
GeoMOOSE Documentation

Release 2.7.1

Dan "Ducky" Little

July 20, 2015

CONTENTS

1	Getting Started with GeoMOOSE	1
1.1	GeoMoose Feature Overview	1
1.2	GeoMoose User Quickstart	2
1.3	Try the online Demo	10
1.4	Install It!	10
1.5	Instructional Materials	24
1.6	Developer Specific Documentation	24
2	Developer Documentation	25
2.1	config.js Configuration Options	25
2.2	Understanding GeoMOOSE Services	28
2.3	Writing a GeoMOOSE (JavaScript) Extension	32
2.4	GeoMOOSE Sprite Generator, Toolbar and Catalog Styles	36
2.5	How To Build GeoMOOSE from source	37
2.6	API Docs	38
3	GeoMOOSE License	39
3.1	License Summary	39
3.2	License Text	39
4	Implementer Documentation	41
4.1	Configuration References	41
4.2	Vector Layer Editing	69
4.3	Service Documentation	74
4.4	Extensions Documentation	82
4.5	Helpful How To's	86

GETTING STARTED WITH GEOMOOSE



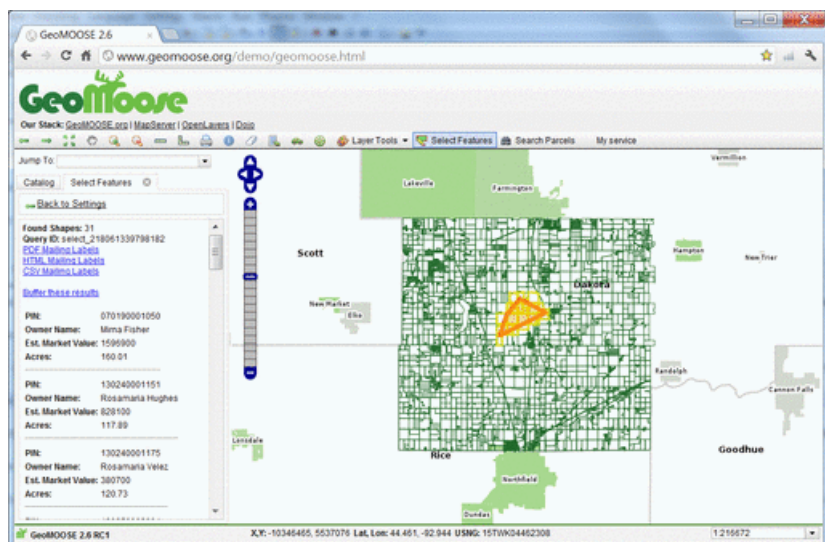
1.1 GeoMoose Feature Overview

1.1.1 Web GIS Portal

GeoMOOSE is a browser based mapping framework for displaying distributed cartographic data. It is particularly useful for managing spatial and non-spatial data within county, city and municipal offices (from which GeoMoose originated). It extends the functionality of MapServer and OpenLayers to provide built in services, like drill-down identify operations for viewing and organising many layers, selection operations and dataset searches.

GeoMOOSE is fast, performing well with hundreds of layers and/or services at a time. Data from multiple custodians can be maintained with different tools and on different schedules as each map layer has it's own set of configuration files for publishing, symbols, templates as well as source data.

The user interface is easily configurable, and additional services can be added through a modular architecture.



Core Features

- Distributed data maintenance amongst multiple owners.
- Access maps from: MapServer, Google, VirtualEarth, Tilecache, ArcGIS REST, WMS.
- Configure multiple views of data sources.
- Discover and filter from data catalogs.
- Tools: measure, drawing, query, fading, re-order, reprojection, jump-to zoom, coordinate readouts, ...
- Displays: MapViewer, Bird's Eye, Side Menu, Navigation, Tabbed User Controls.
- XML based MapBook configuration file for User Interface, Source Layers and Tools.
- Integration with Mapserver.
- Modular design facilitates integration with non-spatial systems (such as asset management).
- Publish almost unlimited number of layers.
- PDF printing.

Implemented Standards

- WMS
- WFS (client)
- WFS-T (client)

Details

Website: <http://www.geomoose.org/>

Licence: MIT based license. <http://www.geomoose.org/info/license.html>

Software Version: 2.6

Supported Platforms: Windows, Linux, Mac

Commercial Support: http://www.geomoose.org/info/commercial_support.html

Community Support: http://www.geomoose.org/info/mailling_lists.html

Quickstart

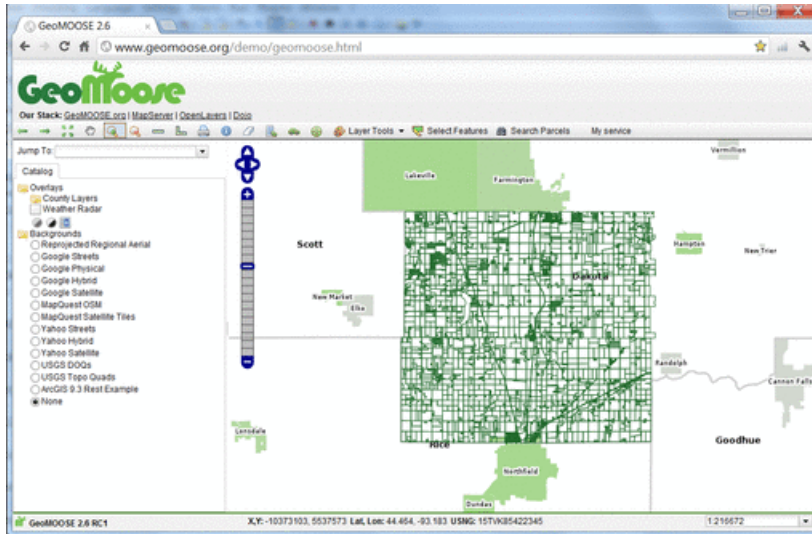
- *[Quickstart documentation](#)*

1.2 GeoMoose User Quickstart

1.2.1 Getting Started

Geomoose is a GIS data portal management framework. The basic capabilities of GeoMoose are contained in the on-line demo <http://www.geomoose.org/demo/geomoose.html> which may be downloaded as part of the standard installation (<http://www.geomoose.org/demo/geomoose.html>).

When you start GeoMoose the mapping environment will appear.

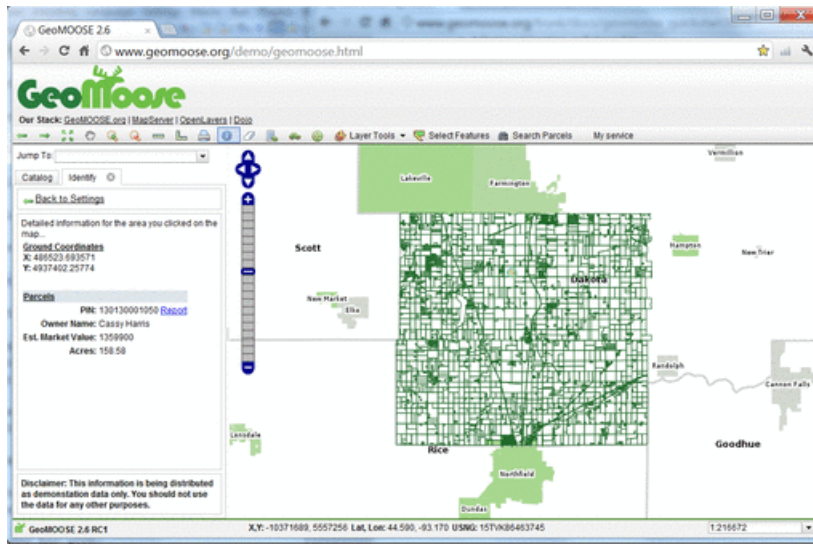


The Interfaces presented above shows:

1. A banner bar
2. A tool bar
3. A map window with
 - A navigation and zoom controls
4. A side menu with:
 - A “Jump to” location pull down list
 - Control tabs starting with “Catalog”, for displaying the layer list. As needed, additional tabs will appear: an “Information” readout, a “Custom” output tab, and others.
5. A Footer bar with:
 - Multiple coordinate readouts, one each for: X,Y (local dataset coordinates), LAT/LON (Decimal) and United States National Grid (USNG)
 - A editable pull down list for a view scales to choose from or define.

1.2.2 IDing a location in GeoMoose

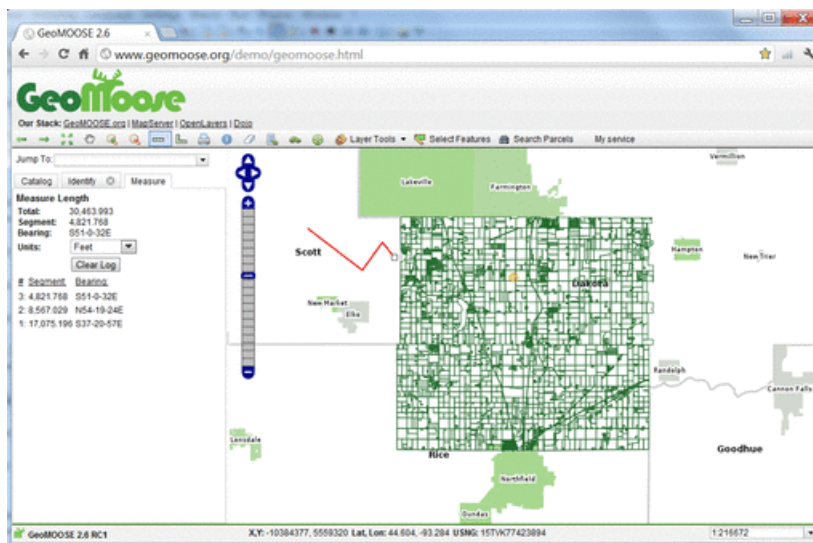
You can ID a location in the interface by clicking on the “identify”, (“i” in a blue circle icon) button in the top toolbar, and then clicking a point on the map to identify.



The menu along the left side of the Map view will display a report related to the point that you clicked in the map. This is a service in the GeoMoose Demo package that has been configured to respond to queries for the “Parcel” layer when a point is clicked.

1.2.3 Measuring in GeoMoose

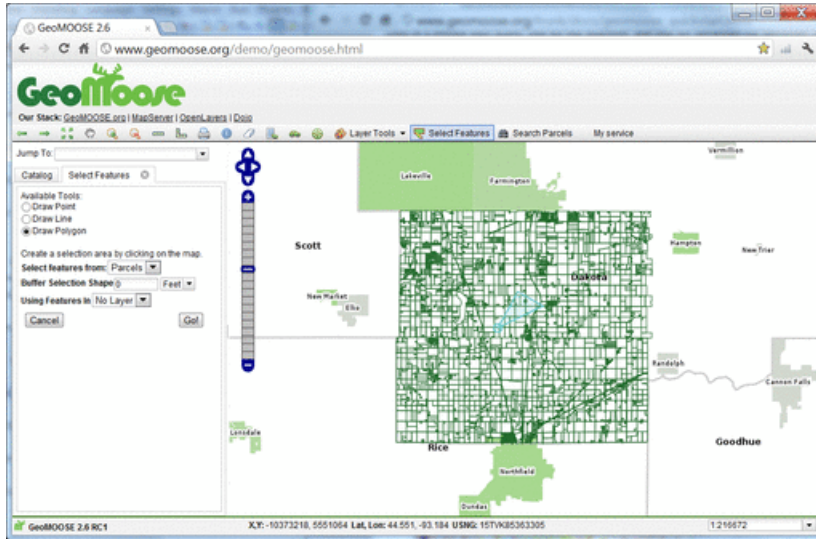
GeoMoose has two measuring tools installed and activated by default. Clicking the straight ruler icon in the top toolbar, will start the linear measuring tool. You can click as many points as you like in the map window and a trailing polyline will be drawn. When you get to your last point, just double click to stop. The total distance of all lines drawn will be used to generate the “Total Length” in the units of your choice in the side menu. Clicking the other ruler icon in the top toolbar, will allow you measure areas.



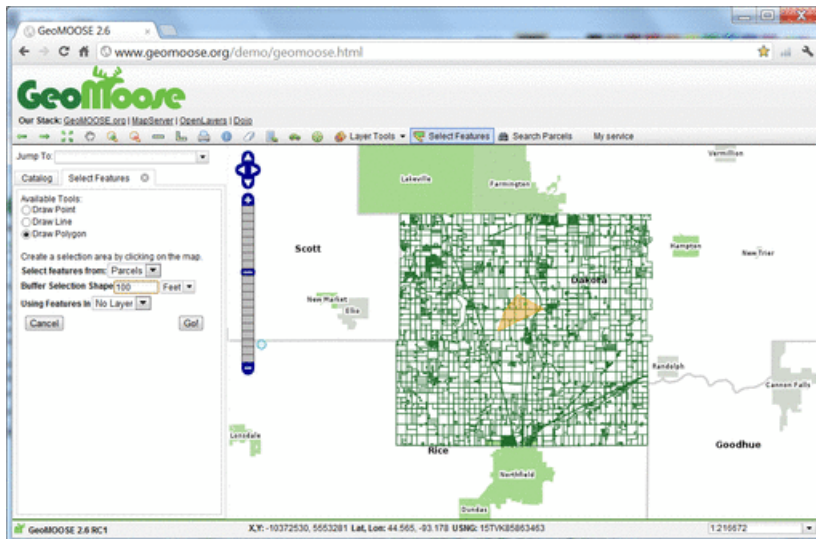
These user tools are considered services by the GeoMoose interface and can be added to via GeoMoose’s MAPBOOK control file.

1.2.4 Selecting Features

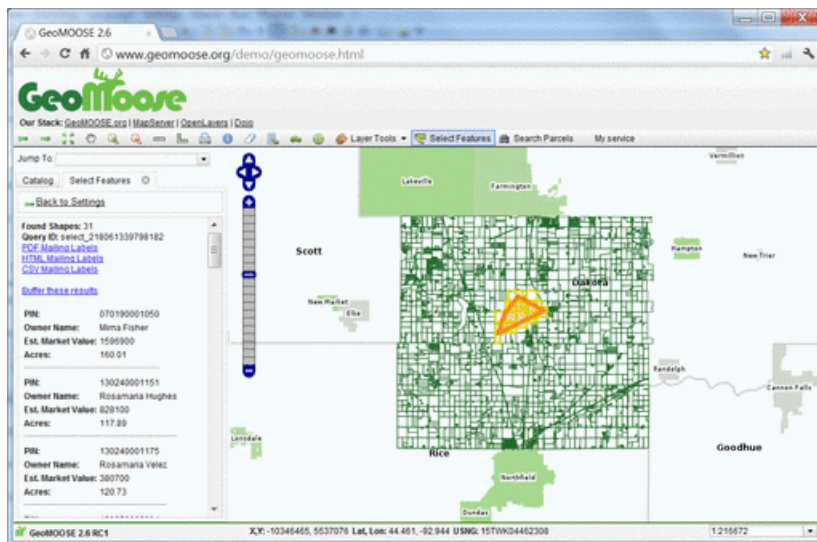
You can select features from a Point, Line, polygon, or a Box. To start, click on the “Select Features” icon (polygon with pencil), and start drawing a selection polygon in the map. You can double click the last point to finish the drawing process. This screenshot shows the drawing process on the map. the left menu is displaying the input form for the criteria of the selection, in this case a Polygon against the “Parcels” layers. You also have the option to select features based on a buffer which is a “0” value by default. This is a service in the GeoMoose Demo package that has been configured to respond to queries for the “Parcel” layer.



The above screenshot is showing the “PARCEL” layer for selects and also using the “Parcels” layer as the attributes to report back on. Two different layers can be used in a double pass query, one for the selection, and one for retrieving the attributes from for reporting. The screenshot below takes the selection process further by adding in a buffering distance of 100ft.



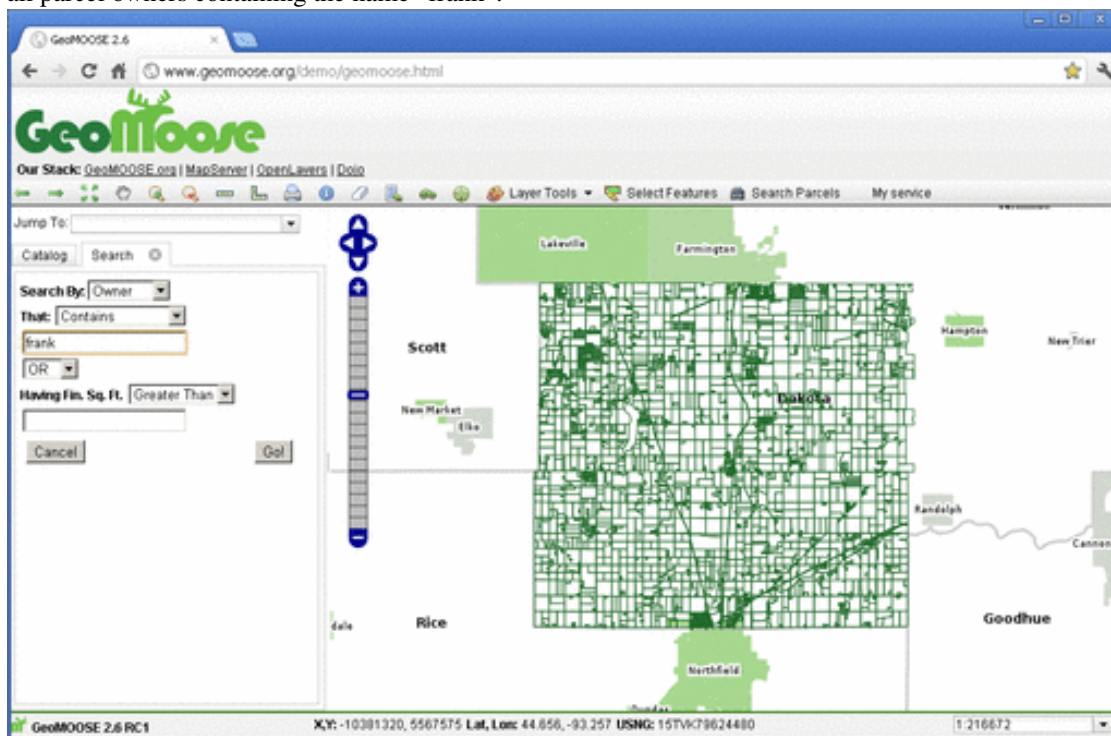
Clicking the “Go!” button in the side menu in the above screenshot will generate a report and display it in the side menu. NOTE: The output can also be configured to be sent to a new window as well.



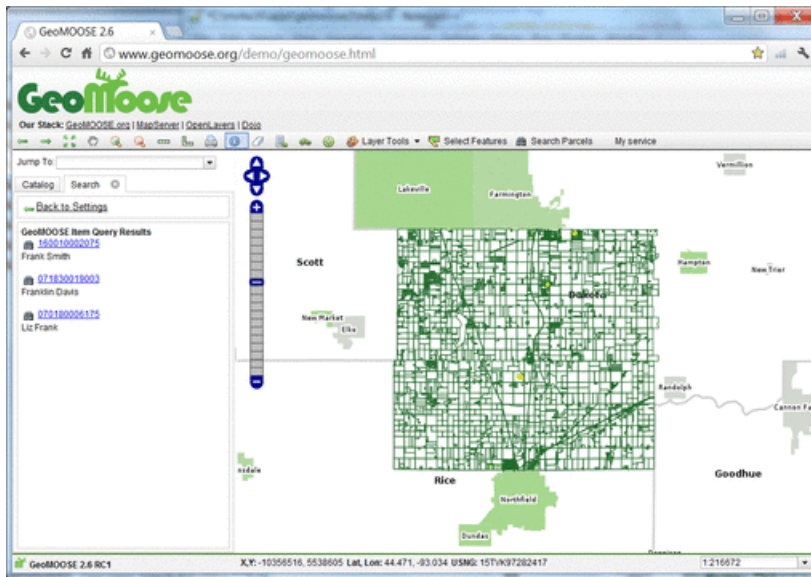
The results above demonstrate the service response to the buffered query by displaying the result in the map via a selection overlay that highlights the original selection area polygon (purple), the crossing and within parcels at a 100ft buffer (orange) and the parcels crossing and within the buffered polygon (yellow). The side menu now displays the reported results of the query process with options for outputting in other formats for mailing labels.

1.2.5 Searching

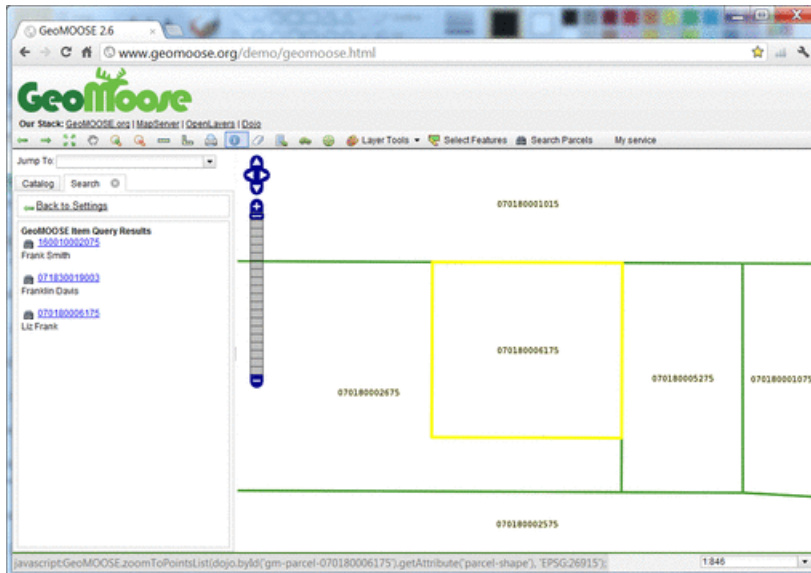
You can also search for features by attribute. To start click on the “Search Parcels” icon and enter your search parameters on the search menu then pressing the “Go!” button. The following screenshot demonstrates a search of all parcel owners containing the name “frank”.



Three results appear in the list and are highlighted on the map.

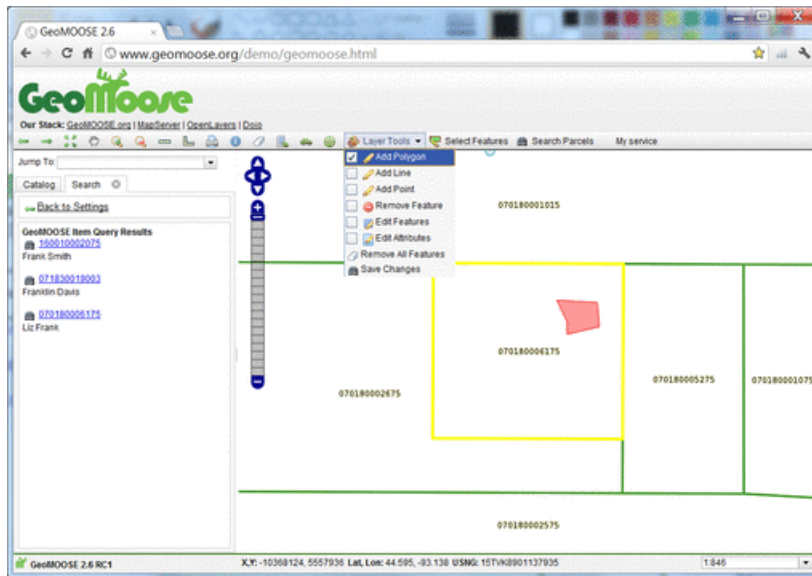


Clicking the binocular icon/parcel number in blue will zoom you to the specific parcel as illustrated in the following screenshot.



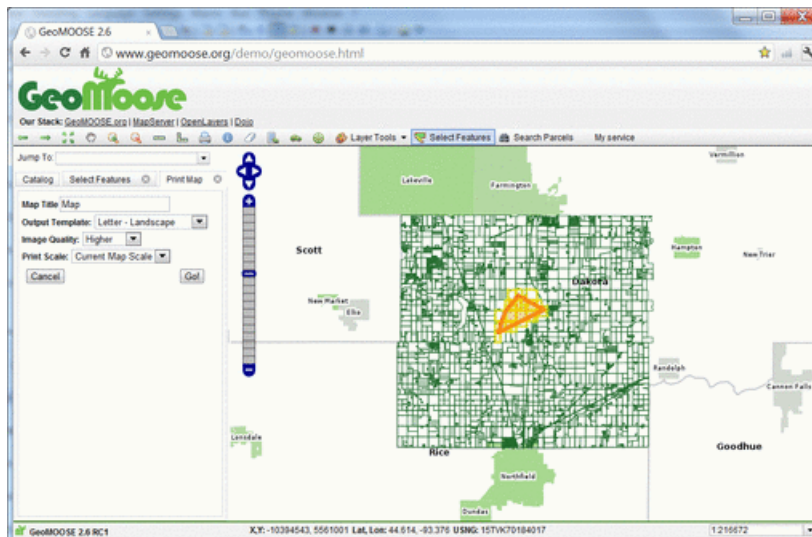
1.2.6 Layer Tools

GeoMoose lets you edit layers graphically. Setting up a layer for editing is discussed elsewhere on the GeoMoose website. The demo is setup for you to edit a sketch layer as illustrated in the following screenshot.

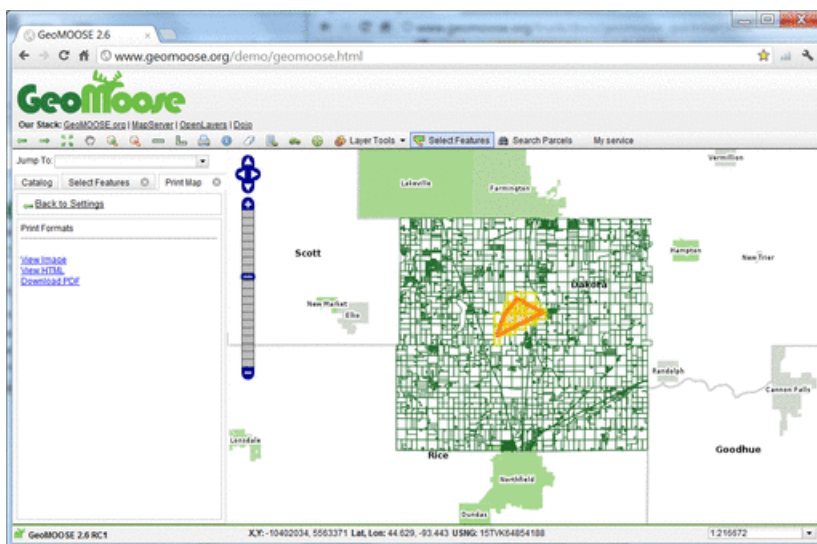


1.2.7 Printing

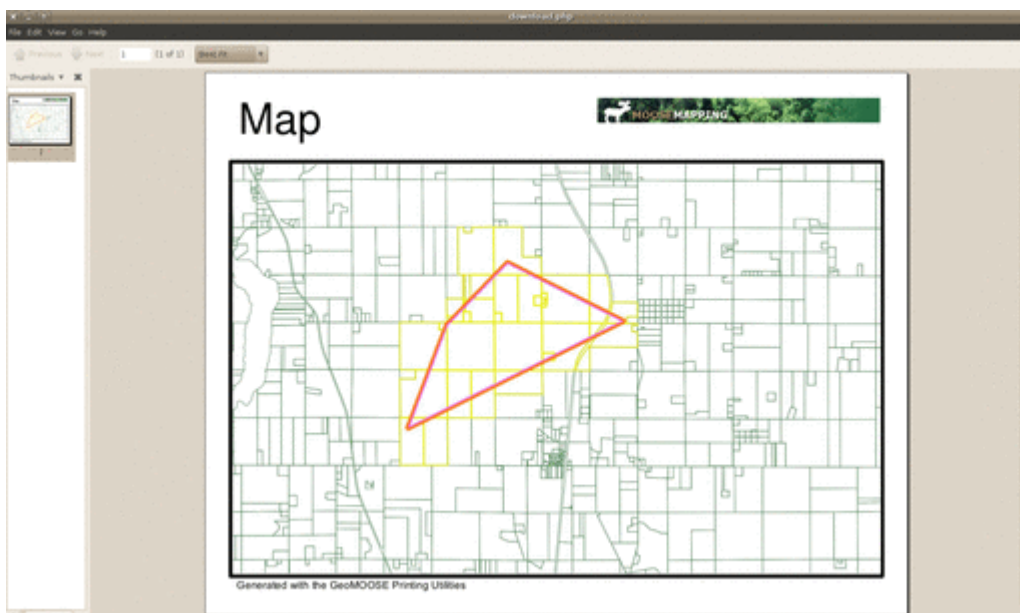
This next screenshot shows what the Print Map service looks as provided with demo for GeoMoose. The side menu in the following view, shows the options available for printing the current mapview. Sheet size, orientation, resolution for both raster image backgrounds and for overall output are also available.



The next menu displays the choices available for output, by default GeoMoose allows a composite Raster Image, a HTML file, or a PDF. Clicking on a PDF output option will present you with a dialog similar to:

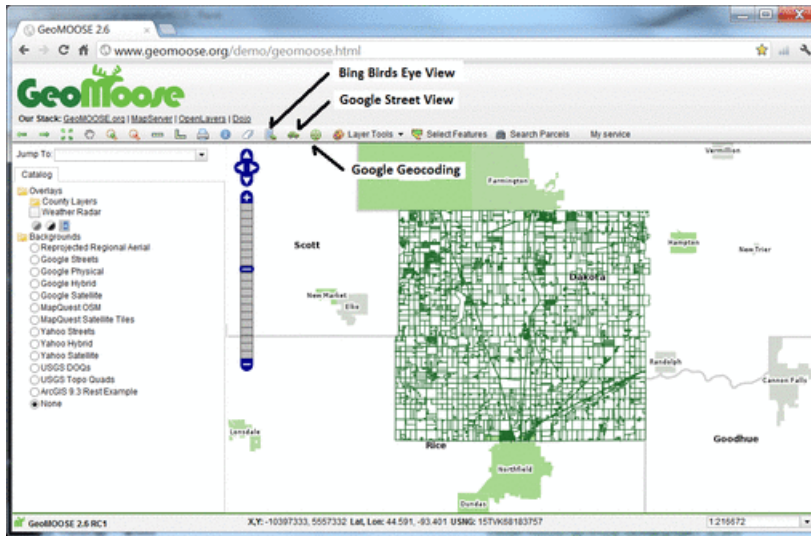


The following screenshot shows an example of a PDF output in landscape mode.



1.2.8 Other Services

GeoMoose can also link to other on-line services such as birds eye view from BING, StreetView from Google and Geocoding from Google. Always remember to review license requirements for any external services to ensure compliance.



1.3 Try the online Demo

If you have not already, looking through the [GeoMOOSE Demo](#) can give you a good idea of how GeoMOOSE looks.

1.4 Install It!

GeoMOOSE is very easy to install on just about any platform. Just follow any of these guides:

1.4.1 Installation UNIX

Preparation

MapServer

GeoMOOSE requires a pretty full feature MapServer build. The following options are used by the development team on UN*X based systems to build MapServer:

```
--with-php=[path to php includes]
--with-agg=[path to agg libs]
--with-gd=[path to gd libs]
--with-wfs
--with-wmsclient
--with-wfsclient
--with-postgis
--with-proj=[path to proj]
--with-geos=[path to geos-config]/geos-config
--with-freetype=[path to freetype]
--with-gdal=[path to gdal-config]/gdal-config
--with-ogr
```

MapServer's PHP/Mapscript must also be installed. Please see the MapServer website for additional details on how to install and compile MapServer. You will also find information about the MapServer mailing lists which are very helpful for trouble shooting and additional instructions.

The current code has been tested against PHP version 5.3.3.

VERY IMPORTANT NOTE: Some Linux distributions come pre-packaged with a version of MapServer. This version of MapServer can be extremely out-dated and can be missing critical functionality (AGG rendering for example).

PHP Packages

GeoMOOSE requires a few PHP packages. As of PHP 5.3, PDO has been integrated into PHP's core and the PECL versions are out dated. On Debian/Ubuntu instances you'll need to add the following PHP packages to your system.:

```
apt-get install php5 php5-dev php5-gd php5-pgsql php5-sqlite php-pear php5-curl
```

Then add the following to your `php.ini`.

```
extension=php_mapscript.so
extension=gd.so
extension=curl.so
extension=pdo.so
extension=pdo_sqlite.so
extension=pdo_pgsql.so
```

Some package managers will automatically add these to your system. Check *phpinfo.php* if you are unsure if the packages have installed.

Installation

For this example we will use `/srv/geomoose` as the target directory. please adjust as appropriate for your local system.

Install the Code

If the Pre-req's have been met then the hard part is actually over. Extract the GeoMOOSE `.tar.gz` file to your target directory.

```
tar -xzf geomoose-[version].tar.gz
```

A Temporary Directory

GeoMOOSE requires a directory for caching WMS images and for writing temporary files for some of the services. By default it uses the directory `/tmp/out`. To create this directory, and make it useful we can do the following.:

```
mkdir /tmp/out
chown www-data:www-data /tmp/out
```

This example assumes the "Apache user" is `www-data` other systems will use `www`, `httpd`, or `apache`. Consult your systems `httpd.conf` for the exact answer.

Many recent Linux distributions clear the `/tmp` directory at startup so you may need to create a startup script to recreate the `/tmp/out` directory. Also, keep in mind that the `/tmp/out` directory can accumulate a large amount of files/disk space over time, so it is recommended that the directory is periodically cleaned. As an example, the following when run regularly via cron (e.g. every 5 minutes), could achieve both goals.:

```
#!/bin/bash
# WARNING: This script is provided as an example only.
#           Make sure you understand what this script does before you
```

```
#           use it. It is likely it will need to be tweaked for your
#           server's configuration.

#
# Create directories if they don't exist
#

# Temp space for MapServer (should match temp_directory.map)
mkdir -p /tmp/www
chmod 1777 /tmp/www

# Temp space for PHP (should match local_settings.ini)
mkdir -p /tmp/out
chmod 1777 /tmp/out

#
# Remove files that were last changed more than 60 minutes ago.
#

find /tmp/www -type f -mmin +60 -exec rm '{}' ';'
find /tmp/out -type f -mmin +60 -exec rm '{}' ;'
```

“local_settings.ini”

GeoMOOSE uses a small “.ini” file for configuring local paths. There is an example for MS4W and for this installation guide. Either copy *unix_local_settings.ini* to *local_settings.ini* or create a new *local_settings.ini* and add the following.

```
[paths]
root=/srv/geomoose/maps/
mapserver_url=/cgi-bin/mapserv
temp=/tmp/out/
```

An Apache Configuration File

Assuming Apache has been configured properly to run PHP the Apache configuration to run GeoMOOSE is very simple.

```
Alias /geomoose/ /srv/geomoose/htdocs/

<Location /geomoose/>
    AllowOverride None
    Options Indexes FollowSymLinks Multiviews
    Order allow,deny
    Allow from all
</Location>
```

Create a file called *geomoose.conf* in your appropriate Apache directory. Common places this are:

- */etc/apache2/conf.d/* - Debian/Ubuntu based systems.
- */etc/httpd/conf.d/* - RedHat Variants.
- */etc/apache2/other/* - Mac OS/X

Restart Apache and Test

At this point the GeoMOOSE demo should be installed. Restart Apache and navigate a web browser to *http://[servername]/geomoose/geomoose.html*.

1.4.2 Installation MS4W

VERY IMPORTANT NOTE There are some configuration changes between GeoMOOSE 2.4 and 2.6. Specifically the existence of *local_settings.ini* and the removal of some mapbook items. The supporting JavaScript has been overhauled as well. If you are upgrading your GeoMOOSE installation and have made configuration changes or added your own data to the maps folder, you should backup those files before proceeding. Once you have your MS4W folders backed up, you should delete these files and folders before proceeding.

Introduction

GeoMOOSE is a client side framework that is developed for use with MapServer (<http://www.mapserver.org>). While GeoMOOSE does not require MapServer to be installed when setting up a basic map viewer, many of the services included with GeoMOOSE (identify, query, search, etc.) do require MapServer. Most people will find it worthwhile to install MapServer and if you are going to install GeoMOOSE on a Windows-based computer, it is highly recommended that you use the MS4W package (MapServer for Windows). If you are installing GeoMOOSE on a Linux computer, see instructions at (http://geomoose.org/docs/install_unix.html).

In an effort to make it as simple as possible to get MapServer installed, we have packaged GeoMOOSE in an MapServer for Windows (MS4W) package. This document provides the steps necessary to download and install all the components required for GeoMOOSE.

IMPORTANT NOTE You need to have administrator level access to your computer to install and configure either Apache or IIS Server.

NOTE If you are new to web servers and developing web sites we **STRONGLY** recommend using Apache as your web server.

Installation

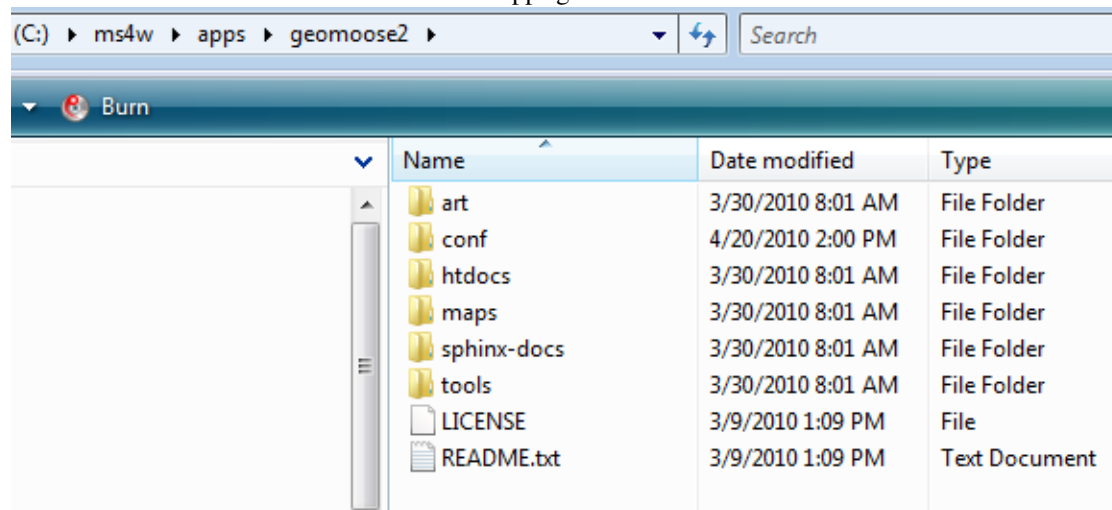
Step 1: MS4W Installation

IMPORTANT NOTE MS4W 3.0.4 (MapServer 6.0.2) + required.

- The MS4W download will install Apache, MapServer and PHP. Download the MS4W package from the Maptools web site at <http://www.maptools.org/ms4w>.
- There is not a traditional “install” program for MS4W, rather you are going to unzip the contents to the root of your C: drive. Actually, MS4W does not have to be installed on the C drive, but it is good practice to have the MS4W directory at the root of the drive (not in “Program Files”). You should not have multiple MS4W folders. (There now actually a traditional install .exe too)
- Once you have the MS4W package unzipped on your computer, you will need to decide which web server to use. The two most common choices are Apache and Microsoft IIS. Unless you need the integrated Windows authentication IIS provides or your organization requires the use of IIS, we recommend using the Apache web server that is included with MS4W.

Step 2: GeoMOOSE Installation

- Go to the GeoMOOSE download page (<http://geomoose.org/info/download.html>). There are two versions of GeoMOOSE for MS4W. “GeoMOOSE 2.2 for MS4W - Web Mercator Demo” is pre-configured with the setting necessary for Google, Bing, and OSM base maps; if you want to use one of these base maps, your map will have to be in the web Mercator projection. If you are not going to use one of these base maps, we recommend using the other “GeoMOOSE 2.6 for MS4W” download.
- The GeoMOOSE installation is similar to MS4W in that there is not a traditional “install” program. The only step is to unzip the files to the C:/MS4W directory. When finished, the GeoMOOSE installation should be in C:/ms4w/apps/geomoose2



Step 3a: Configuring Apache web server

NOTE Skip this step if you going to install IIS server in Step 3b.

- Apache comes pre-configured with MS4W and as long as it is installed at the root of the drive (i.e. “C:/ms4w”), all that is needed to complete the process is installing Apache as a Windows service. From Windows Explorer (or a command prompt), double-click on C:/ms4w/apache-install.bat. A command window will appear for a couple seconds and then disappear.
- To test if the installation is working, from a browser on the computer you installed MS4W and GeoMOOSE, type in the address <http://localhost>. Your screen should display information about MS4W and the applications installed like the image below.

MS4W - MapServer 4 Windows - version 2.3.1

Introduction

Welcome to MS4W v2.3.1, the MapServer package for Windows. This package is intended to simplify your life if you

- a novice MapServer user with little knowledge (or time) to compile MapServer from scratch.
- interested in one of the MS4W addon packages and looking to avoid configuration issues.
- any MapServer user who needs a quick and dirty installation on Windows.

In any of these cases, and probably more, MS4W can help you. However, a few notes about this package:

- This is not a true installer, it will not modify any file or registry setting outside of the MS4W directory.
- Since it is not a true installer, it will not install to any location. It **MUST** be installed at the root of one
- The MS4W addon packages must be extracted at the same root as the base installer, and files must be o

Features


MS4W contains default installations of Apache, PHP, MapServ, MapScript (CSharp, Java, PHP, Python) and some messing up the rest of the install. The base installer package comes pre-configured with the following software:


- [Apache 2.2.10](#) (with OpenSSL 0.9.8i)
- [PHP 5.2.6](#)
- mapserv 5.2.1 at [/cgi-bin/mapserv.exe](#)
- mapscript 5.2.1 flavours:
 - php_mapscript.dll ([phpinfo](#))
 - CSharp mapscript at `ms4w\Apache\cgi-bin\mapscript\csharp`
 - Java mapscript at `ms4w\Apache\cgi-bin\mapscript\java`
 - Python mapscript at `ms4w\Apache\cgi-bin\mapscript\python`
- mapserv utils at `ms4w\tools\mapserv\` which include:
 - legend.exe [MapServer utility documentation](#)
 - msencrypt.exe
 - scalebar.exe
 - shp2img.exe
 - shptree.exe (execute `ms4w\setenv.bat` before using mapserv utilities)
 - shptreevis.exe
 - sortshp.exe
 - tile4ms.exe
- At the bottom of this page is a list of applications installed which use MS4W. Click on the GeoMOOSE 2.6 link to test the installation.
- Further help and troubleshooting information is provided in the MS4W documentation (http://www.maptools.org/ms4w/index.phtml?page=README_INSTALL.html).

Step 3b: Configuring Microsoft IIS 6.0 web server

NOTE Skip this step if you installed Apache server in Step 3a. **NOTE** These instructions are for IIS 6.0 which is part of Windows Server 2003 and Windows XP Professional 64 bit. The steps are similar for other versions of IIS, however the interface can be very different and will not match screen shots provided here.

- Open the “IIS Manager”.
- Create a virtual directory called “geomoose2” and point it to the path “C:/ms4w/apps/geomoose2/htdocs”. See screen shots below for an example of adding a virtual directory.

Virtual Directory Creation Wizard 

Virtual Directory Alias 

Specify a short name, or alias, for this virtual directory.

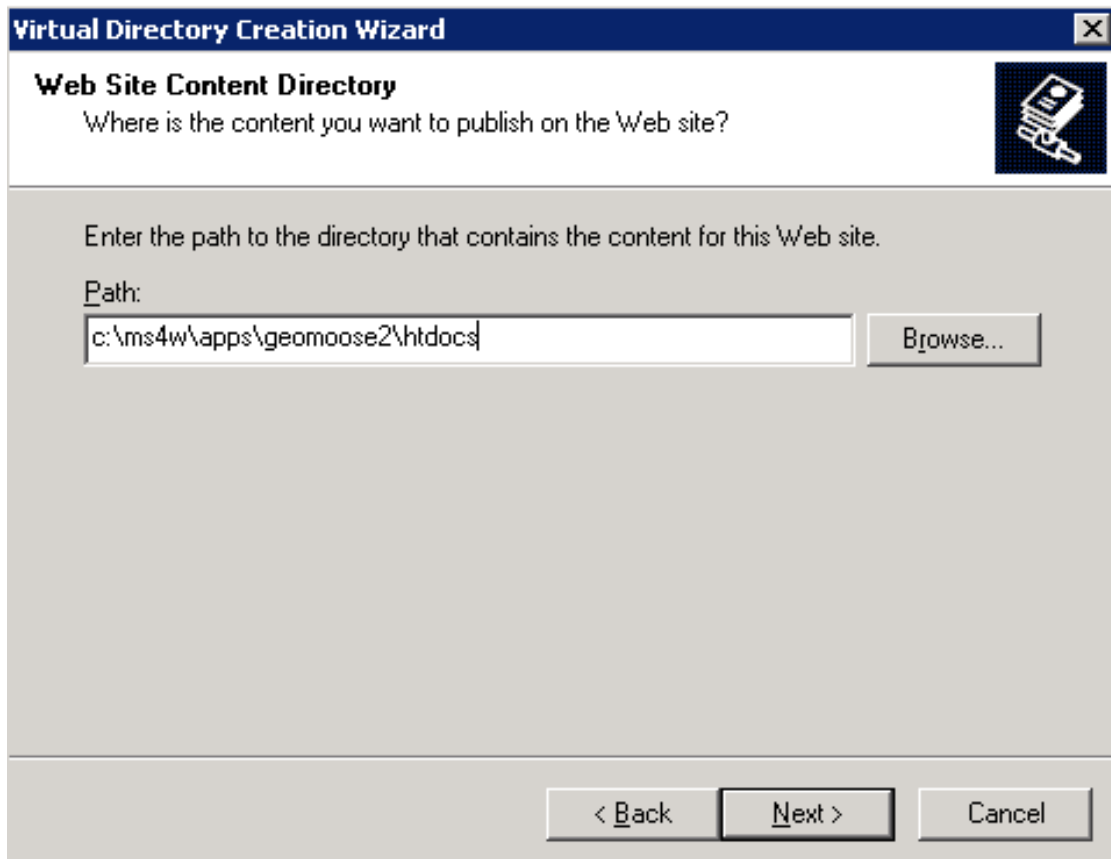
Type the alias you want to use to gain access to this Web virtual directory. Use the same naming conventions that you would for naming a directory.

Alias:

< Back

Next >

Cancel



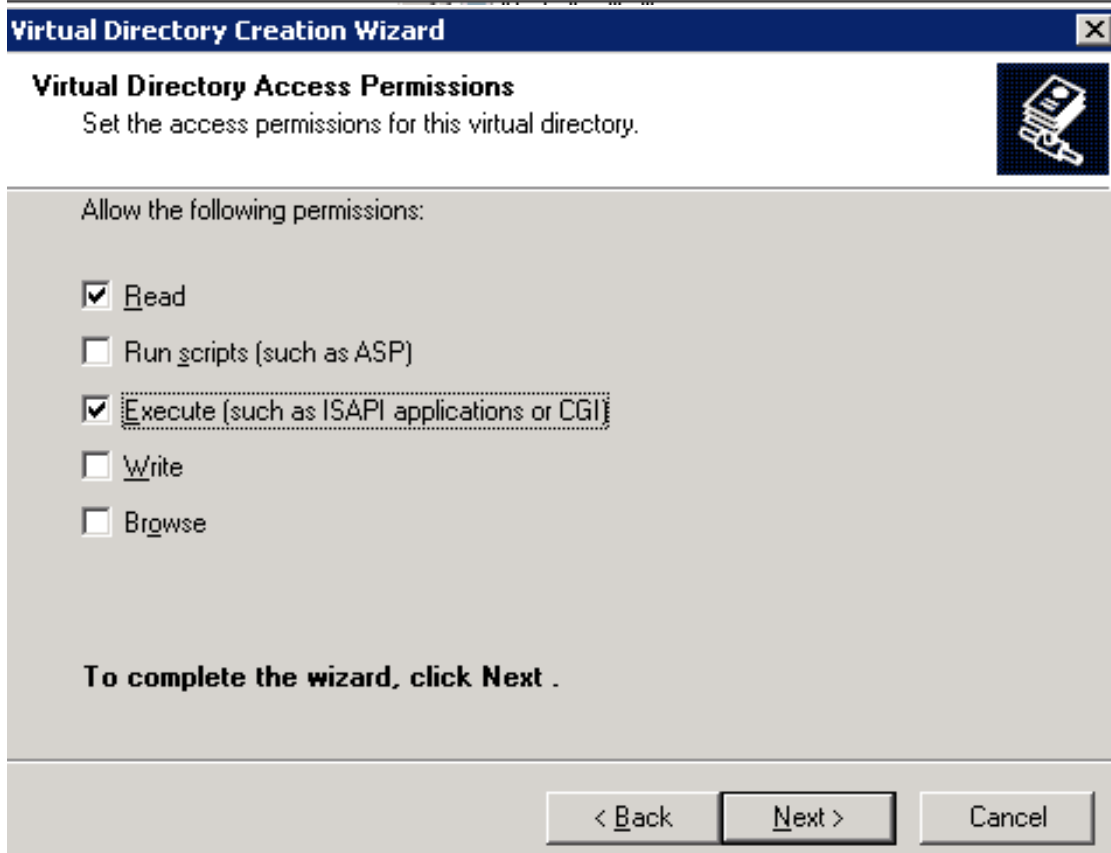
Virtual Directory Creation Wizard

Web Site Content Directory

Where is the content you want to publish on the Web site?

Enter the path to the directory that contains the content for this Web site.

Path:



Virtual Directory Creation Wizard

Virtual Directory Access Permissions

Set the access permissions for this virtual directory.

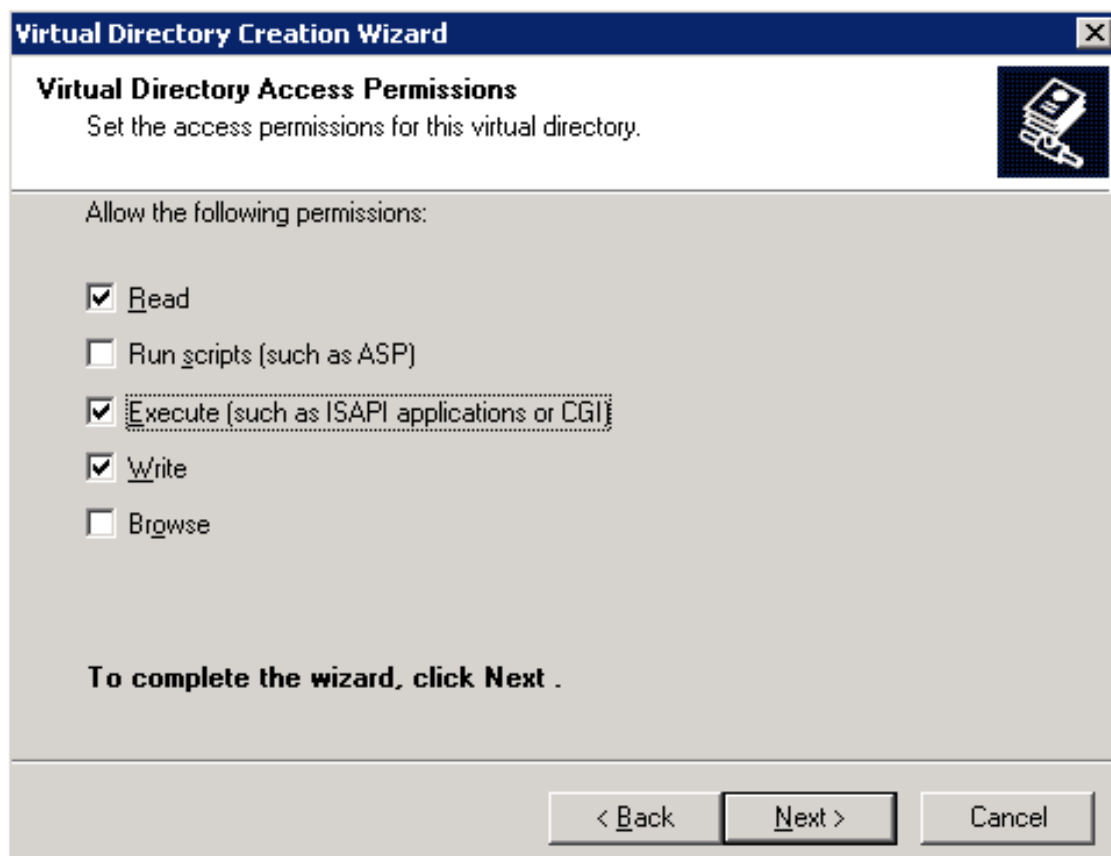
Allow the following permissions:

- ☒ Read
- ☐ Run scripts (such as ASP)
- ☒ Execute (such as ISAPI applications or CGI)
- ☐ Write
- ☐ Browse

To complete the wizard, click **Next** .

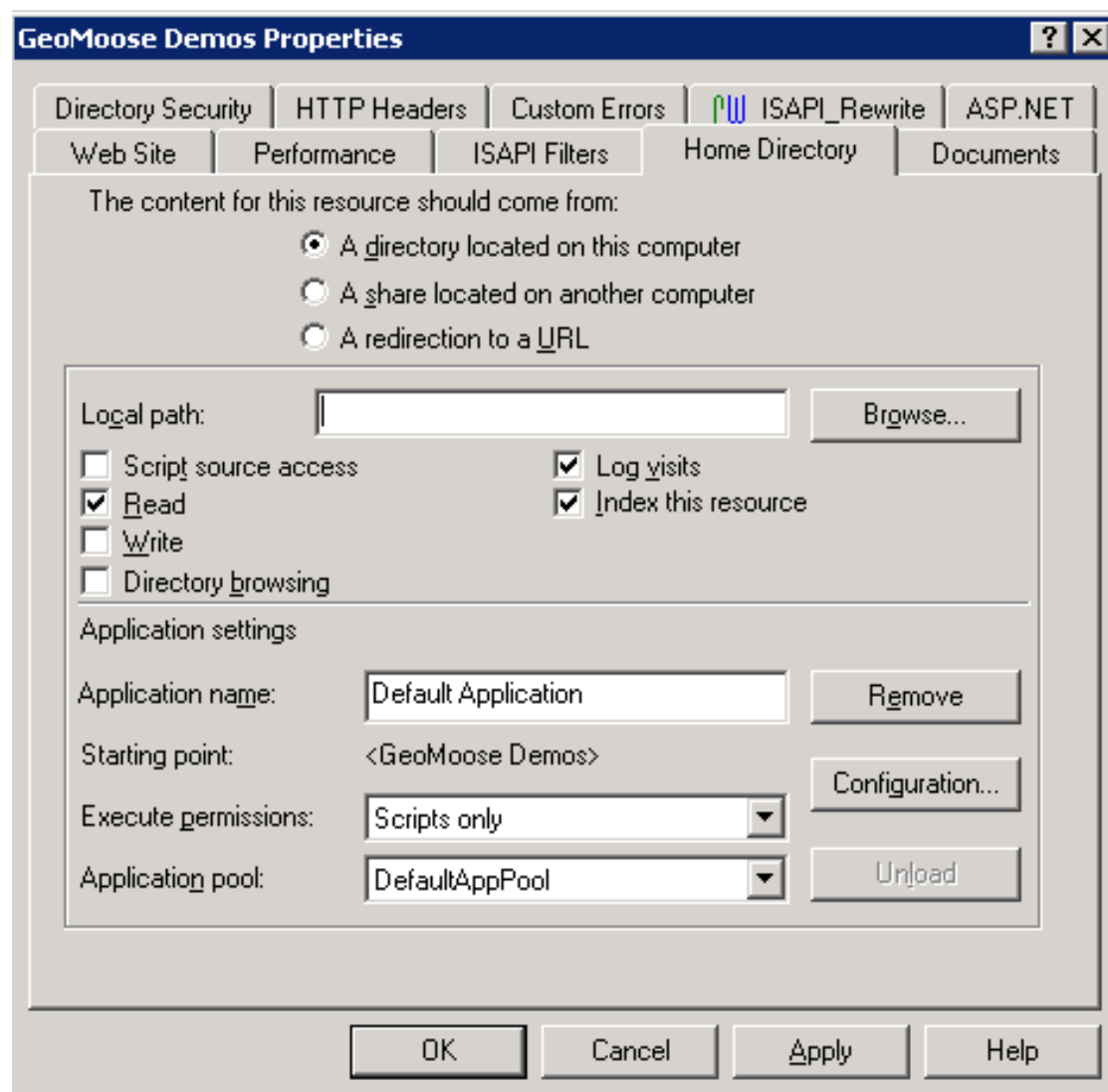


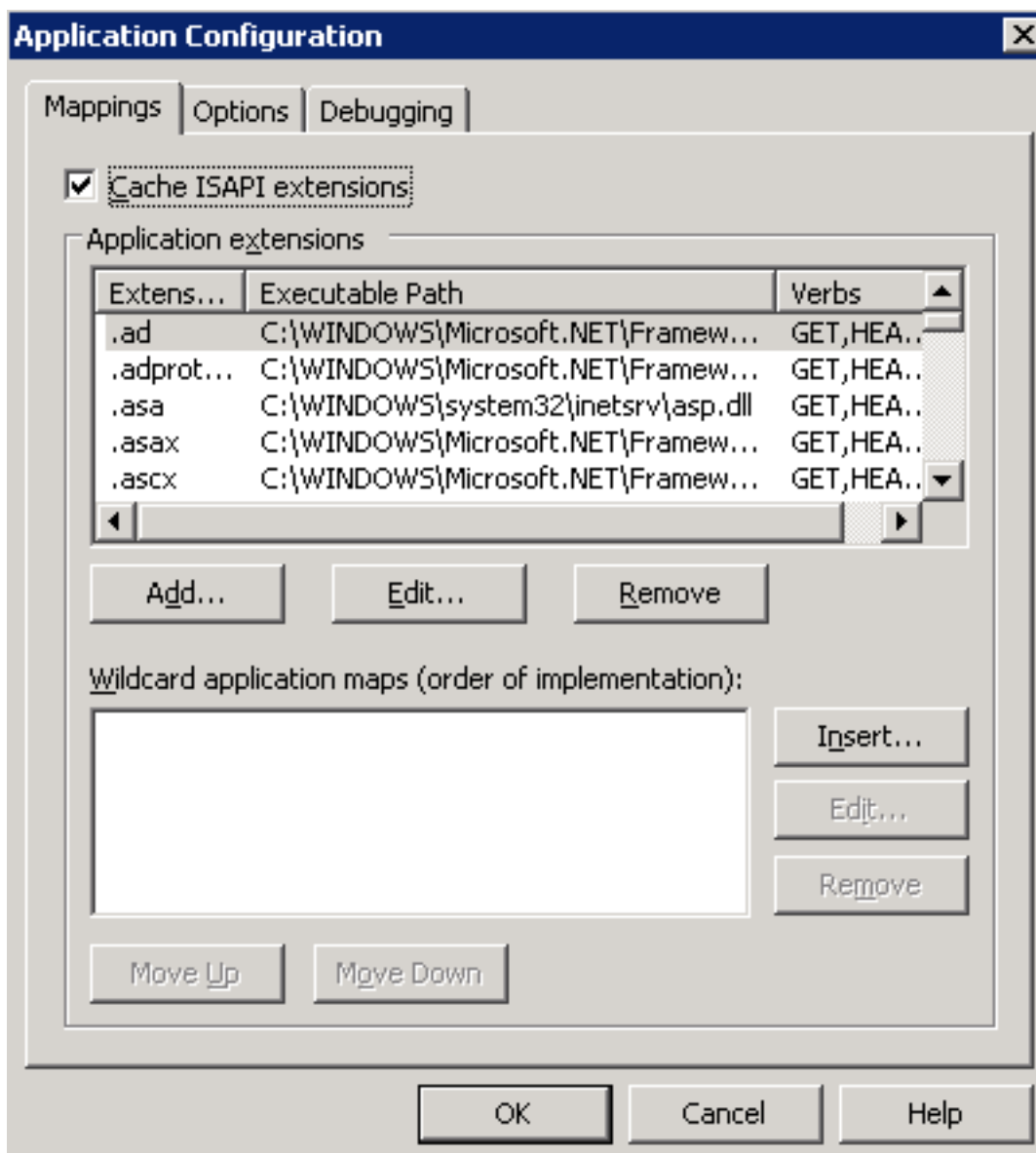
- At the same level as the “geomoose2” directory, create a virtual directory called “cgi-bin” and point it to the path “C:/ms4w/Apache/cgi-bin” using the same settings as the “geomoose2” directory.
- At the same level as the “geomoose2” directory, create a virtual directory called “ms_tmp” and point it to the path “C:/ms4w/tmp/ms_tmp” setting the permissions as shown below.

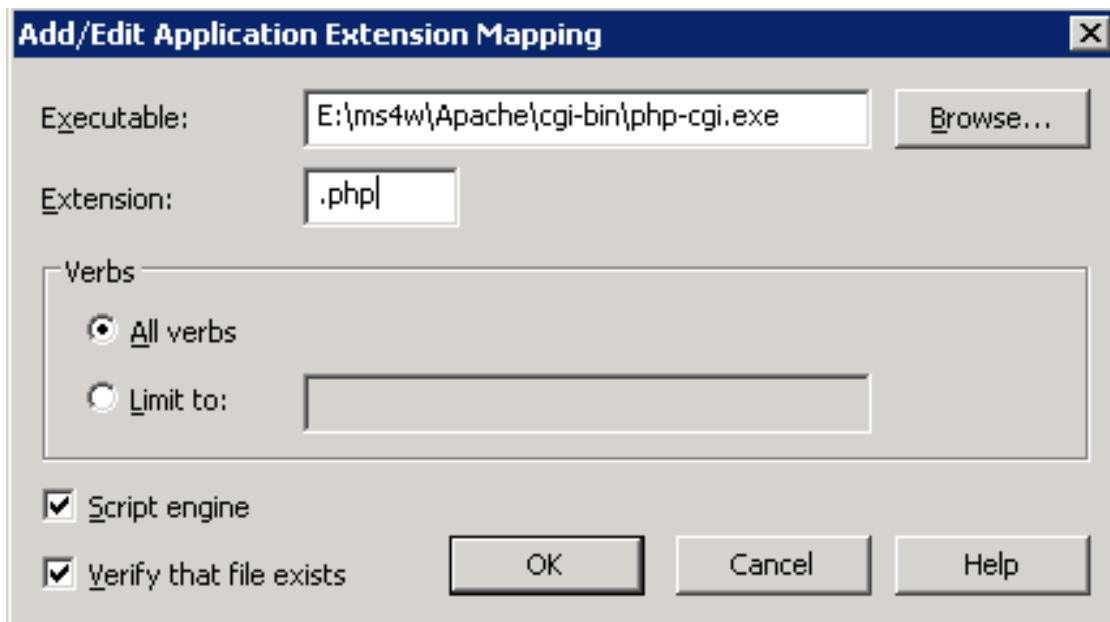


IMPORTANT NOTE There are many versions of PHP available, it is highly recommended that you use the version provided with MS4W as it includes all of the necessary components for MapServer to run properly.

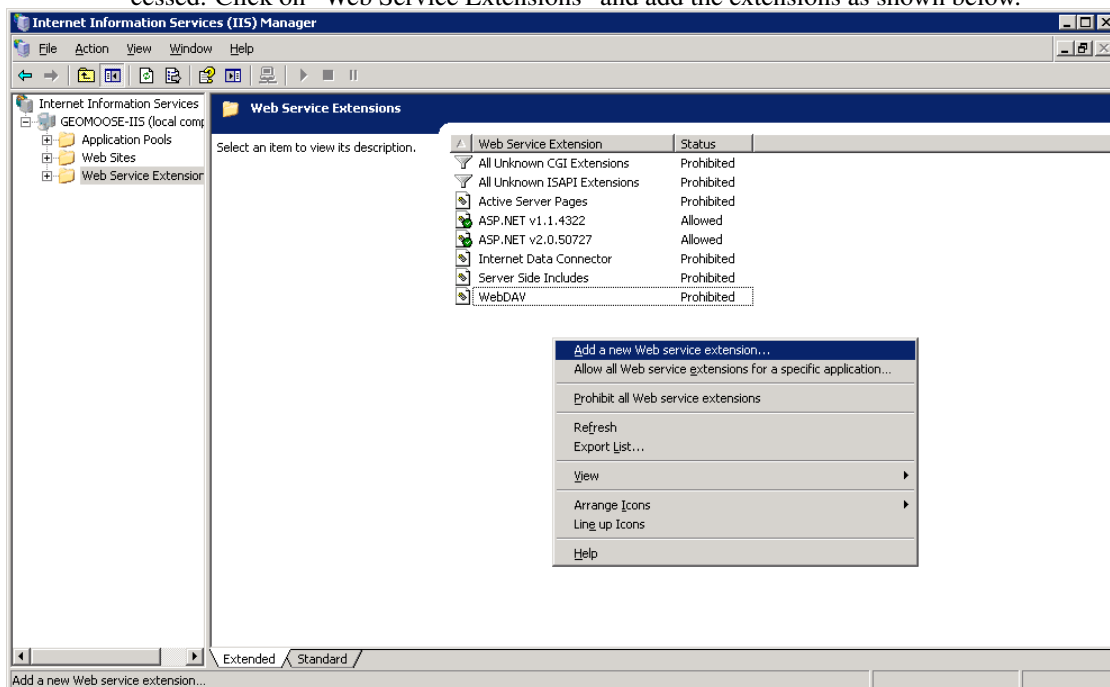
- Set the web service extensions to use PHP from "C:/ms4w/Apache/cgi-bin". This configuration can be done once at the web site level to provide support for all virtual directories (recommended), or at the directory level if some directories need to use different versions of PHP. If you configure PHP at the directory level, make sure to set PHP on all three GeoMOOSE directories.

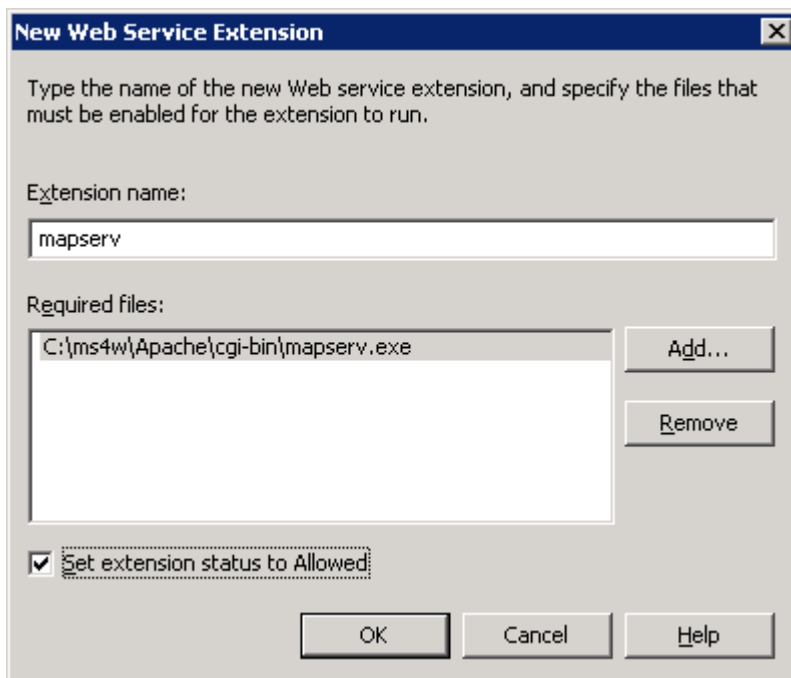
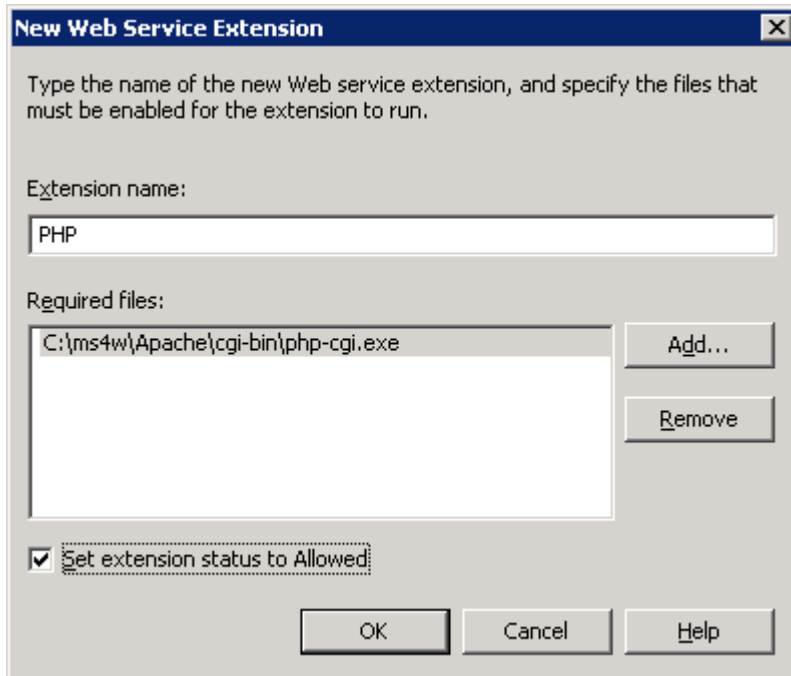






- IIS needs to be configured to allow two CGI applications (php-cgi.exe and mapserv.exe) to be accessed. Click on “Web Service Extensions” and add the extensions as shown below.





- From a text editor (i.e. WordPad) and open “C:\ms4w\Apache\cgi-bin\php.ini”. Search for “cgi.force_redirect” - this value must be set to zero. It will look like this `cgi.force_redirect = 0`
- To test if the installation is working, from a browser on the computer you installed MS4W and GeoMOOSE, type in the address `http://localhost/geomoose2/geomoose.html`.
- At the bottom of this page is a list of applications installed which use MS4W. Click on the GeoMOOSE 2.6 link to test the installation.

NOTE On some Windows-based computers, you will need to set an environment variable for the location of the prog4 library. This step is needed if you receive the error “msProcessProjection(): Projection li-

brary error. No such file or directory”. Instructions for adding the environment variable can be found at <http://mapserver.org/errors.html>.

Common Issues

- Not having Administrator level access. This will prevent Apache from being installed as a service.
- Another web server is running on the computer. If IIS or Apache is already running on the computer on port 80, you will receive an error message when trying to start your new web server. There are many sources on the web which will provide instructions on running both IIS and Apache on the same server by changing the port used by one of the servers.
- Firewall blocking port 80. If you can access GeoMOOSE from the localhost but not other computers, check firewall settings on the computer hosting GeoMOOSE and other external firewalls.

Next Steps

Now that GeoMOOSE is installed, the next step is to start customizing it for your map. Some helpful documents to look at include:

- *Mapbook Reference Guide*
- *GeoMOOSE Configuration Options*
- *How To Add Layers*
- *How To Create Your Own Skin*

GeoMOOSE is proud to be free software. Our *license* reflects that.

1.5 Instructional Materials

1.5.1 FOSS4G 2011 Instructional Materials

- Exercises PDF
- *Writing a GeoMOOSE (JavaScript) Extension*
- *Understanding GeoMOOSE Services*
- Exercises ZIP

1.5.2 MN GIS/LIS Spring 2013/FOSS4G NA 2013 Workshop Materials

- Exercises PDF
- Exercises ODT
- Exercises ZIP

1.6 Developer Specific Documentation

Those really hungry for customization can visit our *Developer Documentation*. You can also find links to our RFC's, API documentation, and release notes.

DEVELOPER DOCUMENTATION

The purpose of this section is to document how to modify and/or extend a specific version of GeoMOOSE beyond what can be done via configuration. The intended audience is people who are familiar with JavaScript and potentially server side programming.

Also, see our *project wide documentation* <<http://www.geomoose.org/developer>> page for information about contributing to the project.

2.1 config.js Configuration Options

- `links_bar_html`
HTML to be contained in the bar after the header and before the toolbar.
- `waiting_html`
Message to use when a user is waiting for something to load.
- `mapserver_url`
The URL to use for MapServer-type map sources.
- `control_panel_side`
Change which side on which the control panel resides.
- `mapfile_root`
The filesystem path on which all mapfiles reside. Used for Mapserver-type map sources.
- `mapbook`
The URL to the mapbook. Defaults to `php/getmapbook.php`.
- `fractional_zoom`
Toggles whether ‘in between’ zooms or strict steps should be supported.
- `scales`
List of scales to be used in the zoom-ladder.
- `max_extent`
The maximum bounding box of the map.
- `initial_extent`
The initial bounding box view of the map.

- `projection`
The projection to be used for the map.
- `ground_units`
Set the ground units of the map. Defaults to 'm'.
- `catalog_name`
Set the catalog's label. Defaults to 'Catalog'
- `group_checkboxes`
Allow entire groups to be toggled on or off with a global checkbox. Defaults to true.
- `flashy_bits`
Toggles Animations on or off. Defaults to true
- `zoomto`
Object representing the zoom to drop downs.
- `jumpto_scales`
Object containing the label:scale values for the scale jumper.
- `startup_service`
Service to call at startup. Defaults to false
- `layer_control_order`
Changes the display order of the layer controls. Stored as an array.
- `layer_controls`
object containing objects describing the layer controls.
- `layer_options.transitionEffect`
OpenLayers defined transition effect for all layers. Defaults to null.
- `layer_options.buffer`
OpenLayers vector layer buffer radius. Defaults to 0.
- `layer_options.ratio`
OpenLayers "ratio" for vector layers. Defaults to 1.
- `catalog.toggle_controls`
Turn on or off the ability to hide the layer controls in the catalog. Defaults to true.
- `catalog.show_controls`
Turn on or off the controls in the catalog. Defaults to true.
- `reference_map`
Not yet implemented.
- `cursor`
Object of CSS cursor definitions
- `cursor.pan`
CSS for pan tool.

- `cursor.zoomout`
CSS for zoomout tool.
- `cursor.zoomin`
CSS for zoomin tool.
- `cursor.measure`
CSS for measure tool.
- `cursor.measurearea`
CSS for measurearea tool.
- `scale_line.enabled`
Toggle whether the scale line should be displayed on the map. Defaults to true.
- `scale_line.top_units`
What units to display on the top of the scale line.
- `scale_line.bottom_units`
What units to display on the bottom of the scale line.
- `scale_line.width`
The width, in pixels, of the scale line.
- `measure_tool`
options for the measure tools.
- `measure_tool.precision`
Number of digits to display in the readings.
- `measure_tool.style`
OpenLayers style object.
- `measure_tool.line_units`
Default units to use for Line measurements.
- `measure_tool.area_units`
Default units to use for Area measurements.
- `default_tool`
Tool to default to when `GeoMOOSE.activateDefaultTool` is called.
- `popups`
options for controlling the behavior of popups.
- `popups.left_offset`
This is used in conjunction with popup CSS styling to properly position the mouse after a popup has been dragged.
- `services`
Options for controlling the behaviour of services.

- `services.disable_hidden_tabs`
Toggle whether to disable tables when they have been hidden. Defaults to false.
- `messages`
Object of error messages.
- `messages.requirement_failure`
Shown when the user does not fill in a required feild.
- `messages.invalid_tool`
Shown when a user tries to start a tool which is not properly configured.
- `messages.service_config_error`
Shown when an admin does not properly configure the tools for a service.
- `messages.mapbook_invalid`
Shown when the server does not return a valid mapbook contents.
- `message.mapbook_version`
Shown when a user tries to load a 1.X version of the mapbook.
- `mapbook_param_error`
Shown when the mapbook contains an invalid <param> tag.
- `messages.service_return_error`
Shown when a service failes to return valid XML.

2.2 Understanding GeoMOOSE Services

2.2.1 Introduction

GeoMOOSE expects services to return a special HTML format that allows it to display information and manipulate the user interface. Learning how to craft custom services enables developers and administrators to extend GeoMOOSE past the capabilities provided in the demos.

2.2.2 Our First Service

To create our first service we are going to focus on a very basic GeoMOOSE PHP-Service. Save these contents to *php/myservice.php*:

```
<?php
header('Content-type: application/xml')
?>

<results>
  <script>
  </script>
  <html>
    This is my first service.
  </html>
</results>
```

The following is a commented version of the code explaining all of the components.

```
<?php
# Switch PHP to return XML instead of HTML
header('Content-type: application/xml')
?>

<results> <!-- opening return tag -->
  <script>
    <!-- this is where we can put javascript to be executed by GeoMOOSE, we leave it blank
  </script>
  <!-- HTML sections are displayed in the tab when data is returned by the service -->
  <html>
    <!-- this is the html we are returning -->
    This is my first service
  </html> <!-- close the html section -->
</results> <!-- close the results -->
```

2.2.3 Defining the Service in the Mapbook

Integrating a service with GeoMOOSE requires two parts. First, defining how to communicate with the service. This is done with a `<service>` tag. Second, we need to define a tool in the toolbar that calls the the service we have defined.

To use our example, we need to find an open place in the Mapbook to add the text. For our purposes, find the `</configuration>` tag, and insert the following text afterwards.

```
<service name="newservice" title="My New Service">
  <url>php/myservice.php</url>
</service>
```

The contextual meaning of the different attributes and children are listed here.

```
<!--
The "name" attribute is used to refer to the service from here after.
The "title" attribute is used to label the service later.
-->
<service name="newservice" title="My New Service">
  <!-- url defines the path to the service -->
  <url>php/myservice.php</url>
</service>
```

The `<service>` entry above defines how the application should communicate with the service. Now, we need to add the service to

- *internal* - These are built-in javascript type tools that trigger different behaviors in the UI.
- *javascript* - These can be used to add arbitrary javascript code to the “onclick” event to a tool.
- *service* - These call services defined by `<service>` tags and is the type we will use in our example.
- *layer* - These call layer editing tools.

The easiest spot to add this entry is before the `</toolbar>` line.

```
<tool name="my_tool" title="My service" type="service" selectable="false" service="newservice"/>
```

The other attributes for a tool are.

- *name* - This is the name of the tool. It can be used to hook to different styling elements.
- *title* - This is the label for the tool. By default the label is only shown when hovering over the tool.

- *selectable* - Tools that are *selectable* cannot be used when other tools are being used. If tool has *selectable* set to false then it will not prevent other tools from operating at the same time.
- *service* - This is the service to call when this tool is clicked. This is only valid for *tool's* with *'type="service"*.

2.2.4 Testing out the new Service

To test the new service, clear the browser cache and reload GeoMOOSE. A small binoculars should appear at the end of the list of tools.

Clicking on the binoculars will make the text “This is my first service.” appear in the Results tab.

2.2.5 Adding User Input

From time to time, it is necessary to allow users to enter parameters to the service. GeoMOOSE supports a number of different inputs types that can be found in [this reference](#).

First, we will modify the service to output some requests.

```
<?php
header('Content-type: application/xml')
?>

<results>
  <script>
  </script>
  <html>
    This is my first service,
    <?php echo $_REQUEST['username']; ?>,
    and you drew the shape
    <?php echo urldecode($_REQUEST['wkt']); ?>.
  </html>
</results>
```

Second, we will add an input “step” to the service. GeoMOOSE allows services to have a number of input steps. Each input step is like a slide in a presentation, or pages of a book, where the user is prompted to enter information. A special type of step is the *type="spatial"*, these will prompt the user to draw a shape on the map. This lets services have both multiple pages of inputs and multiple spatial inputs. In this example, we will add both a spatial and an input step.

```
<service name="newservice" title="My Service">
  <url>php/myservice.php</url> <!-- The URL to our service -->
  <!--
    This will prompt the user to create a shape. The shape
    will be stored in WKT and sent to the script in a field named wkt
  -->
  <step type="spatial" name="wkt">
  </step>
  <!--
    This will ask the user to enter a user name.
  -->
  <step type="input">
    <input name="username" type="user" title="Username:"/>
  </step>
</service>
```

2.2.6 Annoying the User

Up to this point we have explored the `<html>` section of a GeoMOOSE service results. The `<script>` tag is the much more powerful

1. Underlying API's and DOM ID's can change. Try not to use Javascript that is dependent on the back-end code's internal API structure. Many of the DOM ID's used by GeoMOOSE and OpenLayers are automatically generated and even small changes can change those ID's.
2. The file *geomoose.js* contains the *GeoMOOSE* API which we strive to maintain compatibility between revisions.

Our first demonstration will be to annoy the user. We will once again modify *myservice.php*.

```
<?php
header('Content-type: application/xml')
?>

<results>
  <script><![CDATA[
    alert('Hello, <?php echo $_REQUEST['username'] ?>, this is probably annoying.');

```

2.2.7 Doing something Useful

While it can be fun to simply annoy users, it is probably more practical to do something useful with the information we are given. This example parses the WKT of the user's input and zooms to that location on the map.

```
<?php
header('Content-type: application/xml')
?>

<results>
  <script><![CDATA[
    var wkt_parser = new OpenLayers.Format.WKT();
    var feature = wkt_parser.read("<?php echo urldecode($_REQUEST['wkt']); ?>");
    Map.zoomToExtent(feature.geometry.getBounds());
  ]]></script>
  <html>
    This is my first service,
      <?php echo $_REQUEST['username']; ?>,
    and you drew the shape
      <?php echo urldecode($_REQUEST['wkt']); ?>.
  </html>
</results>
```

2.2.8 Stylizing Your Toolbar Icon

Showing the Label

Changing the label is very simple. Set the *show-label* attribute to *true*.

```
<tool name="my_tool" title="My service" type="service" selectable="false" service="newservice" show-label="true">
```

Changing the Icon

GeoMOOSE contains icons for the various default icons but changing the default icon for a custom service does require two steps.

First, we need to add a snippet of CSS either as a `<style>` block in the HTML or in an included CSS file.

```
.my_tool_icon {
  background-image: url("images/toolbar/bell.png");
  background-repeat: no-repeat;
  width: 18px;
  height: 18px;
}
```

Ensure the path to the image is correct. If this snippet is added to a subdirectory like 'css' then the path will be '../images/toolbar/bell.png'.

Second, we need to set the *icon-class* attribute.:

```
<tool name="my_tool" title="My service" type="service" selectable="false" service="newservice" show-label="true" icon-class="my_tool_icon">
```

2.2.9 Going Further

This combination of Javascript and HTML can be used to add any number of new features to GeoMOOSE. Taking the time to write a well crafted services allows GeoMOOSE to integrate and extend existing architecture. Also, while these examples use PHP, there is nothing limiting it to that language. Other installs have used Perl, Python, and Ruby to power GeoMOOSE.

The services included with GeoMOOSE's demo cover a wide range of queries. They are very powerful and mature being in use in production environments. They can serve as exceptional reference material.

See also, [<service>](#).

2.3 Writing a GeoMOOSE (JavaScript) Extension

This tutorial is another description of how to write an extension. This one works with interacting with the Map and uses more JavaScript. The more simple Disclaimer example can be found [here](#).

2.3.1 For Review

GeoMOOSE User Extensions allow developers and installers to retool, enhance, or replace parts of the user interface. This example enhances the user interface by allowing it to remember the last location of the user using a cookie.

2.3.2 The Basic Layout of an Extension

This is the Disclaimer extension, fully commented to describe what each line means:

```
/* Class Declaration, this uses the OpenLayers.Class method
 * with the base class of GeoMOOSE.UX.Extension
 */
DisclaimerExtension = new OpenLayers.Class(GeoMOOSE.UX.Extension, {
  /* disclaim will be our member function that
   * actually takes action */
  disclaim: function() {
    /* disclaim first checks to see if there is a
     * disclaimer message set by the user,
     * if not, it simply prompts, "Hello, World." */
    if(window.DISCLAIMER_MESSAGE) {
      alert(window.DISCLAIMER_MESSAGE);
    } else {
      alert("Hello, World!");
    }
  },

  /* Load is the function called when the extension
   * is loaded by GeoMOOSE. */
  load: function() {
    /* Here we register our extension with
     * the onMapbookLoaded event.
     * This is called once the Mapbook
     * has been read by the client. */
    GeoMOOSE.register('onMapbookLoaded', this, this.disclaim);
  },

  /* To fit the OpenLayers class definition
   * convention we set the CLASS_NAME property to the
   * name of our extension */
  CLASS_NAME: "DisclaimerExtension"
});

/* Now we register the extension with the
 * GeoMOOSE User Extension handler */
GeoMOOSE.UX.register('DisclaimerExtension');
```

2.3.3 Starting Simple: Hello World

As is traditional in computer science lessons, we will start with a basic “Hello, World” example. This will be somewhat similar to what we have done above, however, we will start it as the basis for our new extension.

This basic code will create an alert when the extension is loaded, we’ll put the contents into *extensions/RememberWhere.js*:

```
RememberWhere = new OpenLayers.Class(GeoMOOSE.UX.Extension, {
  load: function() {
    alert('Hello, World.');
```

```
  },

  CLASS_NAME: "RememberWhere"
});

GeoMOOSE.UX.register('RememberWhere');
```

Adding the Extension to GeoMOOSE

While the above code creates the extension, it still needs to be included in the HTML to work. The best place to add the code is before the `</head>` entry in *geomoose.html*, it should look like this when finished

```
<script type="text/javascript" src="extensions/RememberWhere.js"></script>
</head>
```

2.3.4 Adding Hooks: With Events!

In both the Disclaimer example and others, it is best to wait for the Mapbook is loaded to manipulate the map. The Mapbook is

1. We will wait for the Mapbook to load.
2. Once it has loaded we will list to the Map to move, then tell the client the bounds.

These hooks look like this:

```
RememberWhere = new OpenLayers.Class(GeoMOOSE.UX.Extension, {
  load: function() {
    GeoMOOSE.register('onMapbookLoaded', this, this.gotMapbook);
  },

  gotMapbook: function() {
    Map.events.register('moveend', this, this.newExtent);
  },

  newExtent: function() {
    alert(Map.getExtent().toArray().join(','));
  },

  CLASS_NAME: "RememberWhere"
});

GeoMOOSE.UX.register('RememberWhere');
```

2.3.5 Giving the Map a Memory

The example thus far will only really serve to bother a user as they are subjected to seeing their new extents every time they navigate. It would be far more useful for them to actually have the map remember where they were. To do this, we will use cookies.

Adding Cookie Functions

GeoMOOSE 2.4 and older do not come with cookie manipulation functions. GeoMOOSE 2.6 includes *dojo*, which has cookie manipulation functions. Since we are using GeoMOOSE 2.4 for this example, we will be manually parsing cookies. This example would still work in 2.6 but using proper cookie function this would be much shorter code.

```
RememberWhere = new OpenLayers.Class(GeoMOOSE.UX.Extension, {
  load: function() {
    GeoMOOSE.register('onMapbookLoaded', this, this.gotMapbook);
```

```

    },

    gotMapbook: function() {
        Map.events.register('moveend', this, this.newExtent);
    },

    newExtent: function() {
        this.setCookie('gm_remember_extent', Map.getExtent().toArray().join(','));
    },

    setCookie: function(name, value) {
        var cookies = this.getCookies();
        cookies[name] = value;
        var all_cookies = [];
        for(var c_name in cookies) {
            if(c_name) {
                all_cookies.push(c_name+'='+escape(cookies[c_name]));
            }
        }
        document.cookie = all_cookies.join(';');
    },

    getCookies: function() {
        var cookies = document.cookie.split(';');
        var all_cookies = {};
        for(var i = 0, ii = cookies.length; i < ii; i++) {
            var split_on = cookies[i].indexOf('=');
            /* get the name and clean it up */
            var name = cookies[i].substr(0, split_on).replace(/^\s+|\s+$/g, "");
            all_cookies[name] = unescape(cookies[i].substr(split_on+1));
        }
        return all_cookies;
    },

    getCookie: function(name) {
        var cookies = this.getCookies();
        return cookies[name];
    },

    CLASS_NAME: "RememberWhere"
});

GeoMOOSE.UX.register('RememberWhere');
```

2.3.6 Now recalling the Extent on Startup

This adds the last bit of code to have the extension recall the extent on startup.

```

RememberWhere = new OpenLayers.Class(GeoMOOSE.UX.Extension, {
    load: function() {
        GeoMOOSE.register('onMapbookLoaded', this, this.gotMapbook);
    },

    gotMapbook: function() {
        if(this.getCookie('gm_remember_extent')) {
            Map.zoomToExtent(OpenLayers.Bounds.fromString(this.getCookie('gm_remember_extent')));
        }
    }
});
```

```
Map.events.register('moveend', this, this.newExtent);
},

newExtent: function() {
  this.setCookie('gm_remember_extent', Map.getExtent().toArray().join(','));
},

setCookie: function(name, value) {
  var cookies = this.getCookies();
  cookies[name] = value;
  var all_cookies = [];
  for(var c_name in cookies) {
    if(c_name) {
      all_cookies.push(c_name+'='+escape(cookies[c_name]));
    }
  }
  document.cookie = all_cookies.join(';');
},

getCookies: function() {
  var cookies = document.cookie.split(';');
  var all_cookies = {};
  for(var i = 0, ii = cookies.length; i < ii; i++) {
    var split_on = cookies[i].indexOf('=');
    /* get the name and clean it up */
    var name = cookies[i].substr(0, split_on).replace(/^\s+|\s+$/g, "");
    all_cookies[name] = unescape(cookies[i].substr(split_on+1));
  }
  return all_cookies;
},

getCookie: function(name) {
  var cookies = this.getCookies();
  return cookies[name];
},

CLASS_NAME: "RememberWhere"
});

GeoMOOSE.UX.register('RememberWhere');
```

2.3.7 Unlimited Potential

Due to the very run-time replacement happy nature of Javascript applications nearly any part of the GeoMOOSE User Interface can be supplemented with extensions. Interacting with services and the Map allows most behaviors to be changed by the customizer.

2.4 GeoMOOSE Sprite Generator, Toolbar and Catalog Styles

To make GeoMOOSE more efficient all toolbar and layer controls have been combined into a single image. All of the tool images are changed by manipulating CSS settings.

2.4.1 The GeoMOOSE Sprite

A “sprite” is a combined image that contains many smaller “sub images.” In older versions of GeoMOOSE every image was stored in its own file.

To decrease this load we have created a method for creating a sprite and a CSS file that ties together the combined image and the differing tools.

2.4.2 Explaining the Sprite Layout

The sprite has two columns. The left column is the tool in its “normal” state and the right column contains the tool in its “selected” state. Each row represents a different tool.

2.4.3 Generating the Sprite

The sprite is generated with the “createSprite.py” script located in the “tools” directory. The “createSprite.py” script also generates the CSS. The CSS outputs to the standard output. The general operation for generating the sprite is as follows:

1. `cd tools`
2. `./createSprite.py > ../htdocs/css/sprite.css`
3. Open `htdocs/images/all.png` in GIMP
4. Save the PNG as a GIF for IE6 support. *If steps 3 and 4 are neglected IE users will see very strange sprites.*

To specify an image for a “selected” tool add an image in the `images/toolbar` directory with “-selected” included at the end of the name. For example, the `popups` tool has two images in the toolbar directory: “`popups.png`”, “`popups-selected.png`”. When `createSprite.py` runs it searches the directory for the “selected” images and places them in the right-column of the sprite as appropriate.

2.4.4 Toolbar Styling

The toolbar styling has two components:

1. The sprite imaging layout explained above.
2. `.Tool` and `.selected`.
 - (a) `.Tool` – defines the basic CSS for laying out the tool.
 - (b) `.selected` – defines additional information when the tool is selected. In the default installation this makes the toolbar tool background grey.

..note:: You will need to have Python installed on your computer to run this script. If you are running Windows, you probably need to get PIL library for Python. See <http://www.pythonware.com/products/pil/>

2.5 How To Build GeoMOOSE from source

These instructions apply to the GeoMOOSE 2.6 series.

You will need to rebuild GeoMOOSE from source if you make any changes to the core code. This is because the core code is optimized and minified using the Dojo build system and the output is placed in the `htdocs/build` directory. `geomoose.html` is set to use this optimized build because it allow for much faster application load times than using the raw source. While developing, you may find it more convenient to use `geomoose_dev.html` which bypasses the build system and uses the raw JavaScript sources.

Note: These instructions require you to run the Dojo build system. For now, this means you will need access to a Linux like command line (bash) and have Java or OpenJDK installed in your path. This works trivially on Linux and OS X, and probably works from within Cygwin on Windows, but this has not been tested.

The GeoMOOSE documentation is maintained using reStructured text (<http://sphinx.pocoo.org/rest.html>) and built into HTML or PDF using Sphinx. Thus, to build the GeoMOOSE documentation from source, you will also need Sphinx installed (<http://sphinx.pocoo.org>).

2.5.1 Step 0: Obtain a copy of GeoMOOSE

You can use whatever method you prefer, check out from svn:

```
svn co https://svn.osgeo.org/geomoose/geomoose2/trunk geomoose-trunk
```

Or extract from a tarball.

2.5.2 Step 1: Extract the libraries in libs

From the GeoMOOSE base directory (e.g. `geomoose-trunk`):

```
cd htdocs/libs
tar xzf OpenLayers-*.tar.gz
tar xzf dojo-*.tar.gz
unzip proj4js-1.0.2.zip
```

Next, run `patch_dojo.sh` to setup the Dojo build system for GeoMOOSE.

```
./patch_dojo.sh
```

2.5.3 Step 2: Building GeoMOOSE

Again, from the GeoMOOSE base directory, change directories to where the Dojo build system lives. Then, run the `build`. This will rebuild the files in `htdocs/build`:

```
cd htdocs/libs/dojo-1.6.1/util/buildscripts
./build_geomoose2.sh
```

2.5.4 Step 3: Building the GeoMOOSE documentation

Again, from the GeoMOOSE base directory, change directories to where the Sphinx source lives. Next, build the source using the supplied Makefile (or `make.bat` on Windows):

```
cd sphinx-docs
make html
```

2.6 API Docs

API Docs are available here.

GEOMOOSE LICENSE

3.1 License Summary

The GeoMOOSE license is an MIT based license.

3.2 License Text

Copyright (c) 2009-2012, Dan “Ducky” Little & GeoMOOSE.org

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

IMPLEMENTER DOCUMENTATION

This section contains reference information that you would need to install and configure GeoMOOSE for your site. It is assumed that you have read the *Getting Started with GeoMOOSE* section and that you are generally familiar with GIS principles and running a web server on your platform.

4.1 Configuration References

4.1.1 Mapbook Reference Guide

The mapbook document is an XML file that is used as the configuration file for a GeoMOOSE application. It configures things such as map sources, layers, services and tools in the application. For an example of a mapbook, please refer to the GeoMOOSE demo and the mapbook.xml file in the conf folder.

This reference guide explains the structure, elements, and attributes of the Mapbook XML tags, and how they effect the user interface. GeoMOOSE version 2.0 looks to extend the capabilities and ease of configuration from GeoMOOSE 1.x series and therefore a new version of the mapbook format was required to support all of those features.

<mapbook>

All version 2.0 mapbooks should express themselves by using the 'version' tag. GeoMOOSE will check the version attribute at startup and if the minimum version requirement is not met, GeoMOOSE will alert the user with an error. This makes the version attribute required.

The <mapbook> tag contains 6 child elements. Each of these child elements can have their own child elements and are described below.

- <configuration> : The <configuration> tag is used to specify user interface startup settings.
- <map-source> : The <map-source> tag is used to specify a single or collection of layers in GeoMOOSE.
- <layer-controls> : The <layer-control> tag is used to link a layer control to a service. As of GM2 the only layer control available are popups.
- <service> : The <service> tag is used to define a service such as select, printing and identify.
- <catalog> : The <catalog> tag is the layers listing found in the information panel.
- <toolbar> : The <toolbar> tag is the tools that are displayed on the user interface toolbar.

<configuration>

The configuration tag is used to specify user interface startup settings and other application settings. This is done within the <configuration> ... </configuration> section of the mapbook.xml. The configuration tag has <param> children. <param> elements have a simple format, a “name” attribute which specifies the interface setting to change and then CDATA that specifies the values. The following are valid parameters for the configuration children. All configuration parameters are documented in *GeoMOOSE Configuration Options*. Here is an example from the GeoMOOSE Demo:

```
<configuration>
  <param name="links_bar_html"><![CDATA[
    <b>Our Stack:</b>
    <a target="_blank" href="http://www.geomoose.org">GeoMOOSE.org</a> |
    <a target="_blank" href="http://www.mapserver.org">MapServer</a> |
    <a target="_blank" href="http://www.openlayers.org">OpenLayers</a> |
    <a target="_blank" href="http://www.dojotoolkit.org">Dojo</a>
  ]]></param>
  <param name="projection">EPSG:3857</param>

  <param name="zoomto"><![CDATA[
    {
      "Jump To:" : {
        "World" : [-20614760.777156,1751325.1919492,-1927436.1053437,7915207.1517617],
        "Parcel Data" : [-10384069.859924,5538318.529767,-10356632.423788,5563580.927174],
        "Full State of MN" : [-10742765,5398288,-9920914,6310641]
      }
    }
  ]]></param>
  <param name="max_extent">-20037508.342789,-20037508.342789,20037508.342789,20037508.342789</param>
  <param name="initial_extent">-10384069.859924,5538318.529767,-10356632.423788,5563580.927174</param>
  <param name="measure_tool.show_area_segments">>false</param>

  <param name="layer_controls.legend.on">>false</param>
  <param name="layer_controls.up.on">>false</param>
  <param name="layer_controls.down.on">>false</param>
  <param name="layer_controls.metadata.on">>false</param>
  <param name="layer_controls.legend.on">>true</param>

  <param name="group_checkboxes">>false</param>

  <param name="default_tab">Catalog</param>

  <param name="ground_units">m</param>
  <param name="maxResolution">156543.03390625</param>
  <param name="numZoomLevels">20</param>

  <param name="reference_map.enabled">>true</param>
  <param name="coordinate_display.usng">>true</param>
  <param name="jumpto_scales"><![CDATA[
    {
      "1:100000" : 100000,
      "1:50000" : 50000,
      "1:24000" : 24000,
      "1:10000" : 10000,
      "1:5000" : 5000
    }
  ]]></param>
</configuration>
```

<map-source>

The `map-source` tag is used to specify a single or collection of layers in GeoMOOSE. These are all in the `<map-source> </map-source>` section and later referenced in the `<catalog> </catalog>` section. All map-sources have two required attributes:

- `name` – The name that GeoMOOSE will use to reference this mapsource.
- `type` – The type of layer. This attribute determines what other children are required (or available) and tells GeoMOOSE what type of mapping service with which to communicate. Valid types include: Mapserver, WMS, WFS, and others (described below).
- `map-source`'s also support an optional `opacity` attribute that is a number between 0 and 1 specifying how transparent the image will be displayed. For example `opacity=".5"` will display the image from the map-source as half-transparent.
- `buffer` attribute. Boolean, defaults to true, also optional. When set to "false" the image does not "buffer" around the map, it is trimmed to the exact size of the map display. Useful for MapServer scalebars.

All map-source types support two children:

- **<param>** – The 'param' child will add or change parameters sent over the URL to a <map-source>. This can be necessary
 - `name` – This is the name that will be used in the URL.
 - `value` – This is the value the name will be set to in the URL.
- **<layer> (required)** – At least one layer child is required for each map-source. Even if a mapserver-type layer is used where
 - `name` – For a WMS map-source, this refers to a layer as specified in the GetCapabilities document. For a Mapserver map-source, this refers to the name of a layer in the Mapfile. If a Mapfile has all of the layers set to default or all of the layers should be on or off as a whole, then the 'name' should be set to 'all.'
 - `status` - Boolean. Toggles whether the <layer> should be on by default. **WARNING!** It is possible for conflicts to arise from setting the status attributes of <layer> elements and mixing that improperly with the settings in the catalog.

AGS

ESRI provides access to ArcGIS data through a standard services. For more information about ESRI services go to <http://www.esri.com>. The basic *additional* parameters referenced to use an ArcGIS service are:

- `<url>` – Url for location of service.
- `<layer>` – This is the name used internally, by GeoMOOSE to refer to the layer.
- `<param>` – `name=FORMAT`. Parameter to identify format of the service. Values for the format used in the example is "png".

Example:

```
<map-source name="ags" type="ags">
  <url>http://services.arcgisonline.com/ArcGIS/rest/services/NatGeo_World_Map/MapServer/export
  <layer name="NatGeo_World_Map"/>
  <param name="FORMAT" value="png"/>
</map-source>
```

Google

Google provides a variety of map services and API's that GeoMoose can consume. For more information about specifics and licensing please go to <https://developers.google.com/maps/>. The basic *additional* parameters referenced to use a Google service are:

- *<layer>* – This is the name used internally, by GeoMOOSE to refer to the layer. “all” recommended for Google map-source types.
- *google-type* – This is the type of Google layer that will be provided. This can include: satellite, physical, streets, and hybrid.

Example:

```
<map-source name="google_streets" type="google" google-type="streets">
  <layer name="all"/>
</map-source>
```

Mapserver

This type of layer is meant to communicate with the default mapserver as specified in config.js. The basic *additional* parameters reference to use a Mapserver layer are:

- *<layer>* – The name of layers to be referenced by the map source. This name must be referenced in the mapserver file. The layer name could also be “all” and all layers within the mapserver file will be used.
- *<file>* – Layers of type ‘mapserver’ require a value *<file>* child specifying the location of the mapfile on disk. Mapfiles can be specified with relative paths in *<file>* tags.

Examples:

```
<map-source name="usgs" type="mapserver">
  <file>./demo/wms/usgs.map</file>
  <layer name="DOQ"/>
  <layer name="DRG"/>
</map-source>

<map-source name="borders" type="mapserver" reference="true">
  <file>./demo/statedata/basemap.map</file>
  <layer name="city_labels" status="on"/>
  <layer name="county_labels" status="on"/>
  <layer name="city_poly" status="on"/>
  <layer name="county_borders" status="on"/>
</map-source>
```

Vector

Available 2.6+. Vector layers provide the basis for additional GeoMOOSE functionality.

Example:

```
<map-source name="sketch" type="vector" active="true" status="on">
  <style type="stylemap"><![CDATA[
    {
      "label" : "${title}",
      "strokeColor" : "${line_color}",
      "fillColor" : "${fill_color}"
    }
  ]]></style>
</map-source>
```

```

    }
  ]]></style>
  <attribute name="title" type="user" default-value="" label="Feature Label:"/>
  <attribute name="line_color" type="color" default-value="#ff0000" label="Stroke Color:"/>
  <attribute name="fill_color" type="color" default-value="#ff0000" label="Fill Color:"/>
  <!--
  <attribute name="opacity" type="select" default-value="100" label="Opacity (%):"/>
  <attribute name="line_opacity" type="select" default-value="100" label="Stroke Opacity (%):"/>
  -->
  <attribute name="label_only" type="checkbox" default-value="false" label="Only show label in
</map-source>

```

WFS

This type of layer is meant to communicate with Web Feature Service servers. The Web Feature Service is a geospatial data manipulate interface based on HTTP protocol provided by OGC. It belongs to Data Service according to OGC web service architecture and it is the development of OGC Web Map Service. Web Feature Service allows a client to retrieve geospatial data encoded in Geography Markup Language (GML) from multiple Web Feature Services. It also supports INSERT, UPDATE, DELETE, QUERY and DISCOVERY operations on geographic features using HTTP as the distributed computing platform.

- *<Style>* – Allows you to reference a stall map for display.
- *<url>* – URL location of services
- *<attributename>* –
- *<feature-namespace>* –
- *<feature-type>* –
- *<geometry-name>* –
- *<schema>* –

Example:

```

<map-source name="census_cities" type="wfs">
  <style type="stylemap"><![CDATA[
    {
      "strokeColor" : "#00ff00",
      "label" : "${nameIsad}"
    }
  ]]></style>

  <url>/geoserver/GeoMOOSE_Testing/ows</url>

  <attribute name="geoid10" type="text" label="ID:" default-value="27999"/>
  <attribute name="nameIsad10" type="text" label="Name:"/>

  <feature-namespace>geomoose</feature-namespace>
  <feature-type>census_cities</feature-type>
  <geometry-name>wkb_geometry</geometry-name>
  <schema> <![CDATA[http://localhost:8080/geoserver/GeoMOOSE_Testing/wfs?service=WFS&version=1
</map-source>

```

WMS

This type of layer is meant to communicate with Web Mapping Service servers. This is the OGC standard for serving raster images over the web. Many sites deliver WMS services that can be consumed by a GeoMOOSE application. If the type is a WMS data source an optional attribute can also be specified to request the images as tiles. This attribute is `tiled=true`.

- `<layer>` – The name of layers to be referenced by the map source. This name must be referenced in the mapserver file. The layer name could also be “all” and all layers within the mapserver file will be used.
- `name` – This is the name used internally, by GeoMOOSE to refer to the layer. Unlike GeoMOOSE 1.0, the title of the layer displayed in the Catalog is not a one-to-one relationship with the entry in the mapbook. This allows greater flexibility in divorcing the display-order of layers in the map and display order in the catalog.
- `queryable` – This is an optional attribute that tells GeoMoose whether a WMS map-source type is queryable. This attribute is only needed for WMS map-source types. Valid options are `queryable='true/false'`.
- `<url>` – Layers of type ‘wms’ require a `<url>` child specifying the URL of the WMS.
- `<param>` – `name=FORMAT`. The format of the WMS to be delivered. The value for this parameter is typically “image/png” or “image/gif”.
- `<param>` – `name=TRANSPARENT`. Configure if transparency is used.

Example:

```
<map-source name="metro" type="wms">
  <url>http://www.datafinder.org:80/wmsconnector/com.esri.wms.Esrimap/MN_MetroGIS_DataFinder_WM
  <layer name="stream_net_1"/>
  <param name="TRANSPARENT" value="TRUE"/>
  <param name="FORMAT" value="image/png"/>
</map-source>

<map-source name="test" type="wms" queryable="true">
  <url>http://maps.co.lincoln.or.us/cgi-bin/mapserv.exe?</url>
  <param name="map" value="/ms4w/apps/lincoln/04/queryLincoln2.map"/>
  <layer name="taxlots" queryable="true" />
  <param name="TRANSPARENT" value="TRUE"/>
  <param name="FORMAT" value="image/gif"/>
</map-source>
```

XYZ

Many TMS-like tiles are accessible in this format.

Example:

```
<map-source name="mapquest" type="xyz">
  <layer name="osm" />
  <url>http://otile1.mqcdn.com/tiles/1.0.0/osm/{z}/{x}/{y}.png</url>
  <url>http://otile2.mqcdn.com/tiles/1.0.0/osm/{z}/{x}/{y}.png</url>
  <url>http://otile3.mqcdn.com/tiles/1.0.0/osm/{z}/{x}/{y}.png</url>
  <url>http://otile4.mqcdn.com/tiles/1.0.0/osm/{z}/{x}/{y}.png</url>
</map-source>
```

Yahoo

Note: As of GeoMoose 2.6.2 the Yahoo MapSource type is no longer available. This is because GeoMoose 2.6.2 updated to OpenLayers 2.12 which has [dropped support](#) for the Yahoo layer types.

Do not use the Yahoo layer types, they will no longer work.

<service>

There is a lot of service documentation. See also, [Understanding GeoMOOSE Services](#), [Service Communication Protocol Reference](#), and [Standard Services](#).

The service element is the key to GeoMOOSE's interoperability. All functions are a type of service (identify, select, Birdseye view, etc.). Services have attributes to define some of it's behaviour:

- *name* - This is the internal name of the service.
- *display* - Sets whether the the service will be displayed while "launching." It is useful to set this to false if there is no text-entry required for the service.
- *title* - This title will be displayed automatically in the input tab.
- *target* - If set then this attribute will display the results in a different window.
- *display-submit* - If set to *false* will not display the "Go!" button.
- *clear-highlight* - Boolean. Defaults to *false*. Controls whether the highlight layer should be cleared when closing the service's tab.

Service have one `<url>` child. The `<url>` child specifys the URL for of the service for GeoMOOSE to call:

```
<url>php/idnetify.php</url>
```

Services each have `<step>` children. There are two types of steps:

- ***type=spatial*** - **This type of step allows the layer to. Spatial steps have additional inputs that are used to define additional in**
 - *name* - This attribute specifys the CGI/form variable name in which the representation of the shape will be stored.
 - *default* - Sets the default tool used for drawing the shape. Valid values: point, line, polygon.
 - *point | line | polygon | edit-polygon* - Turns on/off the various drawing tools. Valid values: true, false. The default behaviour for these are stored in Configuration.
 - ***format*** - **This specifys the format of the shape:**
 - * *=WKT* - Use well-known text.
 - * *=delim* - Use a text delimited format.
 - *jump-start* - When true then the service will attempt to start after the shape has been finished.
 - *reproject* - This can be used to specify a shape projection other than the default map projection.
 - *show-tools* - If set to false this attribute will not display the list of tools. It is useful when only one drawing tool is defined.
- *type=input* - This step only allows for the standard inputs.

Steps can have many `<input>` children. `<input>` s are very similar to the ones used in HTML forms. All `<input>` s require a *name* attribute and a *type* attribute. The following documents the various *type* s of inputs:

- *type=extent* - Sends the current extent to the service.
- *type=hidden* - Hidden inputs are not displayed to the user but the value set in the *value* attribute is sent to the service.
- *type=select* - Creates a drop down box. The *title* attribute sets the label for the drop down. These have the same *<option>* children as the HTML *<select>* elements.
- *type=sketches* - Sends JSON describing all of the user sketches to a service.
- *type=user* - These are basic user input text fields. Their default value can be set using the *value* attribute. The *title* attribute sets the label for the text field.
- *type=visiblelayers* - Sends the list of visible layers.
- *type=ajax_select* - Creates a drop down box populated by a remote server script. These inputs types feature a *<url>* child which references the script to call. The script should return HTML-style *<option>* elements, a sample return may look like this:

```
<options>
  <option value="a">A</option>
  <option value="b">B</option>
</options>
```

- *type=precision* - Sends the precision of the map to the service.

<layer-controls>

As of GM2 the only layer control available are popups.

<catalog>

The GeoMOOSE Catalog, or Catalog, is the layers listing found in the table of contents on the user interface. The catalog is represented by the *<catalog>* child inside of the mapbook. The catalog has two types of children:

- **group** – The ‘group’ child creates folders in the catalog. Groups contain layer children and can also contain other groups.
 - *title* – This is the text that will be shown in the Catalog to represent the layers specified in the name attribute.
 - *expand* – A group can be started as ‘open’ where the contents of the group are listed. The ‘expand’ attribute can either be ‘true’ or ‘false’. This defaults to ‘false’.
 - *multiple* – While most layers are stacked, there may be a set of layers that should not be stacked on top of each other. The most common example is background layers. In the case of background layers, only one should be selected at a time. To do this in GeoMOOSE, set the group’s ‘multiple’ attribute to ‘false’. This defaults to ‘true’.

Group example (with sub group):

```
<group title="Overlays">
  <group title="County Layers" expand="false">
    <layer title="Parcels" src="parcels/parcels" metadata="true" legend="true" tip="this
      <metadata>http://www.geomoose.org/docs/</metadata>
      <!--
      <legend>images/logo_mini.gif</legend>
      -->
    </layer>
    <layer title="City and County Boundaries" src="borders/city_labels:borders/county_labels">
```

```

    </group>
    <layer title="Weather Radar" src="iastate/nexrad-n0r" />
</group>

```

- **<layer>** – This is the other possible child of the catalog. It may optionally be a child of a group as well. The layer child creates an entry in the catalog for the user to turn on and off layers. The layer child can have additional attributes:
 - *title* – This required attribute is the text that will be shown in the Catalog to represent the layers specified in the *src* attribute.
 - *src* – This required attribute tells GeoMOOSE where to find the source for the layer by specifying the path. The source of the layer is the ‘name’ attribute specified in the **<map-source>** plus a ‘/’ combined with the name attribute of the layer as specified in the **<map-source>**’s **<layer>** child.

Recall this previous **<map-source>** example:

```

<map-source name="borders" type="mapserver" reference="true">
  <file>./demo/statedata/basemap.map</file>
  <layer name="city_labels" status="on"/>
  <layer name="county_labels" status="on"/>
  <layer name="city_poly" status="on"/>
  <layer name="county_borders" status="on"/>
</map-source>

```

there are five possible values of *src* if using only the above **<map-source>**:

```

"borders/all"
"borders/city_labels"
"borders/county_labels"
"borders/city_poly"
"borders/county_borders"

```

Of course listing *borders/all* is redundant with listing each individually. Multiple values can also be specified to turn on/off multiple layers with a single control. To do this, the values need to be divided with colons. If some of the layers in the example above were to be turned on simultaneously the *src* attribute would be set to multiple values as follows:

```

<layer title="City and County Boundaries" src="borders/city_labels:borders/city_poly"/>

```

- **<layer>**s can also have optional attributes:
 - *status* – Layers can be turned on by default by setting ‘status’ to ‘on’. This defaults to ‘off’.
- *legend* – When set to ‘true’, the legend control (a small ‘+’ icon to expand and collapse the legend) will show. This defaults to ‘false’. The configuration paramter [layer_controls.legend.on](#) impacts the default value.
- *show-legend* – When set to “true” the legend will display by default.
- *fade* – Allows the layer to be incrementally made more transparent. Fades the layer out.
- *unfade* – Allows the layer to be incrementally made less transparent. Unfades the layer (makes it opaque again).
- *minscale* – Below this scale the layer will be greyed out in the catalog. This should ideally be whatever MIN-SCALE is in the layer’s mapfile.
- *maxscale* – Above this scale the layer will be greyed out in the catalog. This should ideally be whatever MAX-SCALE is in the layer’s mapfile.
- *metadata* – When set to ‘true’, the **<metadata>** tag will be included as described below. This defaults to ‘false’.
 - **<metadata>** – URL that refers to the location of the metadata for this catalog item.

<metadata> example:

```
<layer title...>
  <metadata>http://www.geomoose.org/docs/</metadata>
</layer>
```

<toolbar>

GeoMOOSE now has a better defined toolbar. Just as the catalog and layer definitions have become more separated, so too has the toolbar and service definitions. The toolbar is specified using the <toolbar> child. The <toolbar> child has no attributes and two kind of child, the <tool> child and the <drawer> child.

<tool> children define the tool in the toolbar and tells geomoose what to do when the tool is clicked. <Tool> children have a number of attributes:

- **name** – The name is the name for the tool.
- **title** - This is the title displayed in the tool bar if the text is shown and in the mouseover popup.
- **selectable** - This is a boolean attribute. Examples of “selectable” tools: Identify, Select, Zoom. Examples of tools that are not usually “selectable”: Print, Search Parcels.
- **type** – **This attribute determines whether the tool calls an internal function or a service as defined in a ‘service’ child. The**
 - type = ‘service’ adds an additional required attribute, ‘service’ which is the name of the service specified in the <service> tag.
 - type = ‘javascript’ has no additional required attributes but the node’s value should contain javascript to be executed when the tool is selected.
 - **type = ‘internal’ adds an additional required attribute, ‘action’ which is the type of action to be taken current va**

- * zoomin – start the zoomin tool
- * zoomout – start the zoom-out tool
- * pan – start the pan tool
- * previous – step back to the previous extent
- * next – step forward to the next extent if jumped back using the ‘previous’ tool.
- * measure – start the measure length tool.
- * measurearea – start the measure area tool.
- * fullextent – zoom to the predefined initial extent of the map.
- * draw_polygon - start the polygon drawing tool for the sketch layer.
- * draw_line - start the line drawing tool for the sketch layer.
- * draw_point - start the point drawing tool for the sketch layer.
- * draw_remove - start the tool for removing shapes from teh sketch layer.
- * draw_edit - start the sketch layer shape editing tool
- * draw_edit_attributes - start the edit attribute tool. This is the tool that allows the user to change the shape colors and opacity.

<drawer> children contain tools and create a small “drop down menu” in the toolbar to compress the space required for related tools. For example the drawing tools can easily be combined into a <drawer>:

```
<drawer>
  <tool name="draw_polygon" title="Draw Polygon" type="internal" action="draw_polygon"/>
  <tool name="draw_line" title="Draw Line" type="internal" action="draw_line"/>
  <tool name="draw_point" title="Draw Point" type="internal" action="draw_point"/>
  <tool name="draw_remove" title="Remove Drawing" type="internal" action="draw_remove"/>
  <tool name="draw_edit" title="Edit Drawing" type="internal" action="draw_edit"/>
</drawer>
```

Specifying Icons for a Tool in the Toolbar

Every tool in the toolbar is given a unique ID that allows it to be referenced in CSS. The icons are all specified as backgrounds in the `htdocs/css/user_tools.css` file. Editing that will show examples of how to specify the icons for a new tool. This change was made to support the use of CSS sprites to increase the load time of GeoMOOSE.

Here is an example featuring the “Print” tool:

```
#tool-print {
    width: auto;
    background-position: 2 2;
    background-image: url('../images/toolbar/printer.png');
}

#tool-print .ToolContent {
    display: -moz-inline-box;
    display: inline-block;
    width: auto;
}

#tool-print .ToolText {
    display: block;
    padding-top: 3px;
    padding-right: 3px;

    display: -moz-inline-box;
    display: inline-block;
}
```

4.1.2 GeoMOOSE Configuration Options

Introduction

GeoMOOSE has a plethora of configuration options. All of them can be changed in the mapbook and it is possible to highly customize the interface without writing a single line of code. This document attempts to keep up with the various settings options in order to give the user more control over their GeoMOOSE installation. Organization is arbitrarily alphabetized (mostly).

How to Change a Setting

Every Mapbook has a `<configuration>` section. In the `<configuration>` section there are multiple `<param>` fields. The `<param>` fields are how settings are changed from their defaults. For example, to add some HTML to the Links bar:

```
<configuration>
  ...
  <param name="links_bar_html"><![CDATA[This is some html, with a <a href="http://www.google.c
```

```
...
</configuration>
```

The Parameters

always_scroll_zoom

Boolean. Default is false. Enables the scroll wheel for zoom all the time:

```
<param name="always_scroll_zoom">true</param>
```

catalog_name

String. Defaults to 'Catalog'. Reference the title of the tab that should be populated with the layers list:

```
<param name="catalog_name">MyCatalogTabName</param>
```

catalog.show_controls

Boolean. Defaults to true. Sets whether or not the controls below the layer names should be visible.

catalog.toggle_controls

Boolean. Defaults to true. When set to true, clicking on the layer name allows the controls below the layer to be turned on or off. When set to false the visibility of the controls is set by catalog.show_controls.

coordinate_display.latlon

Boolean. Default is true. Toggles the display of the latitude and longitude in the footer of the map:

```
<param name="coordinate_display.latlon">false</param>
```

coordinate_display.usng

Boolean. Default is true. Toggles the display of the United States National Grid Coordinates in the footer of the map:

```
<param name="coordinate_display.usng">true</param>
```

coordinate_display.xy

Boolean. Default is true. Toggles the display of the ground unit X,Y in the footer of the map:

```
<param name="coordinate_display.xy">false</param>
```

fractional_zoom

Boolean. Default is false. This allows the Map to use fractional scales like GeoMOOSE 1.X. This functionality, however, breaks tiled layers. See rounding impacts on *jump_to_scales*:

```
<param name="fractional_zoom">true</param>
```

group_checkboxes

Boolean. Defaults to true. Toggles whether group checkboxes are displayed:

```
<param name="group_checkboxes">false</param>
```

ground_units

String. Default is 'm'. Set to 'm', 'in', 'mi', 'dd', to reflect the projection being used:

```
<param name="ground_units">m</param>
```

initial_extent

Array of Floating Point Numbers. This sets the initial view for the map:

```
<param name="max_extent">-20037508.3,-20037508.3,20037508.3,20037508.3</param>
```

jump_to_scales

Associative Array. This associative array features keys that are shown in the “scale drop down” and the scale itself is the value. Application will jump to nearest fixed scale unless you set *fractional_zoom* to true (which makes tiled layers not work):

```
<param name="jump_to_scales"><![CDATA[
{
  "1:100000" : 100000,
  "1:50000"  : 50000,
  "1:24000"  : 24000,
  "1:10000"  : 10000,
  "1:5000"   : 5000
}
]></param>
```

layer_controls.fade.on

Boolean. Defaults to true. Toggles the layer control to fade the layer out:

```
<param name="layer_controls.fade.on">false</param>
```

layer_controls.unfade.on

Boolean. Defaults to true. Toggles the layer control to unfade the layer out:

```
<param name="layer_controls.unfade.on">false</param>
```

layer_controls.legend.on

Boolean. Defaults to true. Toggles whether to show the *legend* control (a small ‘+’ icon to expand and collapse the legend) by default. See also, *mapbook catalog section*

```
<param name="layer_controls.legend.on">true</param>
```

layer_controls.metadata.on

Boolean. Defaults to false. Toggles whether or not to show the link to a layer’s metadata:

```
<param name="layer_controls.metadata.on">false</param>
```

links_bar_html

HTML Text. Default displays the GeoMOOSE Stack. Changes the HTML of the “links bar” formally the “menu bar” in the interface:

```
<param name="links_bar_html"><![CDATA[
    <b>Our Stack:</b>
    <a target="_blank" ef="http://www.geomoose.org">GeoMOOSE.org</a> |
    <a target="_blank" href="http://www.mapserver.org">MapServer</a> |
    <a target="_blank" href="http://www.openlayers.org">OpenLayers</a> |
    <a target="_blank" href="http://www.dojotoolkit.org">Dojo</a>
]]></param>
```

mapbook

String. Default is ‘php/getmapbook.php’. The URL to the mapbook, this is the only configuration setting that cannot be changed by the mapbook.

max_extent

Array of Floating Point Numbers. An array, in the format of [MinX, MinY, MaxX, MaxY] that sets the maximum extents of the map. You can set this in the mapfile without using the brackets but using commas to delineate the members of the array:

```
<param name="max_extent">[-180,-90,180,90]</param>
```

(or in a different projection):

```
<param name="max_extent">-20037508.22-0037508.32,20037508.44,20037508.32</param>
```

messages.invalid_tool

String. Displayed when a drawing tool that is not defined is called. The string *%TOOL%* will be replaced with the name of the offending tool. Alerted to the user when the mapbook features the enablement of a tool that doesn't exist (like "polysquare").

messages.mapbook_param_error

String. When a param cannot be properly set due to any sort of error, this error message will be displayed. *%FIELD%* will be replaced with the "name" attribute from the param. Alerted to the user when something did not go well with the parsing the parameters.

messages.mapbook_version

String. Error presented to the user if they try to load a mapbook that is not compliant with the current GeoMOOSE version. The string *%VERSION%* will be replaced with the mapbook's version.

messages.requirement_failure

String. Message displayed when a service requirement was not met. The users is prompted with this message when they fail to fill in all of the required fields for a service.

messages.service_config_error

String. Is displayed when there is an error with the service being called.

popups.autosize

Boolean. Sets whether or not popups are autosized when added to the map.

projection

String. The projection of the map, defaults to UTM-15N (EPSG: 26915). GeoMOOSE is primarily developed in the state of Minnesota, USA, so many of our test datasets are in UTM-15N. The default Projection is this way for no other reason than that. Display is in the so-called 'web mercator':

```
<param name="projection">EPSG:3857</param>
```

reference_map.enabled

Boolean. Defaults to true. Toggle whether the reference map should be added to the map.

reference_map.height

Integer. Height of the reference map.

reference_map.width

Integer. Width of the reference map.

reference_map.maximized

Boolean. Defaults to true. Toggle whether the reference map is expanded by default.

reference_map.minimum_ratio

Integer. Minimum ratio of display between the main map and the reference map.

reference_map.maximum_ratio

Integer. Maximum ratio of display between the main map and the reference map.

scales

Array of Floating Point Numbers. The scales are actually ground units per pixel resolution settings and not true scales. This array is what sets the “scales” shown on the map.

startup_service

String. When set, the named service will be called on startup.

waiting_html

HTML Text. Default is ‘Loading...’. Used to populate the HTML of a service tab while waiting for a reload.

zoomto

Associative Array of Associative Arrays. This is the settings that populate the “jump to” drop downs. Best to explain this by example. If you have difficulties please email the mailing list:

```
<param name="zoomto[' Jump To: ']"><![CDATA[
    {
        'Dakota County' : [521238.614537864,4924218.86673578,473921.947801381,4974430.368850],
        'Parcel Data' : [497205.409367,4923984.423582,477595.805945,4941970.52988],
        'Full State of MN' : [189783.560000,4816309.330000,761653.524114,5472346.500000]
    }
}]></param>
```

Deprecated Parameters

default_tab

String. Defaults to ‘Catalog’. References the title of the tab that should be shown by default::

<param name="default_tab">Catalog</param>

drawing_tools.default_fill

Color. Defaults to 'green'. Sets the default fill for all sketches.

drawing_tools.default_opacity

Floating Point Number between 0 and 1. Defaults to .8. Sets the opacity of the sketches.

drawing_tools.default_stroke

Color. Defaults to 'red'. Sets the default stroke for all sketches.

layer_controls.cycle.on

Boolean. Defaults to false. Toggles whether the layer control to refresh the layer every layer_controls.cycle.seconds seconds.

layer_controls.cycle.seconds

Floating Point Number. Defaults to 10. The number of seconds between layer refreshes when used with the cycle tool.

layer_controls.down.on

Boolean. Defaults to true. Toggles the layer control to move the layer down on the image stack::

layer_controls.up.on

Boolean. Defaults to true. Toggles the layer control to move the layer up on the image stack::

layer_controls.refresh.on

Boolean. Defaults to false. Toggles the layer control to allow the user to manually refresh the layer.

mapserver_url

String. MapServer URL used when calling type="mapserver" map-source's. mapserver_url is now configured through conf/local_settings.ini.

messages.invalid_response

String. Message displayed when a service does not return a valid answer from the server. Deprecated, removed from config.js

messages.mapbook_required

String. The error shown when a required <param> setting is missing in the mapbook. The string '%FIELD%' will be replaced with the required field's name. 'Deprecated, removed from config.js

messages.service_not_found

String. Message displayed when a service is called but is not defined in the mapbook. The string '%SERVICE%' will be replaced with the name of the invalid service. 'Deprecated, removed from config.js

scale_line.enabled

Boolean. Defaults to true. Toggle whether the scale line is shown on the map. Scale_Line.* are now configured in a ScaleLine.js extension.

scale_line.bottom_units

String. Defaults to 'mi'. Possible unit values are 'm', 'in', 'km', 'ft', 'mi'. Scale_Line.* are now configured in a ScaleLine.js extension.

scale_line.top_units

String. Defaults to 'ft'. Possible unit values are 'm', 'in', 'km', 'ft', 'mi'. Scale_Line.* are now configured in a ScaleLine.js extension.

scale_line.width

Integer. Defaults to 200. Defines the width of the scale line on the map. Scale_Line.* are now configured in a ScaleLine.js extension.

show_service_settings_in

String. Defaults to 'Services'. Reference the title of the tab that should display the prompts for a service when it is called. 'Deprecated, removed from config.js

tabs

Associative Array. The keys of the associative array are the titles of the tabs and the values is the DIV that represent the tab. This would be an example setting that is different than the default:

```
<param name="tabs"><![CDATA[
{
    'Layers' : 'layers-tab',
    'Information' : 'info-tab'
}]]></param>
```

4.1.3 URL Startup Arguments

GeoMOOSE has a number of parameters that can change its startup behavior.

debug

Sets whether GeoMOOSE is in debug mode. Set this parameter to any non-zero value in order to see additional logging information. This is really only useful for development and should probably only be used in conjunction with a “_dev.html” file.

extent

Sets a different startup extent. The extent should be formatted as *min-x,min-y,max-x,max-y* in the ground units of the map. Example:

```
geomoose.html?extent=-180,-90,180,90
```

mapbook

Bypass *php/getmapbook.php*. By default GeoMOOSE will read the mapbook from the file system using *getmapbook.php*, other installations, especially those needing to serve multiple mapbooks can do so by setting the *mapbook* parameter. Example:

```
geomoose.html?mapbook=../mapbooks/wms_only.xml
```

off

Turns off default layers. Uses GeoMOOSE’s path convention. Example:

```
geomoose.html?off=parcels/parcels;county/all
```

on

Turns on layers. This follows the same form as the off parameter but turns layers on.

4.1.4 Local_settings.ini Reference Guide

The *settings.ini* file found in the conf folder is a text document used as the general configuration file for a GeoMOOSE application. It configures things such as projection, paths, mailing labels. For an example of a *settings.ini* file, please refer to the GeoMOOSE demo and the file in the conf folder. This reference guide explains the optional settings.

You should never, ever modify settings.ini

This is very important. *settings.ini* is managed by GeoMOOSE. If you upgrade the application in the same tree as your current application you will lose all of your changes. This can get annoying because replacing the new *settings.ini* with your old one could substantially break the application because some of its defaults will be missing. GeoMOOSE is configured to override *settings.ini* with *local_settings.ini*. There are two example files that you might want to use for *local_settings.ini*; either *ms4w_local_settings.ini* or

`unix_local_settings.ini`. If you want to change these settings, leave `settings.ini` alone and change them in `local_settings.ini`. Think of `localsettings.ini` as an override to the defaults in `settings.ini`. Anything that is in `settings.ini` can go into `localsettings.ini`.

[defaults]

- `mapbook` – location of default mapbook

[map]

- `projection` – EPSG code of coordinate system for the map

[paths]

- `server_name` - name of your web server. For development purposes this can be left set as “localhost”. However, for a public webserver, the `server_name` parameter should be changed to the name of that server.
- `root` – root for the map (only configured through `local_settings.ini`)
- `temp` – web accessible location of the temp files relative to the root of the domain (only configured through `local_settings.ini`)

[identify]

This setting specifies which metadata parameters in the mapfiles are used for the identify service output templates.

- `identify_header=identify/header.html`
- `identify_footer=identify/footer.html`
- `wms_header=identify/wms_header.html`
- `wms_record=identify/wms_record.html`
- `wms_footer=identify/wms_footer.html`

[select]

- `highlight_map` – this setting specifies which mapfile should be used to symbolize the selected or highlight feature on the map

[itemquery]

This setting specifies which metadata parameters in the mapfiles are used for the itemquery service output templates.

- `itemquery_header=itemquery/header.html`
- `itemquery_footer=itemquery/footer.html`
- `itemquery_miss=itemquery/miss.html`

[query]

This setting specifies the output of a query with no results returned.

- query_miss=itemquery/miss.html

[mailing_labels]

These setting specify formatting options for the labels generated from the mailing labels service. Specifies how many rows/columns to print per page:

- label_rows=10
- label_columns=3

Specifies page layout information (only applies to PDF Format):

- label_origin_x=.25
- label_origin_y=.5
- label_width=2.5
- label_height=1

Specifies Font for the label output (only applies to PDF Format):

- label_font=Arial
- label_font_size=8

Specifies the settings for the actual lines of the label. The items in parentheses relate to the data source in the mailing labels service. Items must be enter with parentheses and case sensitive field name.

- label_lines=3
- label_line_1=%OWNER_NAME%
- label_line_2=%BLDG_NUM% %STREETNAME% %SUFFIX_DIR% %STREETTYPE%
- label_line_3=%CITY%, MN %ZIP%

If this is set to “true” then any blank lines in the labels will be “collapsed” in the PDF output.

- label_lines_collapse=true

[print_formats]

Sets if print format will appear in the print formats tab. The format will appear if “1” and will not appear if “0”

- print_image=1
- print_html=1
- print_pdf=1

[html_printing]

Sets the template to use, image width and height for map when selecting the html format for printing output.

- html_template=./print/default_template.html
- html_image_width=800

- `html_image_height=700`

4.1.5 Notes on Internationalization

Changing the Encoding

Changing the input and output encodings in GeoMOOSE is done via the local settings.

In the `conf` directory open the `local_settings.ini` file and add the following lines to change GeoMOOSE to UTF8:

```
[encodings]
input-encoding=UTF-8
output-encoding=UTF-8
```

What Has Been Tested

This has been only really tested with UTF-8 and Latin1 encodings. None of the core team speaks a non-Romanitic-influenced language.

4.1.6 Service Communication Protocol Reference

GeoMOOSE service's 'talk back' to GeoMOOSE using either pure HTML or by using the GeoMOOSE 2.0 Service XML format. If a service returns content of type `text/html` then GeoMOOSE will simply display such content in the 'Results' tab. This type of return has very limited potential as it does not provide a way to control functions in the GeoMOOSE interface. In GeoMOOSE 1.0 we addressed this by using an alternate GeoMOOSE namespace embeded in the HTML. While this was a very technical and generally 'neat' solution it was very difficult to explain and XML was quite under powered as a way to allow services to properly manipulate the user interface. GeoMOOSE 2.0 fixes this by using a new XML format that allows the services to specify Javascript to be run upon loading and HTML to display in the interface. This communication protocol is useful for developers writing their own services.

<results>

The document element is mainly ignored for now. But it has three children:

- `<script>`
- `<popup>`
- `<html>`

<script>

Contains a single CDATA section that contains Javascript to be executed upon loading the interface. Javascripts in the `<script>` tag SHOULD NOT call methods or depend on attributes in GeoMOOSE objects. GeoMOOSE changes all of the time and calling specific methods (like those in the Catalog) may change calling format or even names. There is a limited set of functions that will be supported and whose API will not change without significant notice to users and developers. Those are contained in the single "GeoMOOSE" object. Other examples using this protocol can be found in the layer query templates such as `/maps/landrecords/parcels_popup.html` in the GeoMOOSE demo.

<popup>

Popups contain a single CDATA section that contains HTML to be displayed in a popup. Popups have four attributes:

- x – The “east/west” position of the popup in ground units.
- y – The “north/south” position of the popup in ground units.
- width – The pixel width of the popup.
- height – The pixel height of the popup.

Example:

```
<popup x="-93.129" y="42.111" w="200" h="200">
  This is a cool popup!
</popup>
```

<html>

Contains a single CDATA section that contains HTML to be displayed in the Results Tab.

Example:

```
Content-type: application/xml
<results>
  <script><![CDATA[
    GeoMOOSE.refreshLayers('/wms/metro/stream_net_1');
    alert('I also made an alert!');
  ]]></script>

  <html><![CDATA[
    I refreshed a layer!
  ]]></html>
</results>
```

Additional resources

See also, *<service>*.

4.1.7 What happened to the Menubar?

For the most part it is gone. There is still a place for the menubar to exist, however, there is no longer an elaborate separate code base to make a javascript powered menu function. If you would like to do this, the ability is still there. We removed this functionality for a few reasons.

1. We were maintaining the code. The menubar.js and related configuration was maintained by the GeoMOOSE authors and like much of the Javascript code base, minor browser changes could effect the function and break it. Even worse, when it did break, there were very few people who noticed. This gave us an indication that the feature was under utilized.
2. A secondary configuration file was slow to load. To have two configuration files at startup was causing two Ajax requests to be made at startup. This noticeably slowed the loading time and would cause skews in how the interface rendered.

4.1.8 Using the Reference Map

The display in the reference map is done on a service-by-service basis. This document attempts to cover the implementation of various reference map tasks.

Reference Layers Display with the Main Map

This is the common GeoMOOSE reference map. Under this scenario a source is included in both the main map and the reference map. To do this simply add set the `reference` attribute to `true` in the `map-source`:

```
<map-source name="basemap" type="mapserver" reference="true">
  <file>./demo/statedata/basemap.map</file>
  <layer name="county_borders"/>
  <layer name="county_labels"/>
  <layer name="city_poly"/>
  <layer name="city_labels"/>
  <layer name="USGSGagingStations"/>
</map-source>
```

Now whenever one of these layers is made visible, it will also be made visible in the reference map. These layers would be controlled in catalog just as they would if they were not included in the reference map.

A Fixed Reference Map set of Layers

There are situations in which having an extremely dynamic reference map is wasteful. In this case, it's desirable to setup the reference map to have a fixed set of reference layers. To make a source display all of it's layers in the reference map by default the `reference` attribute must be set to `true` at both the source *and* layer level:

```
<map-source name="states" type="mapserver" reference="true">
  <file>./demo/statedata/statesUTM15N.map</file>
  <layer name="all" reference="true"/>
</map-source>
```

To prevent this layer from changing DO NOT include it in the catalog. If you would like to have this layer in the main map and fixed in the reference map you should create TWO `map-source` entries. One entry should be configured similar to above, for the reference map, and another should be configured with a different name and without any of the `reference` attributes set to `true`. This second `map-source` will be used in the main map (and consequently in the catalog).

Mixing and Matching Layers

The two setups above involved only a single layer in their demonstrations. The Reference Map could have a mix of fixed and dynamic layers. The configurations above just show small examples of how the maps can be setup.

Starting up the Reference Map Collapsed

By default the reference map starts open, this can be changed with the `"reference_map.maximized"` parameter:

```
<param name="reference_map.maximized">false</param>
```

Getting Rid of the Reference Map

In the case that a reference map is undesirable:

```
<param name="reference_map.enabled">false</param>
```

4.1.9 GeoMOOSE User Extensions

GeoMOOSE User Extensions are the latest way to customize a GeoMOOSE installation. While it has been common practice to modify the root of the code to create various custom tools, that has left some installations stagnant and unable to smoothly upgrade or integrate with new GeoMOOSE releases when they have become available. To address these concerns, we have developed GeoMOOSE User Extensions. These extensions will never be clobbered by a GeoMOOSE upgrade. However, extensions written that access core library functions may break as the underlying library API's may change over time.

What is the purpose of User Extensions?

The purpose of user extensions is for the customizer of a site to, “go crazy.” It allows for users to add any functionality they wish to the user interface without forcing their code to break away from the main code base.

But Duck, how do I make one?

Good Question. Here is a brief tutorial...

Step 1: Come up with an idea for a user extension.

Say you want a disclaimer to show at the beginning of the application. Let's just say that disclaimer is, “Hello, World!”

Step 2: Write the User Extension

User extensions are stored in the “extensions/” folder under “htdocs/”. For this example we'll create a file called, “Disclaimer.js”. In “Disclaimer.js” we need to add the basic outline of the class by inheriting from the base user extension class. The file should look like this:

```
DisclaimerExtension = new OpenLayers.Class(GeoMOOSE.UX.Extension, {  
    CLASS_NAME : "DisclaimerExtension"  
});
```

This sets up a blank extension that does nothing. Let's break down the code as to what's happening:

- *DisclaimerExtension* - This defines the Javascript name for your extension. You'll want to give your classes long descriptive names so they do not clobber other potential global variables. Those who are more advanced are probably asking “Why didn't you name space them?” and the answer is: “To make it easier on people who don't necessarily understand what ‘namespacing’ is.”
- *new OpenLayers.Class* - This is the OpenLayers class constructor.
- *GeoMOOSE.UX.Extension* - The base GeoMOOSE User Extension class. By putting this class as the first parameter we're going to “inherit” all of the methods from the base class.
- *CLASS_NAME* : ‘DisclaimerExtension’ - This configures the class so it knows its own name.

Now we need to have the class actually do something. The initial function that is called when an extension is loaded is called “load.” So, now we define the load method and have it display our disclaimer using “alert.”

```
DisclaimerExtension = new OpenLayers.Class(GeoMOOSE.UX.Extension, {
    load: function() {
        alert("Hello, World!");
    },

    CLASS_NAME: "DisclaimerExtension"
});
```

Finally, we need to tell GeoMOOSE to load the extension at startup by adding this at the end of the file:

```
GeoMOOSE.UX.register('DisclaimerExtension');
```

The completed “Disclaimer.js” should look like this:

```
DisclaimerExtension = new OpenLayers.Class(GeoMOOSE.UX.Extension, {
    load: function() {
        alert("Hello, World!");
    },

    CLASS_NAME: "DisclaimerExtension"
});

GeoMOOSE.UX.register('DisclaimerExtension');
```

Step 3: Add the Extension to geomoose.html

- Open “geomoose.html” in a text editor.
- Find “</head>”
- On the directly above “</head>” add:

```
<script type="text/javascript" src="extensions/Disclaimer.js"></script>
```

Step 4: Enjoy!

Now you should see “Hello, World!” when the application starts.

Other Examples

An example of an User Extension which updates the interface on a periodic basis can be found in the “extensions/” directory with GeoMOOSE 2.2, the file is called “ColorChanger.js”.

4.1.10 GeoMOOSE Structure

Once GeoMoose has been installed it will have six subdirectories. Files are discussed in more detail in documentation.

art

The art directory is used to store standard GeoMoose icons and logos. Typically not used by user.

desktop_icons

Original GeoMoose icons.

logo_2011

New logos from 2011 in a variety of formats.

attic

Like all attics, this hold old stuff that isn't used very often. Typically not used by user.

conf

The conf directory contains various configuration controls and templates for GeoMoose. Frequently used by user.

- **Files:**
 - mapbook.xml - The map control XML file.
 - local_settings.ini - Several ini file exist for control file paths for unix, ms4w,svn.
 - settings.ini - Initial file for control.

editor

Directory containing editor support files.

feature-report

Directory containing feature report templates.

print

Directory containing printing templates. Rarely used by user.

htdocs

- **Files:**
 - geomoose.html - starting HTML for GeoMoose.

build

The compiled java script for GeoMOOSE.

css

css files for GeoMoose.

cursors

Standard cursors used by GeoMoose.

extensions

Javascript extension location.

geomoose

The GeoMoose javascript library.

images

Library of standard images used by GeoMoose.

lib

Basic libraries used by GeoMoose (dojo, openlayers).

php

Php services library.

projections

Projection files used by GeoMoose.

skins

Standard skins (colors etc) used by GeoMoose.

maps

Directory of data, mapserver control files, symbol library and display html templates for layers. Frequently used by user.

- **Files:**

- geomoose_globals.map - global map files for mapserver layers
- temp_directory.map - locates temporary directories for Mapserver Mapfile

demo

Directory containing data map files and templates for demo.

development

Directory used in assisting team with development. Typically not accessed by user.

fonts

Font library for GeoMoose mapserver maps. Rarely used by user.

symbols

Symbol.sym file used for mapserver feature layers and graphic images for road symbols. Rarely used by user.

sphinx-docs

This documentation and tools to make it (sphinx). Typically not used by user.

tools

Python scripts (tools) used to build the GeoMOOSE packaging. Typically not used by user.

4.2 Vector Layer Editing

4.2.1 Using Vector Layers

Configuring Vector Layers For Users

Configuration of vector layer is covered elsewhere. This is about how to use them.

1. Turn on the Layer of Interest

In the catalog, click on the check box to turn on the layer to be viewed or edited. If this is a layer fed by a server, such as WFS, there may be a small delay while the feature definitions are downloaded and rendered.

2. Activating the Layer

For vector layers to be used, they need to be “activated” by the user. By default GeoMOOSE uses a small green circular button with a white check-mark embedded within it. Click on this button. If using the default styling the background of this layer will be highlighted with a gold gradient.

3. Using the Layer Tools

Once a layer is activated, then the Layer Tools can be used to manipulate that layer. Layers may or may not support specific tool. If a tool is not supported by a layer, when selected, it will throw an error with an appropriate alter message.

4. Saving Changes

If a layer supports saving changes made to it then use the “Save Changes” menu under the “Layer Tools” menu to save those changes.

4.2.2 Using GeoServer with GeoMOOSE

This is a basic walk-through using to use GeoMOOSE with GeoServer and PostGIS. This is not meant to be comprehensive and the reader should have some knowledge of GeoServer and PostGIS. GeoMOOSE comes configured with a WFS-T example, using the U.S. Census Bureau’s Places layer.

Setup GeoServer

The best documentation on setting up GeoServer comes from GeoServer itself.

And here is the [GeoServer Documentation](#).

Install and startup a fresh GeoServer instance.

Download the Places Layer from the Census Bureau

Download the file [here](#). This is the “place” file for the state of Minnesota. “Places” are a Census terminology for incorporated cities, townships, and equivalents.

Install the Places Shapefile into PostGIS

There are a few good methods for doing this, *ogr2ogr* and *shp2pgsql*. Whichever method is used, the layer should be named *census_cities* for the purposes of this example.

A typical *ogr2ogr* string will look similar to this:

```
ogr2ogr -f PostgreSQL 'PG:dbname=gis2 user=gis password=super_gis' tl_2010_27_place10.shp tl_2010_27_
```

Configure GeoServer

Create a *GeoMOOSE_Testing* Workspace

1. In the left control-panel column of GeoServer, under the “Data” header, click on “Workspaces”.
2. Click the green “+” button.
3. Name the work space “GeoMOOSE_Testing” with the namespace “geomoose”.

Add a *census_plc* Store

1. In the left control-panel column of GeoServer, under the “Data” header, click on “Stores”.
2. Click the green “+” button. Create a new PostGIS Database store.
3. Use the following settings:
 - Workspace: GeoMOOSE_Testing

- Data Source Name: census_data
- Description: Census Cities for Testing
- Enabled: Should be checked to indicate “true”.
- The rest of the settings are for PostgreSQL, please fill them out to match the information you used to load the shapefile. If they do not match the next step will end in a miserable disaster.

Add the Layer

1. In the left control-panel column of GeoServer, under the “Data” header, click on “Layers”.
2. Click the green “+” button.
3. Select “GeoMOOSE_Testing:census_data”
4. A list should appear, next to *census_cities* click the “Publish” button.
5. The information should fill in automatically. Click “Save” at the bottom.

Setup the Layer to be Served Transactionally

1. In the left control-panel column of GeoServer, under the “Data” header, click on “Layers”.
2. You should now see a list featuring the “census_cities” layer. Click on the layer name to view the settings.
3. The basic information for the layer should now be shown. Click on the “Publishing” tab. Ensure that “enabled” is checked.

Set GeoServer to Server Transactionally

1. In the left control-panel column of GeoServer, under the “Services” header, click on “WFS”.
2. Scroll to the section labelled “Service Level”.
3. Set the “Service Level” radio button to “Complete”.
4. Scroll to the bottom and click “Submit”.

Setting up a WFS-T alias for Testing

This is probably the largest amount of witchcraft and wizardry required to complete a WFS-T server. Because the WFS-T server requires AJAX it is critical that the WFS-T server is on the exact same domain as the GeoMOOSE application. By default GeoServer will attach to port 8080, while Apache attached to port 80. This is normal and really does not need to be changed for testing purposes. The GeoMOOSE demo assumes that GeoServer is being proxied by Apache using the following configuration:

```
ProxyPass /geoserver http://localhost:8080/geoserver
ProxyPassReverse /geoserver http://localhost:8080/geoserver
```

There are other methods for doing this using Nginx or IIS but that is out of the scope of this document. The author offers the Apache configuration as a sample and only out of kindness since he is using the aforementioned configuration.

Mapbook Setup

The Mapbook currently features a fragment which supports the above configuration of GeoServer. Here is that fragment:

```
<map-source name="census_cities" type="wfs" status="on">
  <style type="stylemap"><![CDATA[
    {
      "strokeColor" : "#00ff00",
      "label" : "${nameIsad}"
    }
  ]]></style>

  <url>/geoserver/GeoMOOSE_Testing/ows</url>

  <attribute name="geoid10" type="text" label="ID:" default-value="27999"/>
  <attribute name="nameIsad10" type="text" label="Name:"/>

  <feature-namespace>geomoose</feature-namespace>
  <feature-type>census_cities</feature-type>
  <geometry-name>wkb_geometry</geometry-name>
  <schema><![CDATA[http://localhost:8080/geoserver/GeoMOOSE_Testing/wfs?service=WFS&version=1.1.0]
</map-source>
```

4.2.3 Using TinyOWS with GeoMOOSE

This is a basic walk-through on how to use GeoMOOSE with TinyOWS and PostGIS.

More Information on TinyOWS

From the TinyOWS Main Page,

TinyOWS is a lightweight and fast implementation of the OGC WFS-T specification. Web Feature Service (WFS) allows to query and to retrieve features. The transactional profile (WFS-T) allows then to insert, update or delete such features.

From a technical point of view WFS-T is a Web Service API in front of a spatial database. TinyOWS is so deeply tighed to PostgreSQL/PostGIS.

More [here!](#)

Windows Users can find TinyOWS for MS4W [here](#).

Assumptions of this How To

1. This document assumes you have been able to install GeoMOOSE, PostgreSQL, and TinyOWS.
2. For purposes of this example, we'll be using the database `gis`, username `gis` and the password `super_gis`.
3. For our example here, the url for TinyOWS is: `http://localhost/mapserver/cgi-bin/tinyows`

Download the Places Layer from the Census Bureau

Download the file [here](#). This is the “place” file for the state of Minnesota. “Places” are a Census terminology for incorporated cities, townships, and equivalents.

Install the Places Shapefile into PostGIS

There are a few good methods for doing this, *ogr2ogr* and *shp2pgsql*. Whichever method is used, the layer should be named *census_places* for the purposes of this example.

A typical *ogr2ogr* string will look similar to this:

```
ogr2ogr -f PostgreSQL 'PG:dbname=gis user=gis password=super_gis' t1_2010_27_place10.shp t1_2010_27_p
```

Configure TinyOWS

TinyOWS is configured using a small XML file.

The file we used for this example is:

```
<tinyows online_resource="http://localhost/mapserver/cgi-bin/tinyows"
  schema_dir="/usr/local/tinyows/schema/"
  check_schema="0"
  log="/tmp/tinyows.log"
  log_level="15"
  check_valid_geom="0">

  <metadata name="TinyOWS Server"
    title="TinyOWS Server - Demo Service" />

  <pg host="localhost" user="gis" password="super_gis" dbname="gis" port="5432" />

  <layer retrievable="1"
    writable="1"
    server="http://localhost/"
    ns_prefix="census"
    ns_uri="http://localhost/geomoose/census_places"
    name="census_places"
    title="Census Places" />
</tinyows>
```

Mapbook Setup

The Mapbook currently features a fragment which supports the above configuration of GeoServer.

That `<map-source>` entry can be removed and replaced with this fragment:

```
<map-source name="census_cities" type="wfs">
  <style type="stylemap"><![CDATA[
    {
      "strokeColor" : "#00ff00",
      "label" : "${namesad10}"
    }
  ]]></style>

  <url>/mapserver/cgi-bin/tinyows</url>
```

```
<attribute name="geoid10" type="user" label="ID:" default-value="27999"/>
<attribute name="namelsad10" type="user" label="Name:"/>

<feature-namespace>http://localhost/geomoose/census_places</feature-namespace>
<feature-type>census_places</feature-type>
<geometry-name>wkb_geometry</geometry-name>
<schema><![CDATA[http://localhost/mapserver/cgi-bin/tinyows?service=WFS&version=1.1.0&request=

<popup-template><![CDATA[
<div style="font-size: 1.5em">${namelsad}</div>
Area of Land: ${aland}<br>
Area of Water: ${awater}<br>
<br>
<a href="https://www.census.gov/2010census/popmap/ipmtext.php?fl=${statefp}:${geoid}" target=
]]></popup-template>
</map-source>
```

Example Installation

This is the exact input Duck used to get the demo working on his Macbook. Where necessary comments have been included to describe actions not done directly in the terminal.

```
# create the demo database
createdb gis -T postgis_template -O gis

psql "dbname=gis"
gis=# alter table geometry_columns owner to gis;
ALTER TABLE
gis=# alter table spatial_ref_sys owner to gis;
ALTER TABLE
gis=# alter table geography_columns owner to gis;
ALTER TABLE

mkdir /tmp/tinyows
cd /tmp/tinyows

wget ftp://anonymous@ftp2.census.gov/geo/tiger/TIGER2010/PLACE/2010/tl_2010_27_place10.zip

unzip tl_2010_27_place10.zip

ogr2ogr -f PostgreSQL 'PG:dbname=gis user=gis password=super_gis' tl_2010_27_place10.shp tl_2010_27_p

vim /srv/mapserver/cgi-bin/tinyows.xml
# then copy-and-paste tutorial contents.

# I then edited my local geomoose mapbook and added the Mapbook fragment
# from above
```

4.3 Service Documentation

GeoMoose is configured with several standard services. These services may be modified to allow customizations for a specific site.

4.3.1 Standard Services

birdseye

birdseye – a service called from the toolbar that provides a link to the Microsoft Bing! Maps.

```
<service name="birdseye" target="_blank" title="Birds Eye View">
  <url>php/birdseye.php</url>
  <step type="spatial" name="xy" line="false" polygon="false" jump-start="true" default="point" >
    <header>
      Click on the map to view the area using Microsoft Bing! Maps.
    </header>
  </step>
</service>
```

buffered_select

buffer_select – a service called from the toolbar that allows users to select parcels using a mouse

```
<service name="buffered_select" title="Select Features">
  <url>php/select.php</url>

  <!-- Send a selection shape + the visible layers list to the service -->
  <step type="spatial" showTools="true" name="shape" line="true" polygon="true" point="true" >

    <header><![CDATA[Create a selection area by clicking on the map.]]></header>

    <input type="visiblelayers" name="layers"/>

    <!--
      Option values should be the mapbook path to the layer.
      This only supports 'mapserver'-type layers.
    -->
    <input type="select" name="select_layer" title="Select features from:">
      <option value="parcels/parcels">Parcels</option>
    </input>

    <!--
    <input type="user" name="select_buffer" title="Buffer Selected Features (m)">0</input>
    -->

    <input type="length" name="selection_buffer" title="Buffer Selection Shape">0</input>
    <input type="select" name="query_layer" title="Using Features In">
      <option value="">No Layer</option>
      <option value="parcels/parcels">Parcels</option>
    </input>
    <input type="projection" name="projection"/>

    <footnote><![CDATA[
    ]]></footnote>

  </step>
</service>
```

buffered_select_followup

buffered_select_followup – service called within the layer results (html template) for a selected set to buffer the selected set.

```
<!-- this is called after a standard select in order to buffer the previous selection -->
<service name="buffered_select_followup" title="Buffered Select">
  <url>php/select.php</url>
  <step type="input">
    <input type="hidden" name="shape"/>
    <input type="hidden" name="select_layer"/>
    <input type="hidden" name="query_layer"/>
    <input type="hidden" name="selection_buffer"/>
    <input type="length" name="shape_buffer" title="Buffer Features By: "/>
    <input type="projection" name="projection"/>
  </step>
</service>
```

feature_report

feature_report – featurereport: a service not called from the tool bar that is called from a feature layer html.

```
<service name="feature_report" display="false" keep-others="true">
  <url>php/feature_report.php</url>
  <input type="hidden" name="layers" value="lmic/fsa"/>
  <input type="hidden" name="src"/>
  <input type="hidden" name="PIN"/>
</service>
```

geocode_address

geocode_address – a service called from the toolbar that allows users to GeoCode an address using googles geocode services. The google service requires a google key as used in the second input.

```
<service name="geocode_address" title="Geocode Address">
  <url>php/geocode.php</url>
  <step type="input">
    <input type="user" name="address" title="Enter Address: "/>
    <input type="hidden" name="googlekey" value="ABQIAAAA4Q-VLyIpwp3L8M9DIzKb2BT2yXp_ZAY8" />
  </step>
</service>
```

identify

identify – a services called from the toolbar that allows users to identify visible features

```
<service name="identify" title="Identify" display="true">
  <url>php/identify.php</url>
  <step type="spatial" name="shape" line="false" polygon="false" jump-start="true" default="polygon">
    <header>
      Click on the map to see more detailed information.
    </header>
    <input type="visiblelayers" name="layers"/>
    <input type="projection" name="projection"/>
  </step>
</service>
```

```

    </step>
</service>

```

popups

popups – This service is not available at this time.

print

print – a service called from the toolbar that allows printing to an IMAGE, HTML, or PDF. The print service uses templates stored in the `geomoose2/conr/print` directory.

```

<service name="print" title="Print Map" keep-others="true">
  <step type="input">
    <url>php/print.php</url>
    <input type="print_info" name="layers"/>
    <input type="extent" name="extent"/>

    <input type="user" name="title" title="Map Title">Map</input>
    <input type="hidden" name="date" title="Map Date">true</input>

    <input type="select" name="template" title="Output Template: ">
      <option value="letter_landscape">Letter - Landscape</option>
      <option value="letter_portrait">Letter - Portrait</option>
      <option value="poster_landscape">11" x 17" - Landscape</option>
      <option value="poster_portrait">11" x 17" - Portrait</option>
    </input>

    <input type="select" name="quality" title="Image Quality: ">
      <option value="2">Higher</option>
      <option value="3">Highest</option>
      <option value="1">Standard</option>
    </input>

    <input type="select" name="scale" title="Print Scale: ">
      <option value="map">Current Map Scale</option>
      <option value="1000">1:1000</option>
      <option value="5000">1:5000</option>
      <option value="10000">1:10000</option>
    </input>
  </step>
</service>

```

search_parcel

search_parcel – This service called from the tool bar menu allows you to query and search for parcels.

```

<service name="search_parcel" title="Search">
  <url>php/query.php</url>
  <step type="input">
    <input type="hidden" name="highlight" value="true"/>
    <input type="hidden" name="mode" value="search"/>

    <input type="hidden" name="layer0" value="parcels/parcels"/>
    <input type="hidden" name="template0" value="itemquery"/>
  </step>
</service>

```

```
<input type="select" name="fieldname0" title="Search By:">
  <option value="OWNER_NAME">Owner</option>
  <option value="PIN">Parcel ID</option>
</input>
<input type="select" name="comparator0" title="That: ">
  <option value="like-icase">Contains</option>
  <option value="right-like-icase">Begins With</option>
  <option value="eq-str">Matches Exactly</option>
</input>
<input type="user" name="value0" title=""/>

<input type="hidden" name="fieldname1" value="FIN_SQ_FT"/>
<input type="select" name="operator1">
  <option value="or">OR</option>
  <option value="and">AND</option>
</input>
<input type="select" name="comparator1" title="Having Fin. Sq. Ft. ">
  <option value="gt">Greater Than</option>
  <option value="eq">Equal To</option>
  <option value="lt">Less Than</option>
</input>
<input type="user" name="value1" title=""/>
</step>
</service>
```

streetview

streetview – a service called from the toolbar that provides a link to Google Street View

```
<service name="streetview" target="_blank" title="Street View">
  <url>php/streetview.php</url>
  <step type="spatial" name="xy" line="false" polygon="false" jump-start="true" default="point">
    <header>
      Click on the map to view the point in Google Street View.
    </header>
  </step>
</service>
```

query

General Description

GeoMOOSE provides a general query service that allows the user to query multiple layers and fields through a single CGI query. Results can be optionally highlighted.

The Query service is intended to replace the `itemquery.php` service and eventually the identify service. Both ItemQuery and Identify may be dropped as of the official 2.0 release. Currently Shapefiles, OGR, Oracle Spatial, and PostGIS are the only supported layer types for the General Query Service. If you plan to search a layer other than you may consider modifying `query.php` to meet your needs or write another service entirely.

Important settings.ini Variables

- `query_header` - This will be prepended to the HTML results.

- `query_footer` - This will be append to the HTML results.
- `query_miss` - This is the message that will be displayed when no results are found.

As a compatibility step, the current version of `query.php` will default to using the “`itemquery_...`” versions of the above settings when it cannot find the proper “`query_...`” settings.

Parameters

The parameters for this service are unique because they can be variable based on the number of fields and layers that will be queried by the service.

- `mode` is a hidden input that should always be set to “search”.
- `highlight` is a hidden input that is a boolean. “True” will mean the selected features will be highlighted, “false” means they will not be highlighted.
- `zoom_to_first` is a hidden input that is a boolean. When set to “true” GeoMOOSE will always zoom to the first feature found in the search, when set to “false” GeoMOOSE will not exhibit this behaviour.
- `layer[0..n]` is the Mapbook path of the layer to search. Only the 1st layer (layer0) is required. If no other layers are specified then it is assumed that the target of all of the following inputs are layer0.
- `value[0..n]` the value which should be searched against. Only the 1st value (value0) is required. If no other values are specified then it is assumed that the initial value will be used to search all other fieldnames.
- `fieldname[0..n]` is the name of the field in the table or DBF to search against. This is required for every field.
- **`comparator[0..n]` is the name of the comparison operator. Valid comparison operators include:**
 - “eq-str” - String equals match.
 - “like” - Case-sensitive contains operator. The equivalent SQL is “fieldname like ‘%value%’”.
 - “right-like” - Case-sensitive “begins with” operator. The equivalent SQL is “fieldname like ‘value%’”
 - “left-like” - Case-sensitive “ends with” operator. The equivalent SQL is “fieldname like ‘%value’”
 - “like-icase” - Case-insensitive “like.”
 - “right-like-icase” - Case-insensitive “right-like.”
 - “left-like-icase” - Case-insensitive “left-like.”
 - “eq” - Equals operator. This is used for numeric operations.
 - “gt” - Greater-than operator.
 - “lt” - Less-than operator.
 - “ge” - Greater-than or equal-to operator.
 - “le” - Less-than or equal-to operator.
 - “in” - Inputs a semi-colon separated list of values and performs an equals match with them.
- **`operator[1..n]` is the type of connecting operator between searching predicates. There is no “operation0” as it is the start of**
 - “and” - Require both this statement (n) and the previous one (n-1).
 - “or” - Require either this statement (n) or the previous one (n-1).
 - “nand” - Require this statement to be false (n) and the previous one to be true (n-1).
 - “nor” - Require this statement to be false (n) or the previous one to be true (n-1).

- *blanks[0..n]* is a hidden boolean tied to the value[0..n]. If set to “true” blank values will mean this predicate is ignored, when set to “false” then a blank value will be evaluated.
- *template[0..n]* specifies the template to use. This template is specified in the LAYER’s METADATA tag. Only template0 is required. If no other templates are specified it will try to use the template named in template0 for all layers. For example, to mimic the itemquery functionality the “itemquery” template is used in the examples below. It is possible to use other templates, such as *identify_record*, and is even possible to provide this as a drop-down option to the user. This would allow a single search to have different result outputs.

Sample Mapbook Service Definitions

This is a basic example that emulates the old itemquery.php service:

```
<service name="search_parcel">
  <url>php/query.php</url>
  <step type="input">
    <input type="select" name="fieldname0" title="Search By:">
      <option value="OWNER_NAME">Owner</option>
      <option value="PIN">Parcel ID</option>
    </input>
    <input type="hidden" name="comparator0" value="like-icase"/>
    <input type="user" name="value0" title=""/>
    <input type="hidden" name="layer0" value="parcels/parcels"/>
    <input type="hidden" name="template0" value="itemquery"/>

    <input type="hidden" name="highlight" value="true"/>
    <input type="hidden" name="mode" value="search"/>
  </step>
</service>
```

This is an example that allows the user to choose what comparator is used to search:

```
<service name="search_parcel">
  <url>php/query.php</url>
  <step type="input">
    <input type="select" name="fieldname0" title="Search By:">
      <option value="OWNER_NAME">Owner</option>
      <option value="PIN">Parcel ID</option>
    </input>
    <input type="select" name="comparator0" title="That: ">
      <option value="like-icase">Contains</option>
      <option value="right-like-icase">Begins With</option>
      <option value="eq-str">Matches Exactly</option>
    </input>
    <input type="user" name="value0" title=""/>
    <input type="hidden" name="layer0" value="parcels/parcels"/>
    <input type="hidden" name="template0" value="itemquery"/>

    <input type="hidden" name="highlight" value="true"/>
    <input type="hidden" name="mode" value="search"/>
  </step>
</service>
```

This example shows searching multiple fields with selectable operators and comparitors:

```
<service name="search_parcel">
  <url>php/query.php</url>
  <step type="input">
```

```

<input type="hidden" name="highlight" value="true"/>
<input type="hidden" name="mode" value="search"/>

<input type="hidden" name="layer0" value="parcels/parcels"/>
<input type="hidden" name="template0" value="itemquery"/>

<input type="select" name="fieldname0" title="Search By:">
  <option value="OWNER_NAME">Owner</option>
  <option value="PIN">Parcel ID</option>
</input>
<input type="select" name="comparator0" title="That: ">
  <option value="like-icase">Contains</option>
  <option value="right-like-icase">Begins With</option>
  <option value="eq-str">Matches Exactly</option>
</input>
<input type="user" name="value0" title=""/>

<input type="hidden" name="fieldname1" value="FIN_SQ_FT"/>
<input type="select" name="operator1">
  <option value="or">OR</option>
  <option value="and">AND</option>
</input>
<input type="select" name="comparator1" title="Having Fin. Sq. Ft. ">
  <option value="gt">Greater Than</option>
  <option value="eq">Equal To</option>
  <option value="lt">Less Than</option>
</input>
<input type="user" name="value1" title=""/>
</step>
</service>

```

This example is the current service in the demo:

```

<service name="search_parcels" title="Search">
  <url>php/query.php</url>
  <step type="input">
    <input type="hidden" name="highlight" value="true"/>
    <input type="hidden" name="mode" value="search"/>

    <input type="hidden" name="layer0" value="parcels/parcels"/>
    <input type="hidden" name="template0" value="itemquery"/>

    <input type="select" name="fieldname0" title="Search By:">
      <option value="OWNER_NAME">Owner</option>
      <option value="PIN">Parcel ID</option>
    </input>
    <input type="select" name="comparator0" title="That: ">
      <option value="like-icase">Contains</option>
      <option value="right-like-icase">Begins With</option>
      <option value="eq-str">Matches Exactly</option>
    </input>
    <input type="user" name="value0" title=""/>

    <input type="hidden" name="fieldname1" value="FIN_SQ_FT"/>
    <input type="select" name="operator1">
      <option value="or">OR</option>
      <option value="and">AND</option>
    </input>
    <input type="select" name="comparator1" title="Having Fin. Sq. Ft. ">

```

```
        <option value="gt">Greater Than</option>
        <option value="eq">Equal To</option>
        <option value="lt">Less Than</option>
    </input>
    <input type="user" name="value1" title="" />
</step>
</service>
```

Additional resources

See also, *<service>*.

4.4 Extensions Documentation

4.4.1 Adding Hover-Over Popups to A Layer

GeoMOOSE supports hover-over popups on both Vector (WFS) and Raster Layers (WMS/MapServer).

Adding Popups To WFS Layers

Adding popups to Vector layers requires adding the HTML template to the `map-source` configuration and the `popups=true` attribute to its catalog entry.

The HTML Template

The HTML templates are Dojo-type string templates. They are slightly different than MapServer templates and use `${attribute}` formatting:

```
<map-source ...>
    ...
    <popup-template><![CDATA[
    <div style="font-size: 1.5em">${name|sad}</div>
    Area of Land: ${aland}<br>
    Area of Water: ${awater}<br>
    <br>
    <a href="https://www.census.gov/2010census/popmap/ipmtext.php?fl=${statefp}:${geoid}" target=
    ]]></popup-template>
</map-source>
```

And in the catalog, `popups="true"` is added to the `<layer>` entry:

```
<layer title="Census Cities - WFS" src="census_cities" activate="true" fade="false" unfade="false" me
```

Adding Popups to WMS Layers

The demo includes functionality for WMS popups for both points and polygon representations of the Parcels layer.

The steps required:

1. Create a HTML template.
2. Enable WMS Queries in the Mapfile.

3. Enable Popups for the Layer in the Mapbook.

Create a HTML Template

The HTML templates are standard MapServer templates. The reference for them can be found on the MapServer website. This is the example included in the GeoMOOSE Demo:

```
<!-- MapServer Template -->
<b>PIN:</b> [PIN]<br/>
<b>Owner:</b> [OWNER_NAME]<br/>
<br/>
```

```
<a href="javascript:GeoMOOSE.startService('feature_report', {'src' : 'parcels/parcels', 'PIN' : '[PIN]'"
```

Attributes are substituted using [and]. GeoMOOSE can also startup other services from links provided in popups as seen above.

Enable WMS Queries in the Mapfile

MapServer requires a few key elements to be included in order to WMS queries to work.

1. WEB's METADATA must include wms_feature_info_mime_type set to text/html.

```
WEB
    ...
    METADATA
        ...
        'wms_feature_info_mime_type' 'text/html'
    END
END
```

2. The LAYER must have a CLASS with a NAME.
3. The LAYER must also have a template ending in .html:

```
LAYER
    NAME 'parcel_points'
    CLASS
        NAME 'Parcels!'
        ...
    END
    # this references the html file above
    TEMPLATE 'parcels_popup.html'
END
```

Enable Popups in the Catalog

popups="true" is added to the <layer...> entry in the catalog:

```
<layer title="Parcel Points" src="parcel_points/parcel_points" popups="true"/>
```

4.4.2 Bookmark To Tab

Description

This extension extends the built-in bookmarking functionality by placing the generated bookmark in a tab instead of updating the URL in the address bar. It does this by overriding `GeoMOOSE.bookmark()` with a custom function.

Adding to Application

1. The default GeoMOOSE configuration does not provide an interface to trigger `GeoMOOSE.bookmark()`. To take full advantage of this functionality, you will probably want to add the following to the `<toolbar>` section in your `mapbook.xml`:

```
<tool name="bookmark" title="Bookmark this page" type="javascript">GeoMOOSE.bookmark();</tool>
```

2. To enable this extension, add the following line to the `<head>` section of `geomoose.html`:

```
<script type="text/javascript" src="extensions/BookmarkToTab.js"></script>
```

The bookmark tool should then display in a new tab. If you would like to change the layout of the tab you can do so by overriding `window.BOOKMARK_TEMPLATE`:

```
<script type="text/javascript">
  window.BOOKMARK_TEMPLATE = "<a href='%url%'>Bookmark Me</a>";
</script>

<script type="text/javascript" src="extensions/Disclaimer.js"></script>
```

4.4.3 Disclaimer

Description

Some users wish to put a disclaimer at the beginning of their application. `Disclaimer.js` provides this by prompting the user with an alert box when the application starts.

Adding to Application

To add the Disclaimer to your application add the following lines:

```
<script type="text/javascript">
  window.DISCLAIMER_MESSAGE = "This is my disclaimer.";
</script>

<script type="text/javascript" src="extensions/Disclaimer.js"></script>
```

Setting `DISCLAIMER_MESSAGE` sets what disclaimer is show at startup. Without it the user will see “Hello, world.”

4.4.4 Dynamic Select

Description

The default “Zoom To”s with GeoMOOSE are static and predefined in the Mapbook. They do not change based on updates of a layer, and can be tedious to type out even for layers with relatively few features. To address this,

DynamicZoom provides an Ajax-capable feature look up that allows the user to select a dataset, then a feature from that dataset. After selecting the feature, the map zooms to the extent of that feature.

Adding to Application

To add Dynamic Select to your application add:

```
<script type="text/javascript" src="extensions/DynamicZoom.js"></script>
```

Configuration

Configuring Dynamic Select consists of three steps:

1. Create a template file.
2. Edit the layer's metadata.
3. Add the layer to DynamicZoomConfiguration

1. Create the template file

To create the template file, goto the direct where the mapfile exists. For this example, we'll be using the "counties_border" layer inside of the "basemap.map" mapfile. This mapfile can be found in the *maps/demo/statedata* directory that comes with the GeoMOOSE demo. The template is very simple, and creates the needed to create "<option>" tags in HTML:

```
<!-- MapServer Template -->
<option value="[shpext]">[COUNTYNAME]</option>
```

The first line is the MapServer "magic string" that is required for template files starting in version 5.4. The second line, has a few important prts. The first is the "[shpext]" directive, this tells mapserver to place the feature's extent into this part of the string. "[COUNTYNAME]" tells MapServer to place the feature's title, in this case 'COUNTYNAME,' into the template. When working with a different layer, substitute "COUNTYNAME" for whatever the layer's title column name happens to be. For this example, we'll name the file, "county_dynamic_zoom.html"

2. Edit the layer's metadata.

Now we need to tell GeoMOOSE about this template file. The method for doing that is to add a METADATA entry to the mapfile. First, it is necessary to find the LAYER entry in the mapfile, that will look something like this:

```
LAYER
    NAME county_borders
    ...
END
```

Then, check to see if there is already a METADATA section, if there is not then add one:

```
LAYER
    NAME county_borders
    ...
    METADATA
    END
END
```

Finally, the metadata entry needs to be made:

```
LAYER
    NAME county_borders
    ...
    METADATA
        'dynamic_zoom' 'county_dynamic_zoom.html'
    END
END
```

3. Add the layer to DynamicZoomConfiguration

Edit *htdocs/extensions/DynamicZoom.js*, at the very top is a Javascript Object named “DynamicZoomConfiguration.” In DynamicZoomConfiguration there is an entry called layers, it is an object that contains the information for the various layers. If the file is from a default-GeoMOOSE installation then the entry for “Counties” is already there. If it is missing then adding the following lines would create it:

```
...
layers: {
    'Counties' : {
        mapbook_src: 'borders/county_borders', /* this is the src of the layer in the mapbook */
        qitem: '', /* leave blank */
        string: '' /* leave blank */
    }
}
...
```

Other Options

In the above instructions “qitem” and “qstring” were intentionally left blank. However, if they are set, they will act as a filter on the dataset. This is useful if you have multiple feature types stored in a single table. By default, the filtering is case-insensitive.

4.5 Helpful How To's

4.5.1 Getting Started Customizing Your App

Changing the Projection

Introduction

GeoMOOSE 2 uses the [Proj4js library](#) to perform all of its projection work. This is much more robust than the older GeoMOOSE reprojection library but requires a different setup to use different projections within the application.

Projections are defined using the standard Proj4 syntax which configures the datum, origin, and units for the projection among other parameters. Most applications will use a well-known projection that is assigned a unique identifier, or “EPSG code”. The [Spatial Reference website](#) contains a listing of valid EPSG codes and definitions in various formats. The Proj4 C library also includes the full list of EPSG definitions. This list can be found in the file `proj/nad/epsg` included with MS4W.

Proj4js does **not** include the full list of definitions by default, though it can dynamically load definitions directly from the Spatial Reference website. However, it is recommended that you explicitly create an EPSG code definition file for your projection in order to use it in your map.

Step 1: Determine your projection.

First check to see if your projection already exists as an EPSG definition. Having a predetermined projection makes this step quick and easy. Note that the same basic projection may have multiple variants with differing EPSG codes. For example, UTM 15N is defined as [EPSG:26915](#) when based on the NAD83 datum, but as [EPSG:32615](#) when based on the WGS 84 datum. Most commercial map layers use a projection commonly known as Web or [Spherical Mercator](#), which now has the standard code [EPSG:3857](#). The GeoMoose demo is configured to use EPSG:3857 by default.

If your projection is not found in the standard list, you will need to define your own. For example, an arbitrary projection might be defined as:

```
+proj=tmerc +lat_0=38 +lon_0=125 +k=1 +x_0=200000 +y_0=500000 +ellps=bessel +units=m +no_defs
```

The projection definition must be placed within a JavaScript file as described below.

Step 2: Create a Projection Javascript File.

Projection files should be placed in the `htdocs/projections` directory and follow the naming convention `EPSGXXXXX.js` where “XXXXX” is the projection code. If you created a custom projection in step one be sure to make this a number above 100000. In side of this file you should have:

```
Proj4js.defs["EPSG:XXXXX"] = "+proj=tmerc +lat_0=38 +lon_0=125 +k=1 +x_0=200000 +y_0=500000 +ellps=b
```

Again, substitute your own EPSG number and projection information above. The [Spatial Reference](#) website provides Proj4js format files for well known projections. After searching for and finding a projection you will have the option to export the projection in Proj4js format. Simply copy and paste the code into your `EPSGXXXXX.js` file.

Include the Javascript File in your HTML Find the section in the `<head>` of `geomoose.html` that states, “Include your projection here” and place the code:

```
<script type="text/javascript" src="projections/EPSGXXXXX.js"></script>
```

Step 3: Ensure your data sources support your projection

This step will depend on your data sources and target projection. If your data source does not support your projection you will usually need to reproject it. Coordinates and vector layers (e.g. WFS) can generally be reprojected from within the web browser as long as both Proj4js definitions are provided. Raster layers (e.g. WMS) can often be reprojected using MapServer. In the case of the demo it is as simple as adding the EPSG code to the `ows_srs` parameter in `basemap.map`, `parcels.map`, `lmic.map`, and `usgs.map`:

```
WEB
    METADATA
        'ows_title' 'County'
        'ows_srs' [...] EPSG:3857 EPSG:XXXXX'
        'ows_enable_request' '*'
    END
END
PROJECTION
    'init=epsg:26915'
END
```

The initial projection should remain as whatever the source data is in (EPSG:26915 in this case).

Note: Commercial map layers (like Bing, Google, and MapQuest) generally do not support reprojection, so you will not be able to easily use them with any map projection other than EPSG:3857 (or the unofficial equivalent EPSG:900913).

Step 4: Change the mapbook parameter for your projection.

In the `<configuration>` section of the mapbook you will need to change the following line:

```
<param name="projection">EPSG:XXXXX</param>
```

If you are changing the projection for an existing map (such as the demo), you will almost certainly need to change the `max_extent` and `initial_extent` parameters or the entire map may be shifted far off-screen. Essentially you will need to reproject the extent coordinates from the old projection into the new. One way to do approximate this from within GeoMOOSE is by panning and zooming until your data is back in view, then recording the X,Y coordinates at the bottom left and top right of the region. This task will be much simpler if you are able to use a worldwide map layer that supports your target projection. The ArcGIS REST example in the demo supports a wide range of projections and can be useful for this purpose.

For example, after changing the demo to Minnesota state plane South (EPSG:26993) you would want to change the `initial_extent` to something like:

```
<param name="initial_extent">2169678,512220,2268587,618926</param>
```

For more information on finding projections and projection definition strings visit <http://www.spatialreference.org>.

How To Add Layers

Adding a new layer in GeoMOOSE 2 is a two step procedure. The first step is to define how GeoMOOSE should communicate with the layer (whether it is MapServer or a WMS source) and the second step defines the layer in the catalog.

Step 1: Adding a Layer

All requests start with a `<map-source>` tag with a `name` and `type` attribute. The name is the name of the source that will be used with GeoMOOSE. This name does not need to relate to the source that is going to be read. The type attribute determines what type of service will be read. The children of the `<map-source>` are determined by the type attribute. Currently `mapserver` and `wms` types are the two that are supported.

All `<map-source>` types have `<layer>` children. `<layer>` children have only one attribute `name`. The name attribute for `<layer>` children reflects the name of the layer in the MapServer Mapfile or the name of the layer served by the WMS Service. Some MapServer mapfiles will have all layers set to “default” in which case there should still be one `<layer>` entry in the `<map-source>` with the name of “all.”

A Mapserver Layer Mapserver Layers need to have the `<file>` tag filled out specifying the location of the mapfile. This path can be relative to the `root=` value in the `[paths]` section of `local_settings.ini` or an absolute path on the file system.

Here is an example of the parcels layer from the default demo:

```
<map-source name="parcels" type="mapserver">
  <file>./demo/landrecords/parcels.map</file>
  <layer name="all"/>
</map-source>
```

Here is an example of the borders including multiple layers from the default demo:

```
<map-source name="borders" type="mapserver" reference="true">
  <file>./demo/statedata/basemap.map</file>
  <layer name="city_labels" status="on"/>
  <layer name="county_labels" status="on"/>
  <layer name="city_poly" status="on"/>
  <layer name="county_borders" status="on"/>
</map-source>
```

A WMS Layer WMS Layers have a `<url>` child that defines the root URL of the WMS service.

This is an example using the LMIC FSA Photography:

```
<map-source name="lmic" type="wms" tiled="false">
  <url>http://geoint.lmic.state.mn.us/cgi-bin/wms</url>
  <layer name="fsa"/>
</map-source>
```

Step 2: Adding the Layer to the Catalog

After defining how to talk to the layer we need to define how to display the layer in the interface.

Layers are defined in the `<catalog>` section of the mapbook. To add the LMIC FSA photography the `<layer>` entry would look like this:

```
<layer title="LMIC 2003 FSA Aerials" src="lmic/fsa" status="on"/>
```

To add the borders including multiple layers the `<layer>` entry would look like this:

```
<layer title="City and County Boundaries" src="borders/city_labels:borders/county_labels:borders/cou"
```

The individual parts:

- `title=` - Sets the title in the mapbook to be displayed
- `src=` - This is the layer name, it is a combination of the `map-source`'s name combined with the layer's name from the definitions above.
- `status=` - This determines whether the layer is either on or off by when the map loads. Omitting `status=` will leave the layer off.
- `legend=` - (true of false) This determines whether to show the legend control (a small '+' icon to expand and collapse the legend) or not. See also, [mapbook catalog section](#) and [layer_controls.legend.on](#) configuration paramter.
- `show-legend=` - (true of false) This determines whether to show the legend or not by default (i.e. `show-legend="false"` and `legend="true"` lets the user expand and see the legend). See also, [mapbook catalog section](#) and [layer_controls.legend.on](#) configuration paramter.
- `fade=` - (true of false) This determines whether to allow fading for the layer or not.
- `unfade=` - (true of false) This determines whether to allow unfading for the layer or not.
- `metadata=` - (true of false) When set to 'true', the metadata tag is also included. See also, [<catalog>](#) in the Mapbook Reference.

Tips and Advice

Using MapServer GeoMoose fully utilizes MapServer mapfiles. The MapServer files and data are located in the maps directory in the demo. Extensive documentation regarding Mapserver is located at [MapServer](#). A few important issues should be remembered when using MapServer from within the GeoMoose environment:

1. GeoMoose 2.6+ now uses MapServer only as a published WMS (for display). GeoMoose 2.6 is now configured to only consume MapServer spatial maps as a WMS. This means the map file needs to contain the code to publish a WMS. At a minimum the map file should have the metadata code to name the service, identify the projection, and enable a request as illustrated in the following code taken from the parcel.map file in the demo.

```
WEB
    METADATA
        'ows_title' 'County'
        'ows_srs' 'EPSG:26915 EPSG:4326 EPSG:900913 EPSG:3857'
        'ows_enable_request' '*'
    END
END
```

MapServer contains many other options for configuring the WMS which are discussed in detail on the MapServer web site. As an added benefit the WMS now produced can also be consumed by other applications that can consume a WMS.

2. Using geomoose_globals.map file GeoMoose map files typically reference the “geomoose_globals.map” file as included in the parcel.map file on the demo.

```
INCLUDE "../././geomoose_globals.map"
```

The “geomoose_globals.map” file is located in the maps directory and contains basic map parameters that GeoMoose relies on such as the calls to the symbol and fontset files as well as predefined output parameters. The demo has been setup based on relative paths to call the global file which in turn uses the relative path to call the fontset and symbol files. **It is important that if you change the general path structure that is used in the demo that you change the relative path calls in the geomoose_globals file.**

3. Symbols and Fonts Symbols and Fonts that are referenced by MapServer are stored in the maps/fonts and maps/symbols directory. Remember to check that any fonts or symbols referenced in a mapfile exist. MapServer has great documentation on changing symbols and fonts.

4. Expanding map layer functionality The functionality of your map layers can be increased to support queries, identify, selects and reports from within the mapfile as illustrated in the parcel.map file in the demo.

```
LAYER
NAME parcels
DATA './parcels.shp'
...
...
    METADATA
        # drill-down identify service record.
        'identify_record' 'templates/identify.html'

        # query.php / "Search Parcels" functionality.
        'itemquery' 'templates/search_result.html'
        'itemquery-filter' '/*[qstring].*/i'
        'qstring_validation_pattern' '.'
```

```

        # Feature reports are stored in the conf/feature_report directory.
        'feature_report' 'parcel.xml'

        'select_record' 'templates/select_result.html'
        'select_header' 'templates/select_header.html'

    END

...
END

```

This functionality is discussed in more detail in other sections of the documentation.

How To Configure Identify for a Layer

GeoMoose contains an identify service that allows you to identify and display attributes for a map layer. Adding the identify ability to identify features for a layer in GeoMoose 2 is a 3 step procedure. The first step is to create an HTML file that controls which fields and format those fields for display of attributes. The second step is to reference the HTML file in the map file. The third optional step is to change the header and footer.

Step 1: Creating the Identify HTML file

The IDENTIFY HTML file will be used to format how information is displayed in the identify results tab. This file follows standard HTML formats and will reference fields that exist in the layer attribute table. Below is an example of the identify html for parcels in the demo:

```

<!-- MapServer Template -->
<tr bgcolor="#DEE5EB"><td colspan="2"><b><u>Parcels</u></b></td></tr>
<tr>
<td align="right"><b>PIN:</b></td>
<td>[PIN] <a href="javascript:GeoMOOSE.startService('feature_report', {'PIN' : '[PIN]', 'src' : 'par
</tr>
<tr>
<td align="right"><b>Owner Name:</b></td>
<td>[OWNER_NAME]</td>
</tr>
<tr>
<td align="right"><b>Est. Market Value:</b></td>
<td>[EMV_TOTAL]</td>
</tr>
<tr>
<td align="right"><b>Acres:</b></td>
<td>[ACRES_POLY]</td>
</tr>
<tr>
<td>&nbsp;</td>
<td>&nbsp;</td>
</tr>

```

Please note that this form can be used to link to other sites or services using standard HTML like tags as illustrated in the example above.

Step 2: Reference the Identify HTML in MapServer map file

The Identify HTML must then be referenced in your map file within your layer. The following is an example of how this is done from the parcels.map file in the demo.

```
LAYER
NAME parcels
DATA './parcels.shp'
...
...
    METADATA
        # drill-down identify service record.
        'identify_record'      'templates/identify.html'
        ...
        ...
    END
...
END
```

Remember when referencing the identify.html to make sure the correct relative paths are set.

Step 3: Optionally changing the identify header and footer

You may want to change the header and footer for the identify service. This can easily be done by modifying the footer.html and header.html file in the htdocsphpidentify directory or changing the path to a different location as the identify header and footer in the [identify] section of the settings.ini file in the conf directory.

Tips and Advice

Forcing Identify to always identify a specific layer You can force identify to always identify a specific layer by editing the mapbook.xml file in the conf directory and adding the following parameters to the identify service section:

```
<service name="identify" title="Identify" display="true">
    <url>php/identify.php</url>
    <step type="spatial" name="shape" line="false" polygon="false" jump-start="true" default="po
        ...
        ...
        <input type="hidden" name="layers" value="parcels/parcels"/>
        ...
        ...
    </step>
</service>
```

You can add multiple layers to this as well by adding additional source/layer names to the value parameter separated by a ":"

```
<input type="hidden" name="layers" value="parcels/parcels:mymodel1/layername1:mymodel2/layername2",
```

Why no results with point or line layers? You need to set a TOLERANCE in the mapfile, see also *FAQ - Why no results?*

How To Configure Select Capability For A Layer

GeoMoose includes a select service that allows you to graphically select features and display attributes for a map layer. Configuring the select capability for a layer in GeoMoose 2 is a multistep proces.

Step 1: Create the HTML files

The HTML files will be used to format how information is displayed in the select results tab. The files follow standard HTML formats and will reference fields that exist for the select layer. Below is an example of the header HTML for parcels in the demo select_header.html file located in ..maps/demo/parcels/templates:

```
<!-- MapServer Template -->
<a target="_blank" href="php/mailling_labels.php?queryid=[QUERYID]&output=pdf">PDF Mailing Labels</a>
<a target="_blank" href="php/mailling_labels.php?queryid=[QUERYID]&output=html">HTML Mailing Labels</a>
<a target="_blank" href="php/mailling_labels.php?queryid=[QUERYID]&output=csv">CSV Mailing Labels</a>
<br/>
<div style="display: [SHOW_FOLLOWUP]" />
<a href="javascript:GeoMOOSE.startService('buffered_select_followup', {shape: '[SHAPE_WKT]', select_
<br/><br/>
</div>
```

Please note that this HTML can be used to link to other sites or services using standard HTML like tags as illustrated in the example above. The header is displayed one time at the top of the results tab. Following the header will be the results which will display for each selected feature. Below is an example of the results HTML for parcels in the demo select_results.html file located in ..maps/demo/parcels/templates:

```
<!-- MapServer Template -->
<table>
<tr>
<td><b>PIN:</b></td>
<td>[PIN]</td>
</tr>
<tr>
<td><b>Owner Name:</b></td>
<td>[OWNER_NAME]</td>
</tr>
<tr>
<td><b>Est. Market Value:</b></td>
<td>[EMV_TOTAL]</td>
</tr>
<tr>
<td><b>Acres:</b></td>
<td>[ACRES_POLY]</td>
</tr>
<tr>
<td colspan="2"><hr/></td>
</tr>
</table>
```

You can also have a footer file. Remember the first line of your file must be “<!-- MapServer Template -->” followed by the appropriate HTML tags and it must be referenced in your map file correctly as discussed in the following section.

Step 2: Reference the HTML files in your MapServer map file

The select HTML files must then be referenced in the metadata section in your map file within your layer. The following is an example of how this is done from the parcels.map file in the demo located in ..maps/demo/parcels.

```
LAYER
NAME parcels
DATA './parcels.shp'
...
...
    METADATA
        ...
        'select_record'      'templates/select_result.html'
        'select_header'     'templates/select_header.html'
        ...
    END
...
END
```

Remember when referencing the HTML's to make sure you have the correct relative paths set.

Step 3: Add the layer to the Select Service in mapbook.xml

The third step is to add the layer to the select.php reference in the mapbook.xml. You can have multiple layers available for selection in the select service. To add a layer locate the select service called `<service name="buffered_select" title="Select Features">` in the mapbook.xml. Locate the line for parcels in the section and add your line so it appears as follows:

```
<input type="select" name="select_layer" title="Select features from:">
    <option value="parcels/parcels">Parcels</option>
    <option value="source/name">MyNewSearch Layer</option>
</input>
```

`<option value="source/name">MyNewSearch Layer</option>` is the added line

Your new select layer must be referenced by its source and layer name.

Tips and Advice

Why no results with point or line layers? You need to set a TOLERANCE in the mapfile, see also *FAQ - Why no results?*

How To Configure Search Capability For A Layer

GeoMoose includes a search service to search for features by attribute value and display the results for a specific layer. Configuring the search capability for a layer in GeoMoose 2 is a multistep proces.

Step 1: Create the search results HTML file

The searh results HTML file will be used to format how information is displayed in the search results tab. The file follows standard HTML formats and references fields that exist for the serch layer. Below is an example of the HTML for parcels in the demo search_result.html file located in `..maps/demo/parcels/templates`.

```
<!-- MapServer Template -->
<a id="gm-parcel-[PIN]" class='sprite-control sprite-control-find-selected' style="padding-left: 22px"
[OWNER_NAME]><br/>
<br/>
```

Please note that this HTML can be used to link to other sites or services using standard HTML like tags as illustrated in the example above.

Step 2: Reference the HTML file in MapServer map file

The search HTML files must then be referenced in the metadata section in the map file within the layer. The following is an example of how this is done from the parcels.map file in the demo located in ../maps/demo/parcels.

```
LAYER
NAME parcels
DATA './parcels.shp'
...
...
    METADATA
        ...
        itemquery'          'templates/search_result.html'
        'itemquery-filter'    '/*[qstring].*/i'
        'qstring_validation_pattern' '.'
        ...
    END
...
END
```

The itemquery-filter and qstring_validation_pattern must also be set for search to work. Remember when referencing the HTML's to make sure the correct relative paths are set.

Step 3: Create a new search service in mapbook.xml

The third step will be to create a new search service for a layer in the mapbook.xml. This is most easily done by copying the search service and pasting it in and then modifying it. The following is what the search service looks like for the parcel layer.

```
<service name="search_parcels" title="Search">                                <-- create a new serv
    <url>php/query.php</url>
    <step type="input">
        <input type="hidden" name="highlight" value="true"/>
        <input type="hidden" name="mode" value="search"/>

        <input type="hidden" name="layer0" value="parcels/parcels"/>          <--- source and layer
        <input type="hidden" name="template0" value="itemquery"/>

        <input type="select" name="fieldname0" title="Search By:">
            <option value="OWNER_NAME">Owner</option>                        <--- fields you want
            <option value="PIN">Parcel ID</option>                            from yo
        </input>
        <input type="select" name="comparator0" title="That: ">              <--- comparitors for
            <option value="like-icase">Contains</option>
            <option value="right-like-icase">Begins With</option>
            <option value="eq-str">Matches Exactly</option>
        </input>
        <input type="user" name="value0" title=""/>                          <--- value you will

        <input type="hidden" name="fieldname1" value="FIN_SQ_FT"/>            <--- additional valu
        <input type="select" name="operator1">
            <option value="or">OR</option>
            <option value="and">AND</option>
```

```
        </input>
        <input type="select" name="comparator1" title="Having Fin. Sq. Ft. ">
            <option value="gt">Greater Than</option>
            <option value="eq">Equal To</option>
            <option value="lt">Less Than</option>
        </input>
        <input type="user" name="value1" title=""/>
    </step>
</service>
```

Below are highlights of the marked code above:

- `<service name="search_parcels" title="Search">` ← create a new service name
- `<input type="hidden" name="layer0" value="parcels/parcels"/>` ← source and layer name for your layer
- `<option value="OWNER_NAME">Owner</option>` ← fields you want to search on
- `<option value="PIN">Parcel ID</option>` ← from your layer
- `<input type="select" name="comparator0" title="That: ">` ← comparitors for searching
- `<input type="user" name="value0" title="">` ← value you will enter
- `<input type="hidden" name="fieldname1" value="FIN_SQ_FT"/>` ← additional value search

Additional information about the search service and adding services in the mapbook can be found in the mapbook reference. See also, [Mapbook services reference](#) and [Query documentation in Services](#).

Step 4: Add the service to the toolbar

Now that the service has been added the user must be able to select it as a tool. This is done by adding the new service to the toolbar section of the mapbook.xml file. The following is an example of what the search parcel service:

```
<tool name="search_parcels" title="Search Parcels" type="service" service="search_parcels" selectable="true">
```

To add the new service copy this line and paste it below the existing search tool. Change the service name to match the new service. The following is a simple example that illustrates how this could be done for a new service to search for taxlots:

```
<tool name="search_taxlots" title="Search taxlots" type="service" service="search_taxlots" selectable="true">
```

How To change Jump-To (Zoom To)

Here is an example of the zoom to from the default demo:

```
<param name="zoomto"><![CDATA[
{
    "Jump To:" : {
        'World' : [-20614760.777156,1751325.1919492,-1927436.1053437,7915207.1517617],
        'Parcel Data' : [-10384069.859924,5538318.529767,-10356632.423788,5563580.927174],
        'Full State of MN' : [-10742765,5398288,-9920914,6310641]
    }
}
]]></param>
```

Notice that they are separated with commas and there is no trailing comma after the last entry. An additional zoomto parameter could be added before the 'Full State of MN' line as follows:

```
'Farmington' : [-10376231, 5563517, -10367384, 5572942],
```

Remember to include appropriate punctuation. If this is wrong the application may not start correctly as configuration parameters are very important.

How To Create a Custom Printing Template

GeoMOOSE 2 uses a page-templating method for generating PDFs. Any printed map in GeoMOOSE starts as a pre-existing PDF that does not feature a map. This PDF can have inline fonts, J PEGs, watermarks, etc. In GeoMOOSE 1.X it was necessary to write PHP to change the output format of the printing utility with 2.X that is no longer needed. Creating PDF files have become commonplace and can be done with almost any software with the availability of PDF printers.

Step 1: Make Your Template PDF

The first step is to create a PDF that you wish to use as a printing template. This can be done with any software that can output to a PDF. If none of your software has an "Export PDF" option then a PDF printer may be an option on your operating system:

- Mac OS/X - A PDF Printer is built-in. See Apple's Documentation.
- Windows - I've had good luck with CutePDF. <http://www.cutepdf.com/>
- Linux - Use CUPS. <http://www.linux.com/articles/61826>

For purposes of this document the PDF file generated here will be referred to as "my_template.pdf". "my_template.pdf" should be saved/copied into `conf/print`.

Step 2: Define the Template for GeoMOOSE

GeoMOOSE is smart, but not that smart, it does need to be told more information about the template created in Step 1. To do this we need to create an XML template definition file. It is easiest to copy a pre-existing template and modify it. For convenience, it makes sense to name the XML file the same as the PDF file. Copy `letter_landscape.xml` to `my_template.pdf`. You should now have a file that contains:

```
<print-template>
  <!-- This is the template -->
  <template>default_template.pdf</template>

  <!-- page dimensions -->
  <page w='11' h='8.5' />

  <!-- this is the location of the map in the template -->
  <map x="0.43" y="1.43" w="10.15" h="6.42"/>

</print-template>
```

The important element to change is the name of the PDF template file to "my_template.pdf":

```
<print-template>
  <!-- This is the template -->
  <template>my_template.pdf</template>
```

```
<!-- page dimensions -->
<page w='11' h='8.5' />

<!-- this is the location of the map in the template -->
<map x="0.43" y="1.43" w="10.15" h="6.42"/>

</print-template>
```

It is best to have a good guess for the `<page>` and `<map>` attributes but they can be adjusted later to fine tune the template.

Step 3: Add the Template to the Print Options

In the mapbook, the print utility has an `<input>` that allows the user to select a template. To make the template selectable it is necessary to add it to the mapbook definition.

The print service will initially have the following options:

```
<service name="print" title="Print Map">
  <url>php/print.php</url>
  <input type="print_info" name="layers"/>
  <input type="extent" name="extent"/>

  <input type="select" name="template" title="Output Template: ">
    <option value="letter_landscape">Letter - Landscape</option>
    <option value="letter_portrait">Letter - Portrait</option>
    <option value="poster_landscape">Poster - Landscape</option>
    <option value="poster_portrait">Poster - Portrait</option>
  </input>
  (snip)
</service>
```

In order to make “my_template” available it just needs added to the list:

```
<service name="print" title="Print Map">
  <url>php/print.php</url>
  <input type="print_info" name="layers"/>
  <input type="extent" name="extent"/>

  <input type="select" name="template" title="Output Template: ">
    <option value="my_template">MY TEMPLATE!!!!</option>
    <option value="letter_landscape">Letter - Landscape</option>
    <option value="letter_portrait">Letter - Portrait</option>
    <option value="poster_landscape">Poster - Landscape</option>
    <option value="poster_portrait">Poster - Portrait</option>
  </input>
  (snip)
</service>
```

Reloading GeoMOOSE should show the new template as available and users should now be able to print using that template.

Other Notes

It is possible to add dynamic text to a map. This is done by specifying a `<text>` tag. The text tells the printing service to place text on the page and it supports some basic string substitution. Strings featuring a variable like ‘%abc%’ will be substituted with URL values. For example if the string is:

Hello %name%!

And “name=Dave” is passed into the print script via the CGI parameters, the string will become:

Hello Dave!

An example can be found in the demo’s printing templates:

```
<text x=".5" y=".75" size="48" content="%title%"/>
```

The attributes breakdown like this:

- x - The x-position in inches on the page.
- y - The y-position in inches on the page.
- size - The font-size in points.
- content - The string to place on the page.

How To Add Text to a Tool

Some tools should have text. While an Icon is nice, sometimes you may only have a single icon that is used repetetively for a number of searches. As is often the case, it’s simply easier to read, “Search Assets” instead of filing through the four or five relatively obscure looking 16 x 16 pixel icons. Tools on the toolbar are defined in the `<toolbar>` section of the mapbook.

Simply add `show-label="true"` to the tool in the mapbook.xml to show text. To not display text remove the `show-label="true"` as the default is to not display text.

To Show Text For ZoomIn:

```
<tool name="zoomin" title="Zoom In" type="internal" action="zoomin" show-label="true"/>
```

To Not Show Text For ZoomIn:

```
<tool name="zoomin" title="Zoom In" type="internal" action="zoomin"/>
```

How To Put Content in the Menubar

The menubar’s content can be specified by using simply HTML in the Mapbook.

In the `<configuration>` section you can add:

```
<param name="links_bar_html"><![CDATA[  
    <b>THIS IS MY HTML!!!</b>  
]]></param>
```

Clear the browser cache and reload the interface! The HTML should appear where the menubar once existed!

How To Create Your Own Skin

GeoMOOSE 2 has greatly simplified the skinning and CSS required to change elements of the interface.

Step 1: Make a Copy of a Previous Existing Skin

In the current distribution the easiest to modify skin is the “blue” skin. The directory has sub-directories so it will be necessary to do a recursive copy (the default on windows):

```
cd htdocs/skins
cp -r blue myskin
```

Step 2: Rename the Skin and Edit It!

This is just a small step to keep things clean:

```
cp blue.css myskin.css
```

Open myskin.css in a text editor and change the code! After following step 3 you’ll see the changes in the interface.

Step 3: Set GeoMOOSE to use your Skin

Open “geomoose.html” in a text editor. In the <head> section there should be a tag starting <link... which mentions the skin. Comment out all of the skins you will not be using and add a <link> tag including your skin.

This is the default geomoose.html with the *myskin* skin selected:

```
<!--
    This is where your skin is defined.
    For an example, comment out the line containing "green.css" and
    uncomment "blue.css".
-->
<!--<link type="text/css" rel="stylesheet" href="skins/green/green.css"/> -->
<!--<link type="text/css" rel="stylesheet" href="skins/blue/blue.css"/> -->
<link type="text/css" rel="stylesheet" href="skins/myskin/myskin.css"/>
```

Tips and Advice

Changing just the Header Image While some folks may want to change the entire look of the site you may only wish to change the header image to match your website. Follow all three steps above. This creates a unique skin for your site. Then change to the skin’s image directory:

```
cd htdocs/skins/myskin/images
```

In that directory is the “logo_top.png” image. This is the image that is used in the header, it is 670 pixels by 59 pixels. Making a replacement image of similar size (the height is especially important) will change the logo atop the website! If your logo does not match the dimensions of the GeoMoose logo just edit the myskin.css file and change the padding-left and padding-right style properties for the logo to more closely match your images dimensions. If you change the background color of your logo and want the padding to match this color then change the color of the “menubar.png” image to match your logo’s background color.

Changing the Background Color of Menus The background color of both the toolbar and menubar (links menu) is controlled by editing the background color property style for each. For example, go to #toolbar and add background: red; or background rgb(255,0,0); to change the background property for the toolbar style to red.

Changing the contents of the Menubar The Menubar is the first menu under the logo above the toolbar. The links in this menu can be changed by editing the mapbook.xml file in the conf directory. Change the `links_bar_html` parameters as needed. See also, [links_bar_html](#) and [How To Put Content in the Menubar](#)

Moving the Links from Left to Right The layout in the header can also be changed by editing styles. For example, the menubar of links can be moved to be right justified by

1. removing the `display: block ;` property from the `#logo` and `#menubar` styles,
2. adding a `float: right;` property to `#menubar`, and
3. adding `display: block;` and `clear: both;` properties to the `#toolbar` style.

How To Change Default Drawing Colors

GeoMOOSE 2 features client side red-lining/drawing/sketching tools. These tools allow the user to place their own markup on the map and edit that markup. Currently, the shapes drawn can have their shape, fill color, and stroke color adjusted after creation. Default values can be changed within the draw service section in the mapbook:

```
<attribute name="line_color" type="color" default-value="#ff0000" label="Stroke Color:"/>
<attribute name="fill_color" type="color" default-value="#ff0000" label="Fill Color:"/>
<attribute name="label_only" type="checkbox" default-value="false" label="Only show label in print?">
```

These colors can be any one of the 16 supported W3C colors (red, yellow, etc), a six digit RGB color code (ex., #00FF00), or a three digit RGB color code (ex., #0F0).

How to set the Defaults for the Measure Tools

Units

The measure tool units are defined using configuration parameters. There is a parameter for line measuring units and area measuring units:

- `measure_tool.line_units = mi,ft,m`
- `measure_tool.area_units = mi,ft,m,yd,acre`

Translations:

- `mi` = Miles
- `ft` = Feet
- `m` = Meters
- `km` = kilometers
- `yd` = Yards
- `acre` = Acres

All of the `area_units` specifications are assumed to be square units (for example, `mi` = Square Miles, for area measurement). This example sets the default units to square Miles:

```
<param name="measure_tool.area_units">mi</param>
```

Precision

By default the measure tool will display three digits of precision. This can be changed via the `measure_tool.precision` configuration variable:

```
<param name="measure_tool.precision">4</param> <!-- measure tool will now display four digits of precision -->
```

Segments

By default the measure area tool will display “segment” areas. This can be changed via the `measure_tool.show_area_segments` configuration variable:

```
<param name="measure_tool.show_area_segments">false</param><!-- measure area tool will not display segments -->
```

4.5.2 Going Further Customizing Your App

How to use a WMS Service in a Different Projection

Not all WMS services use the projection that a GeoMOOSE application may use natively. In this case, it is necessary to reproject the request to display properly.

Step 1: Prerequisites

Before we can do the reprojection both projections must be defined! GeoMOOSE has EPSG:4326 predefined so if the WMS service uses EPSG:4326 you will not need to define this projection. Please review [Adding Projections](#) for instructions on creating projections and adding them to the interface.

Step 2: Configure the WMS map-source

To define a WMS map-source in a different projection the `projection` attribute needs to get set to the projection code:

```
<map-source name='nasa' type='wms' tiled='false' projection='EPSG:4326'>
  <url>http://onearth.jpl.nasa.gov/wms.cgi</url>
  <layer name="global_mosaic"/>
  <param name="format" value="image/jpeg"/>
</map-source>
```

Step 3: Add the Layer to the Catalog

This is just like any other layer:

```
<layer title='NASA Global Mosaic' src='nasa/global_mosaic'/>
```

How To Customize Your Waiting Message

Some services can take serious time to load. This is especially true of the printing service which needs to assemble many high resolution images. To help the user understand the application is running it is often useful to have a

waiting message. To customize the waiting message simply change the `waiting_html` configuration parameter in your mapbook:

```
<param name="waiting_html"><![CDATA[
    <center>
        <b>Please wait while this loads...</b>
    </center>
]]></param>
```

How to Add Google, MapQuest or Bing Layers

Notes on All Layers

- Commercial layers can only be used as basemap layers. You cannot elevate them any higher in the list.
- Commercial layers cannot be printed. The licensing agreements for these layers do not allow them to be printed.
- You must shift your map into Web Mercator (aka “the Google Projection”). Please read [Changing the Projection](#)

How to Add Google Layers

Step 1: Add the Google API to geomoose.html

- Open “geomoose.html” in a text editor.
- Find the `<script>` tag that contains ‘OpenLayers.js’ in the ‘src’ attribute.
- Then add the Google API, the code should look like this:

```
<script type="text/javascript" src="http://maps.google.com/maps?file=api&v=2&key=ABQIAAAAnfs7bKE
<script type="text/javascript" src="OpenLayers-2.8/OpenLayers.js"></script>
```

Step 2: Add the Google map-source Definitions Now it is necessary to put the map-source definitions into the Mapbook:

```
<map-source name="google_physical" type="google" google-type="physical">
    <layer name="all"/>
</map-source>

<map-source name="google_streets" type="google" google-type="streets">
    <layer name="all"/>
</map-source>

<map-source name="google_hybrid" type="google" google-type="hybrid">
    <layer name="all"/>
</map-source>

<map-source name="google_satellite" type="google" google-type="satellite">
    <layer name="all"/>
</map-source>
```

Step 3: Add the Definitions to the Catalog Google layers can only be used as basemap layers. So when using them as a background it is usually wise to simply create a backgrounds group and allow the user to toggle between the various layers:

```
<group title="Backgrounds" expand="true" multiple="false">
  <layer title="Streets" src="google_streets/all" status="on"/>
  <layer title="Physical" src="google_physical/all" status="off"/>
  <layer title="Hybrid" src="google_hybrid/all" status="off"/>
  <layer title="Satellite" src="google_satellite/all" status="off"/>
</group>
```

How To Change the Tabs

As of GeoMOOSE 2.6, the tabs system uses the Dojo conventions.

To add a custom tab follow and extend the example in *extensions/CustomTab.js*:

```
/*
 * GeoMOOSE Custom Tab Example.
 *
 * This creates a customized tab for the user interface
 * and then adds it to the control panel.
 *
 * TODO: This works, but we should really refactor it to use the
 * standard extension framework.
 */

dojo.declare('MyTab', [GeoMOOSE.Tab], {
  title: 'My Custom Tab',

  startup: function() {
    this.inherited(arguments);
    this.set('content', "This is an example of a <b>custom tab.</b>");
  }
});

dojo.addOnLoad(function() {
  GeoMOOSE.addTab('my_custom_tab', new MyTab());
});
```

But, I have a lot of text.

Sometimes it is not convenient to use JavaScript to populate a lot of text into a tab. We have an easy way to do this:

```
dojo.declare('MyTab', [GeoMOOSE.Tab], {
  title: 'My Custom Tab',

  startup: function() {
    this.inherited(arguments);
    this.set('href', 'custom_data.html');
  }
});

dojo.addOnLoad(function() {
  GeoMOOSE.addTab('my_custom_tab', new MyTab());
});
```

Instead of setting the HTML directly, it can be loaded from an external URL. The first method (DHTML) should be used when adding additional Dojo/Dijit objects to the tab, using the HTML will not work as the Dojo-tagged-elements

will not be properly parsed.

How To Call a Service at Startup

GeoMOOSE is centered around its services. Services are defined in the mapbook by the `<service>` tag and they define how GeoMOOSE should communicate with external scripts. There are a number of scenarios in which it is necessary to call a service on startup. The primary example of this functionality is the ability to zoom to and highlight a feature upon opening the interface.

Given the item query service provided with the demo:

```
<service name="search_parcel">
  <!-- The URL of the service to call -->
  <url>php/itemquery.php</url>

  <!-- most service have one or two "steps" this one is a non-spatial input -->
  <step type="input">

    <!-- the field to search against -->
    <input type="select" name="qitem" title="Search By:">
      <option value="OWNER_NAME">Owner</option>
      <option value="PID">Parcel ID</option>
    </input>

    <!-- this is the actual search string -->
    <input type="user" name="qstring" title=""/>

    <!-- these are the rest of the settings for the service -->
    <input type="hidden" name="layer" value="parcels/all"/>
    <input type="hidden" name="zoom_to_first" value="false"/>
    <input type="hidden" name="highlight" value="true"/>
  </step>
</service>
```

Using this service there are two important things to note:

1. The name of the service, “search_parcel”
2. The inputs of type “user”, these are the inputs that would require the user to enter text. We need to bypass that with parameters for the URL. In this example, the only “user” input is “qstring”.

Calling the URL

To call the service, the url must specify the “call” parameter and the appropriate user-inputs:

```
geomoose.html?call=search_parcel&qstring=Johnson
```

- The “call” parameter specifies the name of the service to call
- The “qstring” parameter is the same name of the input of type “user” and its value is set to “Johnson”

How To Avoid getmapbook.php

There are installations that do not use the PHP services provided with GeoMOOSE. If the reader is truly interested in this document then they are probably one of those exact users. This how to is written assuming you have a strong sense of how your server is configured and how the GeoMOOSE services system functions in your environment.

Why This is a Bad Idea

The mapbook is the all-knowing configuration file for GeoMOOSE. All of the built-in services use this file as the center definition for the application. There are also a few non-philosophical reasons why getmapbook.php exists:

- **Central Configuration.** Using the default configuration, to change the current mapbook, only one line in conf/settings.ini needs to change and all of the services and the main HTML/JS application switch to using that mapbook.
- **Future Security.** While GeoMOOSE 2.0 has not implemented the feature yet, we are actively building a security model that uses the getMapbook() function as the central marschall for authorization. This security feature will allow a single GeoMOOSE installation to be used for either internal or external use. It also promises to allow an organization to prevent differing options to users based on a login.

[/soapbox]

Why It's Okay

As stated in the introduction, if you're doing this, you should know what you're doing.

Here are a couple of reasons to override the mapbook setting:

- If you are having difficulty getting PHP working on your server and want to experiment with GeoMOOSE
- You want to use a different server-side scripting language (such as Python or ASP) and are not interested in using the other GeoMOOSE PHP services.

Step 1: Override the 'mapbook' Setting

The 'mapbook' attribute of CONFIGURAITON is the only one that cannot be overridden after the mapbook is loaded. Thus, some small changes need to be made to the geomoose.html file. Find the line that includes "compiled.js" and add the URL to your mapbook afterwards:

```
<script type="text/javascript" src="compiled.js"></script>
<script type="text/javascript">
    CONFIGURATION['mapbook'] = 'mapbook.xml';
</script>
```

Step 2: Change the settings.ini File (PHP Only)

It is possible to use the other PHP services with a mapbook over-ride present but they still need to know the location of that mapbook. The settings.ini file in the conf/ director contains such information.

conf/settings.ini:: [defaults] ; Location of the default mapbook mapbook=../htdocs/mapbook_demo.xml

How to Create Custom "Input" Types

From time to time is may be necessary to create a custom "input" type for a service with GeoMOOSE. This happens when you cannot get information about the map or from the user in a way that GeoMOOSE supports by default.

Step 1: Create an Input Type

Create the file, this one I called “MapWidth.js”:

```
GeoMOOSE.Services.InputType.MapWidth = OpenLayers.Class(GeoMOOSE.Services.InputType, {
    MAPBOOK_NAME: "map_width",

    getValue: function() {
        var extent = Map.getExtent();
        return extent.right - extent.left;
    }
});
```

Step 2: Add it to the “geomoose.html”

- Open “geomoose.html” in a text editor.
- Find `</head>` in the file.
- Add the following line:

```
<script type="text/javascript" src="MapWidth.js"></script>
```

Step 3: Add the input type to a Service

Find the service to add the type to and add the input definition:

```
<service ... >
    <input type="map_width" name="mw"/>
</service>
```

Step 4: Enjoy

This is the easiest part!

4.5.3 Other How To

How to Test, Report, and Ticket (in detail)

Background

GeoMOOSE uses [Trac](#) to keep track of documentation, tasks, bugs, enhancements, and most other details of the project. Trac is also the system for feedback about testing, particularly around a release. See also general faq *How_should_I_report_a_bug?*

Where to test?

Testing should be done on the [stable demo](#) or [trunk demo](#) depending on which you are reporting on. Another publically accesible instance can also be used if the demo is not configured with that specific functionality.

What to test?

Generally try to give the site a thorough check of the main functionality (search, identify, print, zoom, measure, layer tools, select) and specifically the issue that concerns you. This helps find any issues that may have been introduced as part of the change for your issue of concern.

Oops, there is an error. How to report this?

1. See also general faq *How_should_I_report_a_bug?*
2. Login to [Trac](#) which requires an OSGeo userid
3. [Open a ticket](#)
4. Document your browser(s) and GeoMOOSE version
5. Document the steps you could use to replicate the error
6. Document the error behavior
7. Make an effort to assign the correct Component, Type, Priority, Version, and useful keywords. For things you don't know, a guess or the default are ok. The Type for bugs is 'defect'

That works, but... How to request an enhancement?

1. See also, general faq *How_should_I_request_an_enhancement?*
2. Login to [Trac](#) which requires an OSGeo userid
3. [Open a ticket](#)
4. Identify start state
5. Document behavior your enhancement needs to do
6. Document end state
7. Make an effort to assign the correct Component, Type, Priority, Version, and useful keywords. For things you don't know, a guess or the default are ok. The Type for enhancements is 'enhancement'
8. Consider providing a patch, funding, testing, documentation or other efforts to make this a more likely enhancement

What happens now that there is a ticket?

1. Developer will ascertain that the work needs to be completed. If not the developer will add to notes
2. Developer will work and mark assigned status to "accepted"
3. Developer will document and comment on the programming completed
4. Developer will change status to "testing"
5. Trunk demo version will be updated

How do I follow up?

1. Pay attention and respond promptly when you get notification of activity on the ticket
2. Follow the steps identified in the original ticket to replicate the problem this time testing on the [trunk demo](#)
3. If problem has been addressed
 1. Enter what steps you did to verify the problem has been addressed
 2. Identify what browser and version of GeoMoose you tested this on (if relevant)
 3. Resolve problem as “fixed” in status
4. If problem still exists or causes additional problems
 1. Enter what steps you did to that still causes problem
 2. Identify what browser and version of GeoMoose you tested this on (if relevant)
 3. Convert action to: “reassign to” programmer that did the work
5. If you find a new problem then start this process again with the new problem.

Note: follow through with your reported tickets

Please note if you open a ticket you are responsible to ensure that when development is completed you will test to ensure that the work addresses the documented problem. When you open a ticket you are also committing to following this operating procedure.

How to Run Trunk

Background

To stay up to date as GeoMOOSE development happens, you may want to track trunk. To run the most current GeoMOOSE code you will need to svn checkout or svn export <http://svn.osgeo.org/geomoose/geomoose2/trunk/>. If reading yesterday’s newspaper will suffice, you may want to consider running the nightly builds since it is slightly less effort, see also *download*. Everytime you download and run that, you will have yesterday’s status.

How to

Windows (MS4W)

1. Have MS4W (MS4W version 3.0.4+ for GeoMoose 2.6) properly installed on your computer, see also [install MS4W](#),
2. Create /ms4w/apps/geomoose2/ if you don’t already have it (if you already have it, back it up or delete it)
3. Explore to /ms4w/apps/geomoose2/ and svn checkout <http://svn.osgeo.org/geomoose/geomoose2/trunk/>. Also consider svn export. (Note: you will need an svn client to do that, [TortoiseSVN](#) works well on Windows. Access it by right clicking in a window.)
4. Rename /ms4w/apps/geomoose2/conf/ms4w_local_settings.ini to local_settings.ini
5. If you don’t already have /ms4w/httpd.d/httpd_geomoose2_ms4w.conf, copy it from /ms4w/apps/geomoose2/ms4w/httpd.d/
6. If you don’t already have /ms4w/Apache/htdocs/geomoose2.pkg.html, copy it from /ms4w/apps/geomoose2/ms4w/Apache/htdocs/

7. Restart apache (/ms4w/apache-restart.bat)
8. Point browser to <http://localhost/geomoose2/geomoose.html>
9. To stay current with trunk: if you did svn checkout, then svn update in /ms4w/apps/geomoose2/ (if you did svn export do it again.)