoserver/wfs?service=wfs&version=1.3.0&request=GetFeature&typeName=ThermalSpring:AZThermalSprings)



**Application Schemas**

One hurdle to using GeoServer to deploy NGDS Geothermal Data services is the resolution of schemas. Creating additional configuration files for GeoServer allows the use of "app-schemas" which will automatically download the indicated web-accessible schema (.xsd) and be referenced in the GetFeature request. This solves a problem with service deployment using ArcGIS, where custom schema locations is necessary. The following documentation is based on Tomcat running GeoServer, set up on an Oracle VM VirtualBox. The data for the services are pulled from a PostGIS database. Likely, the configuration files will be the same if running GeoServer on Windows platform

*Install the App-Schema .jar files*

See this page <http://sourceforge.net/projects/geoserver/files/GeoServer%20Extensions/> for app-schema versions and downloads.

* Download the app-schema plugin zip file for the same version of your GeoServer instance.
* 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.

*Configuration files*

App-Schema GeoServer documentation is available on thier site here: <http://docs.geoserver.org/stable/en/user/data/app-schema/index.html>.

Two configuration files are necessary for app-schemas, one of which will have to be manually created. The mapping-file.xml file will be created, and the datastore.xml will be edited. GeoServer automatically creates the datastore.xml file, it just needs to be edited to specify that the layer will use app-schema configuration.

**Mapping file**:

This file will contain the database connection information, the web address to the schema (.xsd), and the TypeMapping which correlates the fields in your database table to those in your schema. Please see these two blogs for FeatureTypeMapping examples: <http://lab.usgin.org/node/116> and <http://lab.usgin.org/groups/building-geosciml-wfs-server/ncgmp-geosciml-mappedfeature>. See GeoServer documentation here: <http://docs.geoserver.org/stable/en/user/data/app-schema/mapping-file.html>

Example mapping file: [https://github.com/ccaudill/](https://github.com/ccaudill/GeoserverWorkspace/archive/4.1.2013.zip)[GeoServer](https://www.google.com/url?q=https%3A%2F%2Fgithub.com%2Fccaudill%2FGeoserverWorkspace%2Farchive%2F4.1.2013.zip&sa=D&sntz=1&usg=AFQjCNEtyK3gBGxHp-b9NT0CKPNmWHbkkA)[Workspace/archive/4.1.2013.zip](https://github.com/ccaudill/GeoserverWorkspace/archive/4.1.2013.zip) under folder hf - azheatflow - HeatFlow.xml

NOTE: The name of your layer **must** match exactly the name of the targetElement in your mapping file, which is the element indicated in the schema. Thus, the layer name in GeoServer used for app-schema configuration must match the name specified in your schema document.

**datastore.xml file**:

The datastore.xml file points to the file location of the mapping file and also indicates that an app-schema is to be used. See GeoServer documentation here: <http://docs.geoserver.org/stable/en/user/data/app-schema/data-stores.html#app-schema-data-stores>

Example mapping file: [https://github.com/ccaudill/](https://github.com/ccaudill/GeoserverWorkspace/archive/4.1.2013.zip)[GeoServer](https://www.google.com/url?q=https%3A%2F%2Fgithub.com%2Fccaudill%2FGeoserverWorkspace%2Farchive%2F4.1.2013.zip&sa=D&sntz=1&usg=AFQjCNEtyK3gBGxHp-b9NT0CKPNmWHbkkA)[Workspace/archive/4.1.2013.zip](https://github.com/ccaudill/GeoserverWorkspace/archive/4.1.2013.zip) under folder hf - azheatflow - datastore.xml

* The datastore.xml file **must** keep the letter and number string that geoserver gives as its internal id in the datastore id and workspace id fields. (Try to rename this as something human-readable, and GeoServer will not be able to open when restarting.)
* As you remove the type=PostGIS and enabled=true from the datastore.xml, the layer no longer shows as available, and now points to the app-schema. When the type field under dataStore is left as "PostGIS", the store will try to look for kyanite, no matter what else is in the file.Remove type field completely and indicate under connectionParameters dbtype=app-schema and store will look at the app-schema.

*File structure*

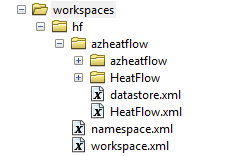
Simply insert these configuration files into the correct location in your GeoServer file structure and GeoServer should automatically download the schemas, pulling those schemas from a cache each time a GetFeature request is performed on the given service.

See this blog <http://lab.usgin.org/node/117> for an example GeoServer directory structure. The following example of the file structure begins with the workspaces folder.

In **OpenGeo Suite GeoServer on Windows platform**, the folder path to the workspaces folder was: C:\Program Files (x86)\OpenGeo\OpenGeo Suite\webapps\geoserver\data\workspaces.

In the **Oracle VM VirtualBox on Linux platform**, the folder path to the workspaces folder was: /var/lib/tomcat6/webapps/geoserver/data/workspaces. In the following example,

1. hf is the name of the Workspace (created in the GeoServer interface)
2. azheatflow is the name of the Store (created in the GeoServer interface)
3. HeatFlow is the name of the Layer (SQL view was created in the GeoServer interface)
4. HeatFlow.xml is the name of the mapping file (stored in the same location as the datastore.xml file)



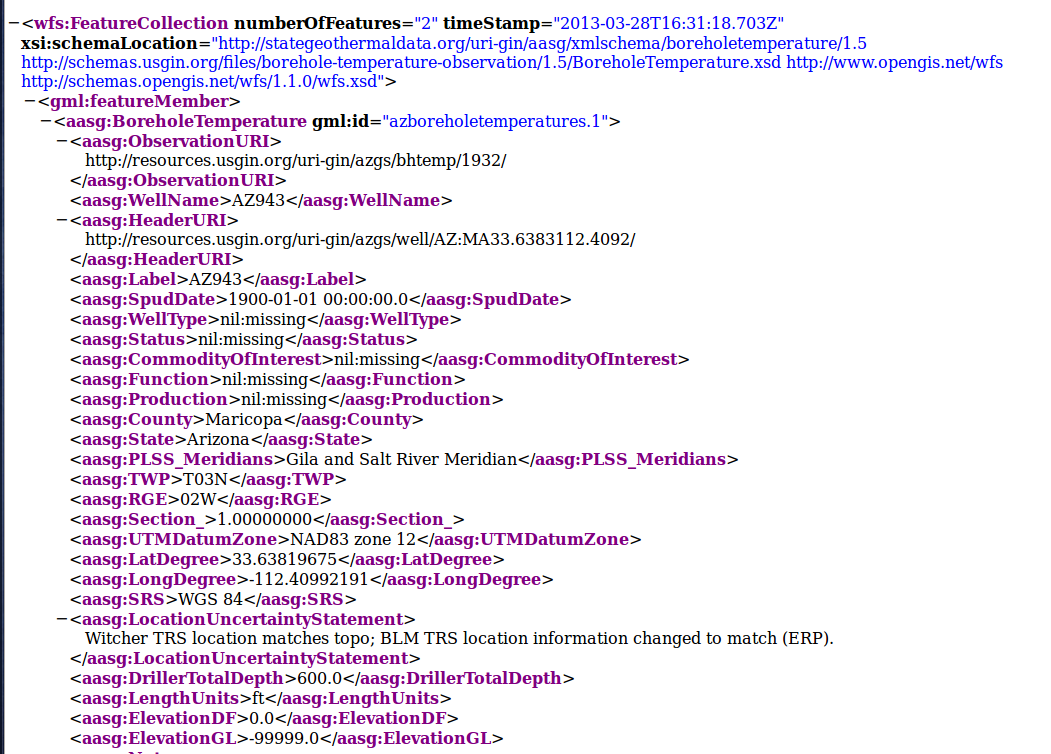
It is not necessary to edit the namespace.xml or workspace.xml files, nor the files that are in the HeatFlow folder (which are the featuretype.xml and layer.xml files).

One other important file location is the cache of the schemas, which GeoServer creates. In the Oracle VM VirtualBox on Linux platform, the folder path to the schema cache location is /var/lib/tomcat6/webapps/geoserver/data/app-schema-cache. In OpenGeo Suite GeoServer on Windows platform, the folder path to the schema cache location is C:\Program Files (x86)\OpenGeo\OpenGeo Suite\webapps\geoserver\data\app-schema-cache. In these locations, the filing of the downloaded schemas is a little convoluted. For example, the online link to the heat flow schema is <http://schemas.usgin.org/files/heat-flow/1.23/HeatFlow.xsd>. In GeoServer, this schema is filed in as series of folders from the app-schema-cache folder as such: /org/usgin/schemas/files/heat-flow/1.23/HeatFlow.xsd

Making the final location of the schema in Linux:

/var/lib/tomcat6/webapps/geoserver/data/app-schema-cache/org/usgin/schemas/files/heat-flow/1.23/HeatFlow.xsd

*GetFeature request example using app-schema configuration*



*App-schema errors and resolution*

"**java.lang.IllegalArgumentException: Filter Function problem for function strConcat argument #1 - expected type String" when preforming a GetFeature request.**"

This error resulted from the first element in AttributeMapping in the mapping file, where an OCQL expression was not accepted. We attempted using a OCQL statement here because the GetFeature returned an arbitrary string as the gml:id instead of the OBJECTID. This was because the OBJECTID field in our PostGIS data table was not specified as the primary key. Once this field was specified as such, GeoServer could read this as the gml:id by the appropriate expression for the element in the mapping file:

<AttributeMapping>

<targetAttribute>aasg:HeatFlow</targetAttribute>

<idExpression><OCQL>getId()</OCQL></idExpression>

</AttributeMapping>

According to GeoServer developers, "For performance reasons, we strongly recommend avoiding using CQL to generate gml:id in the mapping file as this prevents the use of database indices and the generation of efficient SQL queries."

It must be noted that one important caveat to creating tables in any database structure using ArcMap is that a primary key is **not** indicated. We found this when the return of a GetFeature request in GeoServer using app-schemas showed not only an arbitrary string for the gml:id, but the data was not correctly mapped to the fields (or columns from the PostGIS data table). It appeared that the first column, the OBJECTID, was acutally being mapped as data and therefore put into the first column from the mapping file, which was ObservationURI. This caused more choas; as the schema was being read by GeoServer with a one-field offset, it misinterpreted some fields as not required and completely left them out of the GetFeature request. To fix this, we used pgAdmin III to specify the primary key as the objectid in the table. Then, GeoServer was able to acknowledge this and leave that field's integer out of the data when the GetFeature was returned, thus giving a correct return of the data from that PostGIS table. For those using pgAdmin III, specify a primary key by right-clicking the PostGIS database table and selecting "Properties". Under the "Constraints" tab, click the Add button beside "Primary Key". Give a name, such as "bht\_p\_key" then click the "Columns" tab to choose the appropriate field (here, the objectid). Click OK.

"**java.lang.VerifyError: Cannot inherit from final class**"

Be certain that the version of the app-schema download is the same version is the same version of GeoServer you're using. Those specific .jar files in the app-schema plug-in are essential. Using all the same versions cleared this error. See this page <http://sourceforge.net/projects/geoserver/files/GeoServer%20Extensions/> for app-schema versions and downloads.

"**Can't parse complex datastore config.**"

This error was resolved with an awful lot of tweaking the mapping file. See the above for a polished example.

"**org.xml.sax.SAXParseException: Content is not allowed in prolog.**"

This error was caused by characters that were unintentionally added to the first line of the mapping file. This had two causes - one was the copy and paste method used to create the file, and the second was the use of XML Copy Editor. This free XML editor evidently inserts it's own code into the file. Every time I opened and saved the file with this software, I would get this error. Removing characters in the mapping file that were before the first line ( i.e., <?xml version="1.0" encoding="UTF-8"?>) and using another free editor solved these problems.

*Helpful housekeeping tips*

**catalina.sh**

Is GeoServer thinking forever to take you to the Layers page or on a request, or not able to return a request at all? Try editing the **catalina.sh** file in the GeoServer data directory to indicate that more of Tomcat's memory needs to be allocated to running GeoServer. To this file, add:

JAVA\_OPTS="-Djava.awt.headless=true -Dfile.encoding=UTF-8 -server -Xms512m -Xmx1024m -XX:NewSize=256m -XX:MaxNewSize=256m -XX:PermSize=256m -XX:MaxPermSize=256m -XX:+DisableExplicitGC"

So that a portion of the file will now look like this:

# JSSE\_HOME (Optional) May point at your Java Secure Sockets Extension

# (JSSE) installation, whose JAR files will be added to the

# system class path used to start Tomcat.

#

# CATALINA\_PID (Optional) Path of the file which should contains the pid

# of catalina startup java process, when start (fork) is used

#

# $Id: catalina.sh 609438 2008-01-06 22:14:28Z markt $

# -----------------------------------------------------------------------------

JAVA\_OPTS="-Djava.awt.headless=true -Dfile.encoding=UTF-8 -server -Xms512m -Xmx1024m -XX:NewSize=256m -XX:MaxNewSize=256m -XX:PermSize=256m -XX:MaxPermSize=256m -XX:+DisableExplicitGC"

# OS specific support. $var \_must\_ be set to either true or false.

cygwin=false

os400=false

darwin=false