# QUICK LINKS

<table>
<thead>
<tr>
<th>ZOO Kernel Documentation</th>
<th>ZOO Services Documentation</th>
<th>ZOO API Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installation</td>
<td>ZOO Community</td>
<td>Search mailing list</td>
</tr>
<tr>
<td>Development Guide</td>
<td>Deploying Web Processing Services using ZOO-Project – Examples of Python based WPS using PgRouting</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** The entire documentation is also available as a single PDF document and ePub document.
ABOUT ZOO DOCUMENTATION

ZOO Documentation is a collaborative process; once users contribute documentation through the ZOO Wiki, the Documentation Committee will then review the contributions to add them into the official docs. The committee is currently composed of:

Nicolas Bozon (nicolas.bozon at gmail.com)
Gérald Fenoy (gerald.fenoy at geolabs.fr)
Jeff McKenna (jmckenna at gatewaygeomatics.com)

A special thank to the ZOO-Project sponsors and knowledge partners for their support.

The committee would like to thank Venkatesh Raghavan for his help on promoting ZOO Workshop in many places.

The committee would like to thank Hirofumi Hayashi and Daisuke Yoshida for their help and Japanese translation effort for the ZOO-Project Workshop 2010.

The committee would also like to thank the early contributors to the documentation: Luca Delucchi, René-Luc D’Hont, Angelos Tzotsos and Guillaume Sueur.
The ZOO-Project documentation is released under the Creative Commons Attribution-ShareAlike 4.0 International Public License (CC-BY-SA). This license must be kept when editing or creating a new documentation file.
ZOO WIKI

Users wishing to contribute steps and documents should use the ZOO Wiki.
This section contains the official ZOO Project documentation.

5.1 ZOO Kernel Documentation

The following sections will assist you with the ZOO Kernel:

5.1.1 Introduction

Authors  Nicolas Bozon, Gérald Fenoy, Jeff McKenna

Last Updated  $Date: 2011-12-07 14:19:47 +0100 (Wed, 07 Dec 2011) $

ZOO Kernel is the heart of the ZOO. It is a powerful server-side C Kernel which makes it possible to manage and chain Web services, by loading dynamic libraries and handling them as on-demand Web services. The ZOO Kernel is written in the C language, but supports several common programming languages in order to connect to numerous libraries and models.

Using ZOO Kernel as a Web Processing Platform

ZOO Kernel works with Apache and can communicate with cartographic engines and Web mapping clients. It simply adds the WPS support to your spatial data infrastructure and your Web mapping application!

Note:  If you’d like some background on the WPS standard, head to: http://www.opengeospatial.org/standards/wps

Supported Languages

ZOO Kernel supports the following programming languages, and let’s you use them to create new ZOO Services from new or existing code:

<table>
<thead>
<tr>
<th>Language</th>
<th>ServiceProvider</th>
<th>DataStructure</th>
<th>Return</th>
</tr>
</thead>
<tbody>
<tr>
<td>C / C++</td>
<td>Shared Library</td>
<td>maps* M</td>
<td>integer</td>
</tr>
<tr>
<td>Fortran</td>
<td>Shared Library</td>
<td>CHARACTER*(1024) M(10,30)</td>
<td>integer</td>
</tr>
<tr>
<td>Java</td>
<td>Class File</td>
<td>HashMap</td>
<td>integer</td>
</tr>
<tr>
<td>Python</td>
<td>Module File</td>
<td>dictionary</td>
<td>integer</td>
</tr>
<tr>
<td>PHP</td>
<td>Script File</td>
<td>Array</td>
<td>integer</td>
</tr>
<tr>
<td>Perl</td>
<td>Script File</td>
<td>Object or Array</td>
<td>integer</td>
</tr>
<tr>
<td>JavaScript</td>
<td>Script file</td>
<td>Array/Object</td>
<td>Array/Object</td>
</tr>
</tbody>
</table>
5.1.2 Installation

**Authors** Nicolas Bozon, Gérald Fenoy, Jeff McKenna

**Last Updated** $Date: 2011-12-07 14:44:57 +0100 (Wed, 07 Dec 2011) $

This page provides documentation on how to compile then install the ZOO Kernel on Unix, Windows, and Mac OS X platforms.

**Prerequisites**

**Authors** Nicolas Bozon, Gérald Fenoy, Jeff McKenna, Luca Delucchi

**Last Updated** $Date: 2013-03-28 10:26:04 +0100 (Thu, 28 Mar 2013) $

**Table of Contents**

- Prerequisites
  - Obtaining the ZOO Kernel Source
  - Prerequisites
  - Compile libcgic
Obtaining the ZOO Kernel Source

Use the following command to get the ZOO Kernel source code through Subversion:

```
svn checkout http://svn.zoo-project.org/svn/trunk zoo-project
```

For users which get a developer account, use the following:

```
svn co svn+zoosvn://svn.zoo-project.org/var/svn/repos/trunk zoo-project
```

The first line of the instruction above defines a specific tunnel to access the svn server through the SSH protocol. Indeed, the ZOO SVN server listens on the 1046 (1024+22) port rather than the default one (22).

Prerequisites

The following libraries are required on your system before you can install the ZOO Kernel:

- cgic ([http://www.boutell.com/cgic](http://www.boutell.com/cgic))
- cURL ([http://curl.haxx.se](http://curl.haxx.se))
- FastCGI ([http://www.fastcgi.com](http://www.fastcgi.com))
- libxml2 ([http://xmlsoft.org](http://xmlsoft.org))
- OpenSSL ([http://www.openssl.org](http://www.openssl.org))
- Python ([http://www.python.org](http://www.python.org))

Optional libraries include:

- MapServer (optional for WMS, WFS and WCS output) ([http://mapserver.org](http://mapserver.org))
- PHP Embedded (optional) ([http://www.php.net](http://www.php.net))
- Java SDK (optional) ([http://java.sun.com](http://java.sun.com))
- SpiderMonkey (optional) ([http://www.mozilla.org/js/spidermonkey/](http://www.mozilla.org/js/spidermonkey/))

Compile libcgc

The first step is to compile libcgc from the `zoo-project/thirds` directory. For such a task, please use the following command:

```
cd thirds/cgic206
make
```

Make sure that a `libcgc.a` is created in your `zoo-project/thirds/cgic206` directory. If yes, then you can go to the next step.

On Windows, rather than using the `make` command, please use:

```
nmake /f makefile.vc
```

**Warning**: If you don’t compile libcgc first, and try to compile the ZOO Kernel, you will get an error such as *cannot find -lcgc*
Unix

Authors  Nicolas Bozon, Gérald Fenoy, Jeff McKenna, Luca Delucchi

Last Updated  $Date: 2013-03-28 15:10:57 +0100 (Thu, 28 Mar 2013) $

Table of Contents

• Unix
  – For the impatient
  – Configure Options

Note:  You must be sure to perform the prerequisite steps before following this page.

For the impatient

To build the zoo_loader.cgi CGI program with the default options, cd to the directory where you extracted the ZOO Kernel source code package and use the following commands:

$ cd zoo-kernel
$ autoconf
$ ./configure
$ make

Unless something went wrong, you should have executables in the current directory for the zoo_loader.cgi CGI program. You can copy the zoo_loader.cgi program and the main.cfg file to your HTTP server’s CGI directory and start using it.

At this step your ZOO-Kernel should work. Nevertheless, don’t forget to correct the main.cfg settings to set tmpPath and tmpUrl to fit your web server configuration.

Configure Options

Here is the list of available options as returned by ./configure --help:

--with-PACKAGE=[ARG]  use PACKAGE [ARG=yes]
--without-PACKAGE  do not use PACKAGE (same as --with-PACKAGE=no)
--with-gdal-config=FILE  specify an alternative gdal-config file
--with-xml2config=FILE  specify an alternative xml2-config file
--with-mapserver=PATH  specify the path for MapServer compiled source tree
--with-python=PATH  To enable python support or specify an alternative directory for python installation, disabled by default
--with-pyvers=NUM  To use a specific python version
--with-php=PATH  To enable php support or specify an alternative directory for php installation, disabled by default
--with-perl=PATH  To enable perl support or specify an alternative directory for perl installation, disabled by default
--with-java=PATH  To enable java support, specify a JDK_HOME, disabled by default
--with-js=PATH  specify --with-js=path-to-js to enable js support, specify --with-js on linux debian like, js support is disabled by default
All the options are described in more details below.

**(Required) GDAL Support**  If your gdal-config program is not found in your PATH then you can use the `--with-gdal-config` option to specify its location. For instance, let's suppose that your gdal-config was installed in `/usr/local/bin` and this directory is not in your PATH, then you can use the following command:

```
$ ./configure --with-gdal-config=/usr/local/bin/gdal-config
```

**(Required) XML2 Support**  If your xml2-config program is not found in your PATH then you can use the `--with-xml2config` option to specify its location. For instance, let's suppose that your xml2-config was installed in `/usr/local/bin` and this directory is not in your PATH, then you can use the following command:

```
$ ./configure --with-xml2config=/usr/local/bin/xml2-config
```

**(Optional) MapServer Support**  If you want to activate the WMS, WFS and WCS outputs using MapServer then you will have to use the `--with-mapserver` option. You have to set the path to your `mapserver-config` locate in the source code of MapServer as following command:

```
$ ./configure --with-mapserver=/path/to/your/mapserver_config/
```

**(Optional) Python Support**  If you want to activate Python support for the ZOO Kernel then you will have to use the `--with-python` option. If your python-config program is found in your PATH then you don’t have to specify the path where Python was installed, such as:

```
$ ./configure --with-python
```

This assumes that python-config is found in your PATH.

In the case that your python-config is not found in your PATH, then you can specify the Python installation directory you are using. For instance, let’s suppose that you installed Python in `/usr/local`, then you can use the following command:

```
$ ./configure --with-python=/usr/local
```

This assumes that `/usr/local/bin/python-config` exists.

**(Optional) Python Version**  If you want use a specific version of Python you will have to use the `--with-pyvers` option. You can specify a Python version as:

```
$ ./configure --with-pyvers=2.6
```

**(Optional) PHP Support**  To be able to activate PHP support for the ZOO Kernel you’ll need to get a local PHP Embedded installation; for more information about the required configure options when compiling PHP you can refer to this page:

[http://zoo-project.org/trac/wiki/ZooKernel/Embed/PHP](http://zoo-project.org/trac/wiki/ZooKernel/Embed/PHP)

If you want to activate the PHP support for the ZOO Kernel then you will have to use the `--with-php` option. If your php-config program is found in your PATH then you don’t have to specify the path where PHP was installed, then you can use the following command:

```
$ ./configure --with-php
```
This assumes that php-config is found in your PATH.

In the case that your php-config is not found in your PATH, then you can specify the PHP installation directory you are using. For instance, let’s suppose that you installed PHP in /usr/local, then you can use the following command:

```bash
$ ./configure --with-php=/usr/local
```

This assumes that /usr/local/bin/php-config exists.

(Optional) Perl Support  If you want to activate Perl support for the ZOO Kernel then you will have to use the `--with-perl` option. If you do not set any value to this option, then the perl program will be searched in your PATH. So in such a case, you can use the following command:

```bash
$ ./configure --with-perl
```

This assumes that perl is found in your PATH.

In the other case, for custom Perl installations, you can set the installation directory. For instance, let’s suppose that you installed Perl in /usr/local and /usr/local/bin is not in your PATH, then you can use the following command:

```bash
$ ./configure --with-perl=/usr/local
```

This assumes that /usr/local/bin/perl exists.

(Optional) Java Support  If you want to activate Java support for the ZOO Kernel then you will have to use the `--with-java` option and set the installation path of your Java SDK. For instance, let’s suppose that your Java SDK was installed in the /usr/lib/jvm/java-6-sun-1.6.0.22/ directory, then you can use the following command:

```bash
$ ./configure --with-java=/usr/lib/jvm/java-6-sun-1.6.0.22/
```

This assumes that the include/linux and jre/lib/i386/client/ subdirectories exist in /usr/lib/jvm/java-6-sun-1.6.0.22/, include/linux contains the jni.h headers file and jre/lib/i386/client/ contains the libjvm.so file.

Note:  With Mac OS X you only have to set `macos` as the value for the `--with-java` option to activate Java support. For example:

```bash
$ ./configure --with-java=macos
```

(Optional) JavaScript Support  If you want to activate JavaScript support for the ZOO Kernel then you will have to use the `--with-js` option. If you are using a “Debian-like” GNU/Linux distribution then dpkg will be used to detect if the required packages are installed and you don’t have to specify anything here, so you can use the following command:

```bash
$ ./configure --with-js
```

This assumes that js_api.h and libmozjs.so are found in default directories.

If you have a custom installation of SpiderMonkey or you are not using a Debian packaging system, then you’ll have to specify the directory where you installed it. For instance, let’s suppose that you installed your SpiderMonkey in /usr, then you’ll have to use the following command:

```bash
$ ./configure --with-js=/usr
```

This assumes that the /usr/include/js exists and contains the js_api.h headers file and /usr/lib contains libmozjs.so file.
Zoo-Kernel is maintained as a package in OpenSUSE Build Service (OBS). This way, rpm’s are provided for all versions of openSUSE Linux (11.2, 11.3, 11.4, Factory).

**Stable Releases**

For installing Zoo-Kernel in openSUSE there are 3 ways available:

- **One Click Installer** One-click installer that can be found here. For openSUSE 11.4 this is the direct link.

- **Yast Software Manager using a GUI** The Application:Geo repository has to be added to the Software Repositories and then Zoo-kernel can be found in Software Management through search.

- **Command line (as root for openSUSE 11.4)**

```bash
zypper ar http://download.opensuse.org/repositories/Application:/Geo/openSUSE_11.4/
zypper refresh
zypper install zoo-kernel
```

**Unstable Version**

The latest development version of ZOO-Kernel can be found in OBS under the project home:tzotsos. ZOO-Kernel packages are maintained and tested there before being released to the Application:Geo repository.

Installation methods are identical as the stable version. Make sure to use this repository instead.

**Command line installation (as root for openSUSE 11.4)**

```bash
zypper ar http://download.opensuse.org/repositories/home:/tzotsos/openSUSE_11.4/
zypper refresh
zypper install zoo-kernel
zypper install zoo-kernel-grass-bridge
```

Additionally, there is the option of adding the zoo-wps-grass-bridge package. This option will automatically install grass7 (svn trunk).
Try the Installation

- http://localhost/cgi-bin/zoo_loader.cgi?ServiceProvider=&metapath=&Service=WPS&Request=GetCapabilities&Version=1.0.0
- http://localhost/cgi-bin/zoo_loader.cgi?ServiceProvider=&metapath=&Service=WPS&Request=DescribeProcess&Version=1.0.0&Identifier=HelloPy
- http://localhost/cgi-bin/zoo_loader.cgi?ServiceProvider=&metapath=&Service=WPS&Request=Execute&Version=1.0.0&Identifier=HelloPy&DataInputs=a=myname

CentOS

Authors Nicolas Bozon, Gérald Fenoy, Jeff McKenna

Last Updated $Date: 2011-12-07 14:44:57 +0100 (Wed, 07 Dec 2011)$

Table of Contents

- CentOS
  - Requirements
  - Compile ZOO-Kernel and ZOO-Services
  - Testing your ZOO-Kernel

Note: This documentation was created thanks to Guillaume Sueur from Neogeo Technologies which took time to test installing the ZOO-Kernel on a CentOS 5.5 environment.

Requirements

Install some standard tools to be able to run ZOO-Kernel on your platform:

yum install apache2
yum install build-essentials
yum install gcc-c++
yum install zlib-devel
yum install libxml2-devel
yum install bison
yum install openssl
yum install python-devel
yum install subversion

Compile then install FastCGI library from source

wget http://www.fastcgi.com/dist/fcgi.tar.gz
tar xzf fcgi-2.4.0.tar.gz
./configure
make
make install
echo /usr/local/lib >> /etc/ld.so.conf.d/local.conf
ldconfig

Compile then install the autoconf tools:

tar xzf autoconf-latest.tar.gz
./configure --prefix=/usr
make
make install

Compile then install the flex tool:

```
wget http://downloads.sourceforge.net/project/flex/flex/flex-2.5.35/flex-2.5.35.tar.gz?r=http%3A%2F%2F
  tar xzf flex-2.5.35.tar.gz
  cd flex-2.5.35
  ./configure --prefix=/usr
  make
  make install
```

Using the curl provided in the CentOS distribution will produce a ZOO-Kernel unable to run any Service. Indeed, some segmentation faults occur when trying to run Execute requests on the ZOO-Kernel, compiling the ZOO-Kernel setting USE_GDB flag in the CFLAGS of your Makefile will let you run ZOO-Kernel from gdb and be able to get more information on what is going wrong with your ZOO-Kernel. Doing this we can figure out that code on line 173 and line 175 have to be commented in the ulinet.c file to get a ZOO-Kernel working using the curl available in CentOS (curl version 7.15.5). If you don’t apply the modification, you will get an error from a gdb session pointing segfault in Curl_cookie_clearall.

You can optionally compile then install curl from source:

```
wget http://curl.haxx.se/download/curl-7.21.3.tar.bz2
  tar xjf curl-7.21.3.tar.bz2
  cd curl-7.21.3
  ./configure --prefix=/usr
  make
  make install
```

Compile then install Python:

```
wget http://www.python.org/ftp/python/2.6.6/Python-2.6.6.tar.bz2
  tar xjf Python-2.6.6.tar.bz2
  cd Python-2.6.6
  ./configure
  make
  make install
```

Compile then install your own GDAL library:

```
wget http://download.osgeo.org/gdal/gdal-1.7.3.tar.gz
  tar xzf gdal-1.7.3.tar.gz
  cd gdal-1.7.3
  ./configure  # add your options here
  make
  make install
```

Install the Sun JAVA SDK into /usr/share then use the following command to ensure that the libjvm.so will be found at runtime from any context.

```
echo /usr/share/java-1.6.0-openjdk-1.6.0.0/jre/lib/i386/client/ >> /etc/ld.so.conf.d/jvm.conf
  ldconfig
```

### Compile ZOO-Kernel and ZOO-Services

Compile then install ZOO-Kernel and your first ZOO-Services.

First of all, compile the cgic library providen in the SVN source tree:

```
```
svn co http://svn.zoo-project.org/svn/trunk zoo-project
 cd zoo-project/thirds/cgic206
 make

Compile then install ZOO-Kernel.
 cd ../../../zoo-kernel
 ./configure --with-java=/usr/share/jdk1.6.0_23/ --with-python
 make zoo_loader.cgi
 cp main.cgi /var/www/cgi-bin/
 cp  zoo_loader.cgi /var/www/cgi-bin/

Compile then deploy your first ZOO-ServicesProviders (simple HelloPy, line 1 and 2, and the OGR base-vect-ops ServiceProvider, line 3 to 6):
 cp ../zoo-services/hello-py/cgi-env/*.zcfg /var/www/cgi-bin/
 cp ../zoo-services/hello-py/test_service.py /var/www/cgi-bin/
 cd ../ogr/base-vect-ops/
 make
 cp ./cgi-env/* /var/www/cgi-bin/
 vi /var/www/cgi-bin/main.cfg --> set your own informations here

To ensure that the libjvm.so will be found from apache, please restart it:
 /etc/init.d/httpd restart

Testing your ZOO-Kernel

Test your ZOO-Kernel from command line:
 cd /var/www/cgi-bin
 ./zoo_loader.cgi "request=Execute&service=WPS&version=1.0.0&Identifier=HelloPy&DataInputs=a=Djay"
 ./zoo_loader.cgi "request=Execute&service=WPS&version=1.0.0&Identifier=Buffer&DataInputs=BufferDistance=1"
Ubuntu 12.04 dependencies

- install default dependencies

```bash
sudo apt-get install flex bison libfcgi-dev libxml2 libxml2-dev curl openssl autoconf apache2 python-software-properties subversion libmozjs185-dev python-dev build-essential
```

- add ubuntugis repository to obtain the newer GIS libraries

```bash
sudo add-apt-repository ppa:ubuntugis/ppa
sudo apt-get update
```

- install geographic library

```bash
sudo apt-get install libgdal1-dev
```

- download ZOO source

Debian 7.0 dependencies

- install dependencies

```bash
apt-get install flex bison libfcgi-dev libxml2 libxml2-dev curl openssl autoconf apache2 python-dev libgdal1-dev build-essential libmozjs185-dev
```

Installation Workflow

---

**Note:** In the following lines you will find as administrator user note, on Ubuntu you have to add sudo at the beginning of line, on Debian you can use su -c COMMAND

```bash
svn checkout http://svn.zoo-project.org/svn/trunk zoo-project
```

- install cgic from packages

```bash
cd zoo-project/thirds/cgic206/
```

- compile

```bash
make
```

- if you want use Mapserver (optional) for WMS, WFS and WCS output read Compile MapServer

- go to kernel path

```bash
cd ..../zoo-project/zoo-kernel/
```

- create configure file

```bash
autoconf
```

- run configure

```bash
./configure --with-js --with-python
```

---

**Note:** For PHP, you must make sure to compile PHP with --enable-embed.

---

**Note:** To use MapServer add the option --with-mapserver=/path/to/mapserver/source
• compile

make

• copy necessary files into your cgi-bin (as administrator user)

cp main.cfg /usr/lib/cgi-bin

cp zoo_loader.cgi /usr/lib/cgi-bin

• install ZOO ServiceProvider, in this case we try Python service (as administrator user)

cp ../zoo-services/hello-py/cgi-env/*.zcfg /usr/lib/cgi-bin

cp ../zoo-services/hello-py/*.py /usr/lib/cgi-bin/

• change some information in the main.cfg (as administrator user)

nano /usr/lib/cgi-bin/main.cfg
- serverAddress = http://127.0.0.1

• try the installation
  – http://127.0.0.1/cgi-bin/zoo_loader.cgi?ServiceProvider=&metapath=&Service=WPS&Request=GetCapabilities&Version=
  – http://127.0.0.1/cgi-bin/zoo_loader.cgi?ServiceProvider=&metapath=&Service=WPS&Request=DescribeProcess&Version=
  – http://127.0.0.1/cgi-bin/zoo_loader.cgi?ServiceProvider=&metapath=&Service=WPS&Request=Execute&Version=1.0.0&

Note: If you have some problem in the execute request using Python service, add the following to main.cfg:

[env]
PYTHONPATH=<YOUR_PYTHONPATH>

Rewrite rule configuration

• for better readability and fully functional ZOO Kernel, you have to modify the default Apache configuration in order to be able to use the http://localhost/zoo/ url directly. Run (as administrator user)

nano /usr/lib/cgi-bin/main.cfg
- serverAddress = http://localhost/zoo

• first, please create a zoo directory in /var/www/ which is used by Apache as the DocumentRoot (as administrator user)

mkdir /var/www/zoo

• Then, please edit the /etc/apache2/sites-available/default configuration file and add the following lines after the Directory block related to /var/www directory (as administrator user)

<Directory /var/www/zoo/>
  Options Indexes FollowSymLinks MultiViews
  AllowOverride All
  Order allow,deny
  allow from all
</Directory>

• now create a small .htaccess file in the /var/www/zoo containing the following lines (as administrator user)
RewriteEngine on
RewriteRule call/(.*)/(.*) /cgi-bin/zoo_loader.cgi?request=Execute&service=WPS&version=1.0.0&Identifier=$1 [L,QSA]
RewriteRule (.*)/(.*) /cgi-bin/zoo_loader.cgi?metapath=$1 [L,QSA]
RewriteRule (.*) /cgi-bin/zoo_loader.cgi [L,QSA]

- for this last file to be taken into account by Apache, you must activate the rewrite Apache module by copying a load file as bellow (as administrator user)
cp /etc/apache2/mgs-available/rewrite.load /etc/apache2/mgs-enabled/

- or using the a2enmod tool this way (as administrator user)
a2enmod rewrite

- now you should be able to access the ZOO Kernel using a simplified by restarting your Apache Web server (as administrator user)

:: /etc/init.d/apache2 restart

- now you can try:
  - http://localhost/zoo/?Service=WPS&Request=GetCapabilities&Version=1.0.0
  - http://localhost/zoo/?Service=WPS&Request=DescribeProcess&Version=1.0.0&Identifier=HelloPy
  - http://localhost/zoo/?Service=WPS&Request=Execute&Version=1.0.0&Identifier=HelloPy&DataInputs=a=myname

Compile MapServer

- install MapServer dependencies (as administrator user)
  apt-get install git libfreetype6-dev libproj-dev libkml-dev libcairo-dev libxml2-dev libgd2-xpm-dev libxslt1-dev

- extract MapServer from git repository
  git clone https://github.com/mapserver/mapserver.git
cd mapserver
git checkout branch-6-2
git pull

- now compile MapServer
  ./configure --with-ogr --with-gdal --with-wfsclient --with-wmsclient --with-tiff --with-jpeg --with-gd
  make

- now install MapServer (as administrator user)
  make install

- now add path to libraries path (as administrator user)
nano /etc/ld.so.conf.d/mapserver.conf
  - /usr/local/lib/

- run as administrator
  ldconfig
Using OSGeo4W

Install OSGeo4W  Download the OSGeo4W installer from http://trac.osgeo.org/osgeo4w/, and install it with all the dependencies needed by your services (GDAL/OGR for example). The following libs are required: FastCGI, libxml, Python, cURL.

Install other tools and libraries  After installing OSGeo4W on your platform you’ll need more GNU tools and libraries. This package contains full dependencies required to compile on WIN32 platform and this one contains full runtime dependencies to place in your c:OSGeo4Wbin.

Download and Install ZOO Kernel  Download the binary version of the ZOO Kernel for WIN32 then place it in the C:\OSGeo4W\bin directory. Don’t forget to place a main.cfg file in the same directory, you can use a modified copy of this file.

Deploy ZOO Services Providers  Your can use the binary version of the OGR Services Provider available from here. Then place the two libraries with their respective .zcfg files in your local C:\OSGeo4W\bin directory.

Testing  Now you should be able to query your local ZOO Kernel.

Compiling Using Your Own Libraries

Note:  You must be sure to perform the prerequisite steps before compiling the ZOO Kernel.

The following steps are for use with the Microsoft Visual Studio compiler (and tested with MSVC 2008).

1. Make sure the gnuwin32 tools bison.exe and flex.exe are found in your path. You can download the GNUwin32 tools here.
2. Modify the file zoo-project\zoo-kernel\nmake.opt to point to your local libraries. You can find a modified nmake.opt that points to local libs here. You can also find a modified zoo-project\zoo-kernel\makefile.vc file here.
3. Execute:

   nmake /f makefile.vc

4. A file zoo_loader.cgi should be created. Note that if another file named zoo_loader.cgi.manifest is also created, you will have to run another command:
5. Copy the files zoo_loader.cgi and main.cfg into your cgi-bin directory.

6. Using the command prompt, test the zoo-kernel by executing the following command:

```
D:\ms4w\Apache\cgi-bin> zoo_loader.cgi
```

which should display a message such as:

```
Content-Type: text/xml; charset=utf-8
Status: 200 OK

<?xml version="1.0" encoding="utf-8"?>
  <ows:Exception exceptionCode="MissingParameterValue">
    <ows:ExceptionText>Parameter &lt;request&gt; was not specified</ows:ExceptionText>
  </ows:Exception>
</ows:ExceptionReport>
```

7. Edit the file cgi-bin/main.cfg so that it contains values describing your WPS service. An example of such a file running on Windows is:

```
[main]
encoding = utf-8
version = 1.0.0
serverAddress = http://localhost/
lang = en-CA
tmpPath=/ms4w/tmp/ms_tmp/
tmpUrl = /ms_tmp/

[identification]
title = The Zoo WPS Development Server
abstract = Development version of ZooWPS. See http://www.zoo-project.org
fees = None
accessConstraints = none
keywords = WPS,GIS,buffer

[provider]
providerName=Gateway Geomatics
providerSite=http://www.gatewaygeomatics.com
individualName=Jeff McKenna
positionName=Director
role=Dev
adressDeliveryPoint=1101 Blue Rocks Road
adressCity=Lunenburg
adressAdministrativeArea=False
adressPostalCode=B0J 2C0
adressCountry=ca
adressElectronicMailAddress=info@gatewaygeomatics.com
phoneVoice=False
phoneFacsimile=False
```

8. Open a web browser window, and execute a GetCapabilities request on your WPS service: `http://localhost/cgi-bin/zoo_loader.cgi?request=GetCapabilities&service=WPS`

The response should be displayed in your browser, such as:

```
<wps:Capabilities xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://schemas.opengis.net/wps/1.0.0/wpsGetCapabilities_response.xsd" service="WPS" xml:lang="en-US" version="1.0.0">
  <ows:ServiceIdentification>
    <ows:ServiceType>WPS</ows:ServiceType>
    <ows:ServiceTypeCode>1.0.0</ows:ServiceTypeCode>
    <ows:OperationsMetadata>
      <ows:OperationMetadata operationName="GetCapabilities"/>
    </ows:OperationsMetadata>
    <ows:ServiceTypeAbstract>Services that provide execution of processes defined in the Web Processing Service (WPS) standard</ows:ServiceTypeAbstract>
    <ows:ContactInformation>
      <ows:ServiceProviderName>Gateway Geomatics</ows:ServiceProviderName>
      <ows:ServiceProviderAddress>1101 Blue Rocks Road, Lunenburg, NS, Canada</ows:ServiceProviderAddress>
      <ows:ServiceProviderPhone>902-652-0000</ows:ServiceProviderPhone>
      <ows:ServiceProviderEmailAddress>info@gatewaygeomatics.com</ows:ServiceProviderEmailAddress>
    </ows:ContactInformation>
    <ows:ContactRole>Developer</ows:ContactRole>
    <ows:AuthorizationInfo>
      <ows:ImplementationVersion>1.0.0</ows:ImplementationVersion>
    </ows:AuthorizationInfo>
    <ows:SupportedClientLanguages>
      <ows:Language>en-CA</ows:Language>
    </ows:SupportedClientLanguages>
    <ows:LanguageSupportDescription>Supported languages</ows:LanguageSupportDescription>
  </ows:ServiceIdentification>
  <ows:Abstract>Development version of ZooWPS. See http://www.zoo-project.org</ows:Abstract>
```

```
5.1. ZOO Kernel Documentation
```

25
Optionally Compile Individual Services  An example could be the OGR base-vect-ops provider in the zoo-project\zoo-services\ogr\base-vect-ops directory.

1. Edit the makefile.vc located in that directory, and execute:

   nmake /f makefile.vc

   Inside that same directory, the file cgi-env\ogr_service.zo should be created.

2. Copy all of the files inside zoo-services\ogr\base-vect-ops\cgi-env into your cgi-bin directory

3. Test this service provider through the following URL:

   http://localhost/cgi-bin/zoo_loader.cgi?request=Execute&service=WPS&version=1.0.0&Identifier=Buffer&DataInputs=BufferDistance=1@datatype=interger;InputPolygon=Reference@xlink:href=http%3A%2F%2Fwww.zoo-project.org%3A8082%2Fgeoserver%2Fows%3FSERVICE%3DWFS%26REQUEST%3DGetFeature%26VERSION%3D1.0.0%26SRS%3DEPSG%3A4326%26FeatureID%3Dstates.15

   The response displayed in your browser should contain:

   <wps:ProcessSucceeded>Service "Buffer" run successfully.</wps:ProcessSucceeded>

Mac OS X

Authors  Nicolas Bozon, Gérard Fenoy, Jeff McKenna

Last Updated  SDate: 2011-12-07 14:44:57 +0100 (Wed, 07 Dec 2011) $
ZOO Project Documentation, Release 1.4

/library/WebServer/CGI-Executables, and the zoo-demo folder will be placed within your document root at /library/WebServer/Documents

2. Make sure that your Apache server is running, and then access the ZOO Project Demo at:
   http://localhost/zoo-demo/spatialtools.html

3. To add additional services, please follow the following instructions to compile your own ZOO Project instance.

Compiling from Source

1. Install Xcode.
2. Before you start downloading the ZOO-Project source code, you’ll need to install some tools required to compile ZOO-Kernel properly.
   First of all install PROJ, GEOS and GDAL frameworks from here.
   At this step, you should get the following directories on your local hard drive:
   /Library/Frameworks/PROJ.framework
   /Library/Frameworks/GEOS.framework
   /Library/Frameworks/GDAL.framework

3. Then, create a src directory and inside that directory download the gettext source code and uncompress it.
   now, compile gettext with the following commands to produce a universal binary:
   cd gettext-0.18.1.1
   CFLAGS="-O -g -arch i386 -arch ppc -arch x86_64" \  
   LDFLAGS="-arch i386 -arch ppc -arch x86_64" ./configure
   make
   sudo make install

4. Compile and install your ZOO-Kernel
   • Download source from SVN, and use the following command to compile libcgc:
     svn co http://svn.zoo-project.org/svn/trunk zoo
     cd zoo/thirds/cgic206
     make
   • If you produced the libcgc.a file, you can run autoconf and then configure from zoo-kernel directory.
     cd zoo/zoo-kernel
     autoconf
     ./configure --with-python --with-java=macos \ 
     --with-gdal-config=/Library/Frameworks/GDAL.framework/Versions/1.8/Programs/gdal-config
   Obviously, if you don’t need Python or Java support then you should remove the corresponding configure option.

   **Note:** Note that we used the --with-java=macos configure option. Due to the generic location of the JDK on all Mac OS X platforms, you don’t have to provide its full path.

   • Now, run the following commands to compile and deploy your ZOO-Kernel on your Apache server:
You should be ready to request your ZOO-Kernel installation using the following link: http://localhost/cgi-bin/zoo_loader.cgi?request=GetCapabilities&service=WPS.

If everything is ok, you can follow the next steps to deploy new Services Providers.

**Note:** If you are using your own libs (not the default libs on your system) then you must take care to create universal versions of those libs, as the ZOO-Kernel will try to create a universal binary. If you are not following this advice, you might receive compile errors of symbol(s) not found for architecture ppc or file was built for unsupported file format which is not the architecture being linked (ppc).

### Deploy the OGR Services Provider

**Requirements** Before your try to use any service, please set the correct path in the `main.cfg` for tmpPath and tmpUrl.

You can use the following setup:

```
tmpPath = /Library/WebServer/Documents/tmp
tmpUrl = ../../tmp
```

Obviously you’ll then need to create this directory, using the following command:

```
mkdir /Library/WebServer/Documents/tmp
```

**C Version** To compile the base-vect-ops ServicesProvider you’ll need to edit the Makefile in `zoo/zoo-services/ogr/base-vect-ops/` directory. Add “-I/Library/Frameworks/GEOS.framework/Versions/3/Headers/” to the CFLAGS value on the first line. To compile, add GDAL framework to the PATH environmenet variable, to ensure that gdal-config tool will be found, run make and then copy cgi-env files in the `/Library/WebServer/CGI-Executables` directory.

```
cd zoo/zoo-services/ogr/base-vect-ops/
export PATH=$PATH:/Library/Frameworks/GDAL.framework/Versions/1.7/Programs/
make
cp cgi-env/* /Library/WebServer/CGI-Executables
```

You can test using this url if everything is ok with your setup.

**Python Version Requirements**

First of all run python from a Terminal.app and try the following import from the python interpreter:

```
import osgeo.ogr
import libxml2
```

If you get an issue when importing the libxml2 module from your python interpreter then that means you need to install the Python support for the libxml2 library which is already installed on your Mac OS X environment. To accomplish this, you have first to determine what version of libxml2 is installed on your platform, using the following command:

```
xmll2-config --version
```

Download the source corresponding to your version (i.e. on 10.6.6 you get 2.7.3) from the libxml2 [download page](#) into your `src` directory then uncompress it.
Use the following command to install the python support:

```bash
cd src/libxml2-2.7.3/python/
python setup.py install
```

**Deploy OGR Python Services Provider**

- Now copy the `zoo-services/ogt/base-vect-ops/cgi-env` files into `/Library/WebServer/CGI-Executables`.

You can test using this url if everything is ok with your setup.

**Test using Local Demo Page**

- Download the OpenLayers library and uncompress it in your personal Sites directory (located in your home directory).
- Rename the OpenLayers directory as openlayers.
- Download this zip archive and then uncompress it in your personal Sites directory.
- Load your local demo pages using urls similar to the following (replacing MyUserName by your MacOS user name):
  - `http://localhost/~MyUserName/zoo-demo/spatialtools.html`
  - `http://localhost/~MyUserName/zoo-demo/spatialtools-py.html`

### 5.1.3 How To Use the Internal MapServer W*S support

**Authors** Nicolas Bozon, Gérald Fenoy, Jeff McKenna

**Last Updated** $Date: 2015-02-09 16:26:31 +0100 (lun. 09 févr. 2015) $

The key idea of the MapServer W*S support implementation is that it doesn’t require to change a single line of the service source code to activate the automatic publication of your result as WMS/WFS or WCS ressource. You simply need to modify the zcfg file corresponding to your service to make it working.

Here is an overview of the way to install the MapServer W*S support, the configuration required and the internal mechanisms.

**Table of Contents**

- How To Use the Internal MapServer W*S support
  - How to make it working ?
    - Requirement
    - Installation steps
    - Configuration steps
  - How does it work ?
  - Simple sample use cases
    - Testings
How to make it working?

Requirement

- last ZOO-Kernel trunk version
- MapServer version >= 6.0.1

Installation steps

First download lastest zoo-kernel directory available on the svn, do that from the directory of your previous checkout (so where zoo-api, zoo-services and zoo-kernel directories are available), we will use `<PREV_SVN_CO>` here for this directory:

```shell
cd <PREV_SVN_CO>
svn checkout http://svn.zoo-project.org/svn/trunk/zoo-kernel zoo-kernel-ms
```

Uncompress the MapServer archive (ie. `mapserver-6.0.1.tar.bz2`) into `/tmp/zoo-ms-src`, then compile MapServer using the following command:

```shell
cd /tmp/zoo-ms-src/mapserver-6.0.1
./configure --with-ogr=/usr/bin/gdal-config --with-gdal=/usr/bin/gdal-config \   --with-proj --with-curl --with-sos --with-wfsclient --with-wmsclient \   --with-wcs --with-wfs --with-postgis --with-kml=yes --with-geos \   --with-xml --with-xslt --with-threads --with-cairo
make
cp mapserv /usr/lib/cgi-bin
```

Autotools was updated to add the `--with-mapserver` configure option. From your ZOO-Project SVN trunk directory, compile the ZOO-Kernel using the following command:

```shell
cd zoo-kernel-ms
autoconf
./configure --with-python --with-mapserver=/tmp/zoo-ms-src/mapserver-6.0.1
make
cp zoo_loader.cgi /usr/lib/cgi-bin
```

Configuration steps

Main configuration file

Add the following content to your `/usr/lib/cgi-bin/main.cfg` file in the `[main]` section:

```ini
dataPath = /var/www/temp/
mapserverAddress=http://localhost/cgi-bin/mapserv
```

The `dataPath` directory should exists and be writable by apache user. In this directory, a `symbols.sym` file have to be present, containing the following:

```ini
SYMBOLSET
SYMBOL
NAME "circle"
TYPE ellipse
FILLED true
POINTS
 1 1
```
Only one symbol definition is required with any name, used for WMS service output.

Now, your ZOO-Kernel get the MapServer support ready to be used. Note that if you don’t add the mapserverAddress then it imply the ZOO-Kernel will segfault (checking NULL value should correct this behavior).

Here you can optionally add a msOgcVersion parameter to specify which version of the OGC WebService you want to use for each services. For example, if you want to force to version 1.0.0, you can set the following in the [main] section of your main.cfg file:

```
msOgcVersion=1.0.0
```

**Service configuration file** To activate MapServer WebServices output for a service, you have to add a specific parameter in the `<Default>` or `<Supported>` block: useMapserver. This can take the value true or should not appear. If true, it means that the output result is an OGR / GDAL compatible datasource and you want it to be outputted as an OGC web server instance (WMS/WFS/WCS).

You get an optional parameter, to use a custom MapServer style block (used for vector datasource only): msStyle. For example:

```
msStyle = STYLE COLOR 125 0 105 OUTLINECOLOR 0 0 0 WIDTH 3
```

You get the same optional parameter msOgcVersion as for the main.cfg. This will specify that this is the specific protocol version the service want to use (so you may set also locally to service rather than globally).

When you add useMapserver option to an output `<Default>` or `<Supported>` block, then you have to know what are the sensible mimeType:

- `text/xml`: will imply that the output data will be accessible through a WFS GetFeature request (default protocol version 1.1.0)
- `image/tiff`: will imply that the output data will be accessible through a WCS GetCoverage request (default protocol version 2.0.0)
- any other mimeType coupled with useMapserver option: will imply that the output data will be accessible through a WMS GetMap request (you have to limit yourself to what your MapServer installation support, GetCapabilities? request give information of supported output mimeType) (default protocol version 1.3.0)

**How does it work?**

Whatever your service return as default output mimeType, this one will be used when one output including the useMapserver option was found.

So if you get the following `<Default>` and `<Supported>` blocks in the ZOO Configuration File of your service:

```
<Default>
  mimeType = text/xml
  encoding = UTF-8
  schema = http://schemas.opengis.net/gml/3.1.0/base/feature.xsd
</Default>
<Supported>
  mimeType = image/png
  useMapserver = true
</Supported>
```
It means that per default, your service return GML 3.1 Feature. When the client request for `mimeType=image/png`, then the ZOO-Kernel will detect that this `mimeType` get the `useMapServer` option set to true so it will:

1. execute the service using the `<Default>` block definition (this should be understandable by GDAL/OGR)
2. store the result of the service on disk (in the `[main] > dataPath` directory)
3. write a Mapfile (in the `[main] > dataPath` directory) using the MapServer C-API to setup both WMS and WFS services.

**Note:** even if you don’t ask for this, the resulting Mapfile includes both configuration for WMS and WFS in case of Vector datasource.

If your service output a raster file, then the behavior is quite the same except that the ZOO-Kernel will setup both WMS and WCS services for the result of the service. Here you cannot define your own style. Nevertheless, when one band raster is returned then the ZOO-Kernel can use its own default style definitions to classify the raster using equivalent intervals (you can easily see that in the outputed Mapfile), this classification is specific to WMS protocol. You should add a `msClassify` parameter and set it to `true` in your output `ComplexData <Default>` or `<Supported>` node to activate this classification. Special note for client implementers

Note that depending on the request, the ZOO-Kernel can return a location header.

Different request types:

- `ResponseDocument=XXXX@asReference=true` - in this case, the Kernel will return the GetMap/GetFeature/GetCoverage request in KVP in the href of the result.
- `ResponseDocument=XXXX@asReference=false` - in this case, the Kernel will return the result he get using the GetMap/GetFeature/GetCoverage request in KVP used for the href in previous case.
- `RawDataOutput=XXXX@asReference=true/false` - in this case, the Kernel will return the GetMap/GetFeature/GetCoverage request in KVP in the specific location header, which imply that the browser should follow and request MapServer directly.

**Simple sample use cases**

Consider the existing BufferPy service from zoo-services/ogr-base-vect-ops-py. Set the following content to your local BufferPy.zcfg file in the Result output definition, then copy it to `/usr/lib/cgi-bin/`:

```xml
<Default>
mimeType = text/xml
encoding = UTF-8
schema = http://schemas.opengis.net/gml/3.1.0/base/feature.xsd
useMapserver = true
</Default>
<Supported>
mimeType = image/png
useMapserver = true
asReference = true
msStyle = STYLE COLOR 125 0 105 OUTLINECOLOR 0 0 0 WIDTH 3 END
</Supported>
<Supported>
mimeType = image/tif
useMapserver = true
asReference = true
msStyle = STYLE COLOR 125 0 105 OUTLINECOLOR 0 0 0 WIDTH 3 END
</Supported>
```
This modifications make your service ready to return result as WMS GetMap or WFS GetFeature requests. Note that some bug occurs locally using the application/vnd.google-earth.kmz output. Raster

Using the following simple service code we get a service capable to output any kind of internet files (useful for testing this functionality):

```python
import zoo
def HelloPy(conf,inputs,outputs):
    outputs["Result"]['value'] = inputs['a']['value']
    return zoo.SERVICE_SUCCEEDED
```

Define the [Result] output in your HelloPy.zcfg file with the following ComplexData block content:

```xml
<Default>
mimeType = image/png
useMapServer = true
</Default>
<Supported>
mimeType = image/tiff
useMapServer = true
</Supported>
<Supported>
mimeType = text/xml
useMapServer = true
</Supported>
```

It means that the default output mimeType is image/png, so a WMS GetMap request will be returned, or the resulting image/tiff will be returned as WCS GetCoverage request.

With this simple service you can test the new capabilities to output result as WebServices for each mimeTypes. Note, that you'll probably get wrong mimeType, as the default was set to image/png.

There is a support for Zipped ShapeFile but I doubt it is really useful. Anyway, as it is present you can test it easily by passing a zip file in xlink:href for the a value of the HelloPy service.

**Testings**

Using the simple HelloPy service code, you can use the following urls, note it supposes that you get an available http://localhost/data/data.zip file containing a ShapeFile and a http://localhost/data/demo.tif:

**Test 1: Accessing a remote Zipped Shapefile as WFS GetFeatures Request**

http://localhost/cgi-bin/zoo_loader.cgi?request=Execute&service=WPS&version=1.0.0&Identifier=HelloPy&DataInputs=a=Reference@xlink:href=http://localhost/data/data.zip&ResponseDocument=Result@asReference=true@mimetype=text/xml

**Test 2: Accessing a remote Zipped Shapefile as WMS GetMap Request**

http://localhost/cgi-bin/zoo_loader.cgi?request=Execute&service=WPS&version=1.0.0&Identifier=HelloPy&DataInputs=a=Reference@xlink:href=http://localhost/data/data.zip&ResponseDocument=Result@asReference=true@mimetype=image/png

**Test 3: Accessing a remote tiff as WMS GetMap Request:**

http://localhost/cgi-bin/zoo_loader.cgi?request=Execute&service=WPS&version=1.0.0&Identifier=HelloPy
5.1.4 How To Use the Orfeo Toolbox support

**Authors** Nicolas Bozon, Gérald Fenoy, Jeff McKenna

**Last Updated** $Date: 2015-02-09 18:33:28 +0100 (lun. 09 févr. 2015) $

The key idea of the ZOO Orfeo Toolbox support implementation is to take advantage of the OTB Applications by using them directly as ZOO WPS Services, without any modifications.

Here is an overview of the installation procedure and the required configuration to activate the OTB support in ZOO-Project.

**Note:** These installation steps were successfully tested on Ubuntu 14.4 LTS

---

**Table of Contents**

- How To Use the Orfeo Toolbox support
  - How to make it working ?
    - Requirements
    - Installation steps
    - Configuration steps

---

**How to make it working ?**

**Requirements**

- latest ZOO-Kernel trunk version
- Orfeo Toolbox (OTB 4.2.1 )
- Insight Segmentation and Registration Toolkit (ITK-4.7 )

**Installation steps**

**Note:** For OTB and ITK, the CMAKE_C_FLAGS and CMAKE_CXX_FLAGS must first be set to -fPIC

Download lastest ZOO-Kernel code from SVN.

```
svn checkout http://svn.zoo-project.org/svn/trunk/zoo-kernel zoo-kernel
```

Then compile ZOO-Kernel using the needed configuration options as shown bellow:
cd zoo-kernel
autoconf
./configure --with-otb=/usr/local --with-itk=/usr/local --with-itk-version=4.7
make
cp zoo_loader.cgi /usr/lib/cgi-bin

**Configuration steps**

**Main configuration file**  Add the following content to your `/usr/lib/cgi-bin/main.cfg` file in the `[env]` section:

```
ITK_AUTOLOAD_PATH=/usr/local/lib/otb/applications
```

**Services configuration file**  The build of the `otb2zcfg` utility is required to activate the available OTB Applications as WPS services. This can be done using the following command:

```
mkdir build
cd build
ccmake ..
make
```

Run the following command to generate all the needed `zcfg` files for the available OTB Application:

```
mkdir zcfgs
cd zcfgs
export ITK_AUTOLOAD_PATH=/your/path/to/otb/applications
./build/otb2zcfg
mkdir /location/to/your/cgi-bin/OTB
cp *zcfg /location/to/your/cgi-bin/OTB
```

**Test the ZOO OTB support**  Once done, OTB Applications should be listed as available WPS Services when running a GetCapabilities request

```
http://localhost/cgi-bin/zoo_loader.cgi?request=GetCapabilities&service=WPS
```

Each OTB Service can then be described individually using the DescribeProcess request, as for example:

```
http://localhost/cgi-bin/zoo_loader.cgi?request=DescribeProcess&service=WPS&version=1.0.0&Identifier=OTB.BandMath
```

And executed according to your needs, as for the following example executing OTB.BandMath with the OTB sample data as input

```
http://localhost/cgi-bin/zoo_loader.cgi?request=Execute&service=WPS&version=1.0.0&Identifier=OTB.BandMath&DataInputs=il=Re ... _nir.tif;out=float;exp=im1b3*cos%28im1b1%29,im1b2*cos%28im1b1%29,im1b1*cos%28im1b1%29&RawDataOutput=out@mimeType=image/png
```

When executing OTB applications as WPS Services, it is also possible to check the OTB process status, using the usual ZOO GetStatus request.

**5.2 ZOO Services Documentation**

The following sections will assist you with ZOO Services:
5.2.1 Introduction

ZOO Services are example Web services which work with the ZOO Kernel. They are based on various existing Open Source libraries and tend to provide simple Web processing functions such as GIS format conversion, GIS file reprojection, basic spatial operations, basic raster operations...

The available ZOO Services are under development and come without any warranty. They are based on existing code and prove that the ZOO Kernel works with many different codes and languages (please have a look to the ZOO demos!). The ZOO Project team wants to encourage people to use the ZOO Services as a functional basis for Web processing, but above all to provide a source of inspiration to the community for creating new ZOO Services.

What is a ZOO Service?

A ZOO Service is a couple composed of:

- The code you want to turn into a standardized Web service
- A configuration file (.zcfg) which describes this Web service

Learn more on configuring ZOO services by reading the .zcfg Reference.

Example ZOO Services

See the ZOO Services examples page.

5.2.2 ZCFG : the ZOO Service Configuration File

Authors  Nicolas Bozon, Gérald Fenoy, Jeff McKenna

Last Updated  $Date: 2014-11-05 13:29:01 +0100 (mer. 05 nov. 2014) $

Table of Contents

- ZCFG : the ZOO Service Configuration File
  - Main Metadata Information
  - List of Inputs
  - List of Outputs
  - Type Of Data Nodes
    * LiteralData node
    * BoundingBoxData node
    * ComplexData node

The ZOO Service configuration file (.zcfg) describes the service and will be parsed by the ZOO Kernel. We will describe here what such a file contains. You can also take a look at the existing examples of ZCFG files in the cgi-env directory of each services available in the ZOO-Project SVN source tree.

A ZOO Configuration file is divided into three distinct sections:

1. Main Metadata information
2. List of Inputs metadata information (optional since rev. 469)
3. List of Outputs metadata information
Note: The ZOO Service Configuration File is case sensitive.

Main Metadata Information

The first part in a ZOO Configuration file contains the metadata information relative to the service. Note that the “name of your service” between brackets on the first line has to be the exact same name as the function you defined in your services provider code. In most cases, this name is also the name of the ZCFG file without the “.zcfg” extension.

You can see below a description of the main metadata information:

```
[Name of your service]
1 Title = Title of your service
2 Abstract = Description of your service
3 processVersion = Version number of your service
4 storeSupported = true/false
5 statusSupported = true/false
6 serviceType = the programming language used to implement the service (C/Fortran/Python/Java/PHP/JavaScript)
7 serviceProvider = name of your services provider (shared library/Python Module/Java Class/PHP Script/JavaScript script)
8 <MetaData>
9   title = Metadata title of your service
10 </MetaData>
```

List of Inputs

The list of inputs contains metadata information of each supported input, and they are grouped using a <DataInputs> node.

Each input is defined as:

- a name (between brackets as for the name of the service before)
- various metadata properties (Title, Abstract, minOccurs, maxOccurs and, in case of ComplexData, the optional maximumMegabytes)
- a Type Of Data node (description)

A typical list of inputs (<DataInputs>) look like the following:

```
<DataInputs>
1   [Name of the first input]
2     Title = Title of the first input
3     Abstract = Abstract describing the first input
4     minOccurs = Minimum occurrence of the first input
5     maxOccurs = Maximum occurrence of the first input
6     <Type Of Data Node />
7   [Name of the second input]
8     Title = Title of the second input
9     Abstract = Abstract describing the second input
10    minOccurs = Minimum occurrence of the second input
11    maxOccurs = Maximum occurrence of the second input
12   <Type Of Data Node />
13 </DataInputs>
```

Note: you can add <MetaData> node as in the main metadata information.
List of Outputs

The list of outputs is very similar to a list of inputs except it is specified as a `<DataOutputs>` node.

A typical `<DataOutputs>` node looks like the following:

```xml
<DataOutputs>
  [Name of the output]
  Title = Title of the output
  Abstract = Description of the output
  <Type Of Data Node />
</DataOutputs>
```

Type Of Data Nodes

In the beginning of this ZCFG introduction, we spoke about “Type Of Data Nodes” to describe the data type of inputs and outputs.

You can define your data as:

- `LiteralData`
- `BoundingBoxData`
- `ComplexData`

Except for `LiteralData`, each `Type Of Data` node must have at least one `<Default>` node. Even if empty, it has to be present. So, something like the following should be present in your ZCFG file:

```xml
<Default />
```

Otherwise, ZOO-Kernel won’t be able to parse your ZCFG correctly.

LiteralData node

A `<LiteralData>` node contains:

- one (optional) `AllowedValues` key containing all value allowed for this input
- one (optional) `range` properties containing the range ([, ])
- one (optional) `rangeMin` (`rangeMax`) properties containing the minimum (maximum) value of this range
- one (optional) `rangeSpacing` properties containing the regular distance or spacing between value in this range
- one (optional) `rangeClosure` properties containing the closure type (c, o, oc, co)
- one `<Default>` node,
- zero or more `<Supported>` nodes depending on the existence or the number of supported Units Of Measure (UOM), and
- a `dataType` property. The `dataType` property defines the type of literal data, such as a string, an integer and so on (consult the complete list of supported data types).

`<Default>` and `<Supported>` nodes can contain the `uom` property to define which UOM has to be used for this input value.

For input `<LiteralData>` nodes, you can add the `value` property to the `<Default>` node to define a default value for this input. This means that, when your Service will be run, even if the input wasn’t defined, this default value will be set as the current value for this input.
A typical `<LiteralData>` node, defining a `float` data type using meters or degrees for its UOM, looks like the following:

```
<LiteralData>
   dataType = float
   <Default>
      uom = meters
   </Default>
   <Supported>
      uom = feet
   </Supported>
</LiteralData>
```

A typical `<LiteralData>` node, defining a `float` data type which should take values contained in 
\([0.0,100.0]\), looks like the following:

```
<LiteralData>
   dataType = float
   rangeMin = 0.0
   rangeMax = 100.0
   rangeClosure = c
   <Default />
</LiteralData>
```

Or more simply:

```
<LiteralData>
   dataType = float
   range = [0.0,100.0]
   <Default />
</LiteralData>
```

A typical `<LiteralData>` node, defining a `string` data type which support values hillshade, slope, aspect, TRI, TPI and roughness, looks like the following:

```
<LiteralData>
   dataType = string
   AllowedValues = hillshade,slope,aspect,TRI,TPI,roughness
   <Default />
</LiteralData>
```

Properties `AllowedValues` and `range*` can be combined with both `<Default>` and `<Supported>` nodes in the same was as `<LiteralData>` node. For instance, the following is supported:

```
<LiteralData>
   dataType = int
   <Default>
      value = 11
      AllowedValues = -10,-8,-7,-5,-1
      rangeMin = 0
      rangeMin = 100
      rangeClosure = co
   </Default>
   <Supported>
      rangeMin = 200
      rangeMin = 600
      rangeClosure = co
   </Supported>
   <Supported>
      rangeMin = 750
```

BoundingBoxData node

A `<BoundingBoxData>` node contains:

- one `<Default>` node with a CRS property defining the default Coordinate Reference Systems (CRS), and
- one or more `<Supported>` nodes depending on the number of CRS your service supports (note that you can alternatively use a single `<Supported>` node with a comma-separated list of supported CRS).

A typical `<BoundingBoxData>` node, for two supported CRS (EPSG:4326 and EPSG:3785), looks like the following:

```xml
<BoundingBoxData>
  <Default>
    CRS = urn:ogc:def:crs:EPSG:6.6:4326
  </Default>
  <Supported>
    CRS = urn:ogc:def:crs:EPSG:6.6:4326
  </Supported>
  <Supported>
    CRS = urn:ogc:def:crs:EPSG:6.6:3785
  </Supported>
</BoundingBoxData>
```

ComplexData node

A ComplexData node contains:

- a `<Default>` node and
- one or more `<Supported>` nodes depending on the number of supported formats. A format is made up of this set of properties: `mimeType`, `encoding` and optionally `schema`.

For output ComplexData nodes, you can add the `extension` property to define what extension to use to name the file when storing the result is required. Obviously, you’ll have to add the `extension` property to each supported format (for the `<Default>` and `<Supported>` nodes).

You can also add the `asReference` property to the `<Default>` node to define if the output should be stored on server side per default.

**Note:** the client can always modify this behavior by setting `asReference` attribute to `true` or `false` for this output in the request `ResponseDocument` parameter.

You can see below a sample ComplexData node for default `application/json` and `text/xml` (encoded in UTF-8 or base64) mimeTypes support:

```xml
<ComplexData>
  <Default>
    mimeType = application/json
    encoding = UTF-8
  </Default>
</ComplexData>
```
5.2.3 How To Setup ZOO Services

Authors  Nicolas Bozon, Gérald Fenoy, Jeff McKenna, Luca Delucchi

Last Updated  $Date: 2015-02-25 13:32:55 +0100 (mer. 25 févr. 2015) $

ZOO Services are quite easy to create once you have installed the ZOO Kernel and have chosen code (in the language of your choice) to turn into a ZOO service. Here are some HelloWorlds in Python, PHP, Java and JavaScript with links to their corresponding .zcfg files.

Table of Contents

- How To Setup ZOO Services
  - Common informations
  - Python
    * Python ZCFG requirements
    * Python Data Structure used
    * Sample ZOO Python Services Provider
  - PHP
    * ZOO-API
    * PHP ZCFG requirements
    * PHP Data Structure used
    * Sample ZOO PHP Services Provider
  - Java
    * ZOO-API
    * Java ZCFG requirements
    * Java Data Structure used
    * Sample ZOO Java Services Provider
  - Javascript
    * ZOO API
    * Javascript ZCFG requirements
    * Javascript Data Structure used
    * Sample ZOO Javascript Services Provider

Common informations

The function of the process for each programming language take three arguments: the main configuration, inputs and outputs.
Note: The service has to **return 3 if the process run successfully instead it return 4** if the process end with an error.

### Python

You’ll find here information needed to deploy your own Python Services Provider.

### Python ZCFG requirements

Note: For each Service provided by your ZOO Python Services Provider, the ZCFG File must be named the same as the Python module function name (also the case of characters is important).

The ZCFG file should contain the following:

- **serviceType** Python
- **serviceProvider** The name of the Python module to use as a ZOO Service Provider. For instance, if your script, located in the same directory as your ZOO Kernel, was named `my_module.py` then you should use `my_module` (the Python module name) for the serviceProvider value in ZCFG file.

### Python Data Structure used

The three parameters of the function are passed to the Python module as dictionaries. Following you’ll find an example for each parameters

**Main configuration** Main configuration contains several informations, some of them are really useful to develop your service. Following an example

```python
{
    'main': {
        'lang': 'en-UK',
        'language': 'en-US',
        'encoding': 'utf-8',
        'dataPath': '/var/www/tmp',
        'tmpPath': '/var/www/tmp',
        'version': '1.0.0',
        'mapserverAddress': 'http://localhost/cgi-bin/mapserv',
        'isSoap': 'false',
        'tmpUrl': 'http://localhost/tmp/',
        'serverAddress': 'http://localhost/zoo'
    },
    'identification': {
        'keywords': 'WPS,GIS',
        'abstract': 'WPS services for testing ZOO',
        'fees': 'None',
        'accessConstraints': 'none',
        'title': 'testing services'
    },
    'lenv': {
        'status': '0',
        'soap': 'false',
        'cwd': '/usr/lib/cgi-bin',
        'sid': '24709'
    },
    'env': {
        'DISPLAY': 'localhost:0'
    },
    'provider': {
        'addressCountry': 'it',
        ...
    }
}
```
Inputs  The inputs are somethings like this

```
{  
  "variable_name": {  
    "minOccurs": "1",  
    "DataType": "string",  
    "value": "this_is_the_value",  
    "maxOccurs": "1",  
    "inRequest": "true"  
  }  
}
```

The access to the value you have to require for the value parameter, something like this

```
yourVariable = inputs[‘variable_name’][‘value’]
```

Outputs  The outputs data as a structure really similar to the inputs one

```
{  
  "result": {  
    "DataType": "string",  
    "inRequest": "true"  
  }  
}
```

There is no value parameter before you assign it

```
inputs[‘result’][‘value’] = yourOutputDataVariable
```

The return statement has to be an integer: corresponding to the service status code.

To add a message for the wrong result you can add the massage to conf[“lenv”][“message”], for example:

```
conf[“lenv”][“message”] = ’Your module return an error’
```

Sample ZOO Python Services Provider

The following code represents a simple ZOO Python Services Provider which provides only one Service, the HelloPy one.

```
import zoo
import sys

def HelloPy(conf, inputs, outputs):
```
PHP

ZOO-API

The ZOO-API for the PHP language is automatically available from your service code. The following functions are defined in the ZOO-API:

- `int zoo_SERVICE_SUCCEEDED()`: return the value of SERVICE_SUCCEEDED
- `int zoo_SERVICE_FAILED()`: return the value of SERVICE_FAILED
- `string zoo_Translate(string a)`: return the translated string (using the "zoo-service" textdomain)
- `void zoo_UpdateStatus(Array conf, string message, int pourcent)`: update the status of the running service

PHP ZCFG requirements

The ZCFG file should contain the following:

- `serviceType`: PHP
- `serviceProvider`: The name of the php script (i.e. service.php) to use as a ZOO Service Provider.

PHP Data Structure used

The three parameters are passed to the PHP function as Arrays.

Sample ZOO PHP Services Provider

```php
<?
function HelloPHP(&$main_conf,&$inputs,&$outputs){
    $tmp="Hello "+inputs["a"]+" from PHP World !";
    $outputs["Result"]="value"=zoo_Translate($tmp);
    return zoo_SERVICE_SUCCEEDED();
}
?>
```

Java

ZOO-API

Before you build your first ZOO-Service implemented in Java, it is recommended that you first build the ZOO class of the Java ZOO-API.

Note: You should build ZOO-Kernel prior to follow this instructions.

To build the ZOO.class of the ZOO-API for Java, use the following command:
cd zoo-api/java
make
cp ZOO.class libZOO.so /usr/lib/cgi-bin

**Note:** running the previous commands will require that both `javac` and `javah` are in your PATH.

### Java ZCFG requirements

**Note:** For each Service provided by your ZOO Java Services Provider (your corresponding Java class), the ZCFG File must be named the same as the Java public method name (also the case of characters is important).

The ZCFG file should contain the following:

- `serviceType`: Java
- `serviceProvider`: The name of the Java class to use as a ZOO Service Provider. For instance, if your java class, located in the same directory as your ZOO-Kernel, was named `HelloWorld.class` then you should use `HelloWorld`.

### Java Data Structure used

The three parameters are passed to the Java function as `java.util.HashMap`.

### Sample ZOO Java Services Provider

```
import java.util.*;

public class HelloJava {
    public static int HelloWorldJava(HashMap conf, HashMap inputs, HashMap outputs) {
        HashMap hml = new HashMap();
        hml.put("dataType","string");
        HashMap tmp=(HashMap)(inputs.get("S"));
        java.lang.String v=tmp.get("value").toString();
        hml.put("value","Hello "+v+" from JAVA WOrld !");
        outputs.put("Result",hml);
        System.err.println("Hello from JAVA WOrld !");
        return ZOO.SERVICE_SUCCEEDED;
    }
}
```

### Javascript

**ZOO API**

If you need to use **ZOO API** in your service, you have first to copy `zoo-api.js` and `zoo-proj4js.js` where your services are located (for example in Unix system probably in `/usr/lib/cgi-bin/`
Javascript ZCFG requirements

Note: For each Service provided by your ZOO Javascript Services Provider, the ZCFG File must be named the same as the Javascript function name (also the case of characters is important).

The ZCFG file should contain the following:

- **serviceType**: JS
- **serviceProvider**: The name of the JavaScript file to use as a ZOO Service Provider. For instance, if your script, located in the same directory as your ZOO Kernel, was named `my_module.js` then you should use `my_module.js`.

Javascript Data Structure used

The three parameters of the function are passed to the JavaScript function as Object.

Sample ZOO Javascript Services Provider

```javascript
function hellojs(conf, inputs, outputs) {
    outputs = new Array();
    outputs = {};
    outputs["result"]["value"] = "Hello " + inputs["S"]["value"] + " from JS World !";
    return Array(3, outputs);
}
```

5.2.4 How To Use ZOO Status Service

Authors Luca Delucchi

Last Updated $Date: 2015-02-09 16:26:31 +0100 (lun. 09 févr. 2015) $

ZOO Status Service is a utility of ZOO-Project to return the completion percent of your service.

Install ZOO Status Service

To install the ZOO Status Service you have to move in `/path/to/zoo/source/zoo-services/utils/status/` and compile the source running the make command. If no errors are returned during compilation you can copy the content of `cgi-env` to `/usr/lib/cgi-bin/` or where you have your `zoo_loader.cgi` working with this command (you need administration right):

```bash
cp /path/to/zoo/source/zoo-services/utils/status/cgi-env/*{zcfg,zo,py} /usr/lib/cgi-bin
```

With this command you copy the code to permit to ZOO Status Service and some example processes about how it works.

Now you have to add these two lines to main.cfg:

```ini
rewriteUrl=call
dataPath=/var/www/data
```

Here you define the path where the service is able to find the xsl file, specified in the dataPath parameter. You also tell the ZOO Kernel that you want to use the rewriteUrl.

The last operation is to copy the `updateStatus.xsl` to `dataPath` directory as follow:
5.2.5 How To Debug ZOO Services

Authors Luca Delucchi

Last Updated $Date: 2015-02-09 16:26:31 +0100 (lun. 09 févr. 2015) $

There are different ways to debug your services, the most used solutions are via web or via command line.

Web

Using the web request you can see any problem in the log file of Apache.

On Unix system the log file is usually in /var/log/apache2 and the relevant file is error_log. A simple way to read the file is to use the tail command, it permits to see the update of the file for each request

```
tail -f /var/log/apache2/error_log
```

If the log is not clear enough you can add some more debug information to your code. You have to write to standard error.

Python

Using Python, you can for example do this

```
import sys

#add this line when you want see an own message
sys.stderr.write("My message")
```

Javascript

You can use alert to print a string to standard error:

```
// add this line when you want see an own message
alert(’My message’)
// you can debug value of inputs, outputs or conf
alert(inputs[”S”][”value”])
```

Note: If you try to pass an object it will only return [object Object]

Command line

It is possible to use the ZOO kernel zoo_loader.cgi also from command line. This is really useful to debug in a deeper way your service:

```
# in order to use it you have to copy test_service.py and HelloPy.zcfg from
# the example services
./zoo_loader.cgi "service=wps&version=1.0.0&request=execute&identifier=HelloPy&datainputs=a=your name"
```
Working this way you can use the standard debug system of the actual programming language used to develop your service.

**GDB**

From command line you can use also the command line tool GDB to debug `zoo_loader.cgi`, you have to run:

```
# launch zoo_loader.cgi from gdb
gdb zoo_loader.cgi
# now run your request
run "service=wps&version=1.0.0&request=execute&identifier=HelloPy&datainputs=a=your name&responsedocument=Result"
```

At this point you can ask help at the ZOO mailing list copying the result of the command.

**Python**

For Python, you can use `pdb`, more info at [http://docs.python.org/2/library/pdb.html](http://docs.python.org/2/library/pdb.html)

```
import pdb

# add this line when you want investigate your code in more detail
pdb.set_trace()
```

**Javascript**

You can use `alert` also to print in the console, more info in the *Javascript* web section

### 5.2.6 Service Examples

**Authors** Nicolas Bozon, Gérald Fenoy, Jeff McKenna

**Last Updated** $Date: 2011-12-07 14:44:57 +0100 (Wed, 07 Dec 2011) $

ZOO Services are quite easy to create once you have installed the ZOO Kernel and have chosen code (in the language of your choice) to turn into a ZOO service. Here are some HelloWorlds with links to their corresponding .zcfg files.

#### Table of Contents

- Service Examples
  - GDAL ZOO Service
    - Implemented functions
  - OGR ZOO Service
    - Implemented functions

#### GDAL ZOO Service

The GDAL ZOO Service is based on GDAL source code and copyright. This ZOO Service aims to provide some basic raster processing operations to your ZOO Kernel installation. Learn more and read documentation on the official GDAL website.
Implemented functions

**Gdal_Grid** creates regular grid from the scattered data

**Gdal_Translate** converts raster data between different formats

**OGR ZOO Service**

The OGR ZOO Service is based on OGR source code and copyright. This ZOO Service aims to provide some basic vector spatial operations to your ZOO Kernel installation. Learn more and read documentation on the official OGR website.

Implemented functions

Base_Vect_Ops

### 5.3 ZOO API Documentation

The following sections will assist you with the ZOO API:

#### 5.3.1 Description

ZOO API is a Javascript library designed to make the WPS Process creation and chaining easier. It works on the server-side using the Mozilla foundation JavaScript engine, SpiderMonkey. It uses a Proj4js adaptation for server-side reprojection. It also allows to easily convert vector formats (such as GML, KML, GeoJSON, etc). The API lets you orchestrate WPS services simply and offers the ability to add logic and controls in the WPS chaining.

#### Download ZOO API

- zoo-api.js
- zoo-proj4js.js

**ZOO API MIT/X-11 License**

Copyright (c) 2011 3LIZ

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
5.3.2 Classes

Authors  Nicolas Bozon, Gérald Fenoy, Jeff McKenna

Last Updated  $Date: 2011-12-07 14:44:57 +0100 (Wed, 07 Dec 2011) $

The following classes are available in the ZOO API:

ZOO

The following constants and functions are available for the ZOO class:

Constants

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERVICE_ACCEPTED</td>
<td>{Integer} used for</td>
</tr>
<tr>
<td>SERVICE_STARTED</td>
<td>{Integer} used for</td>
</tr>
<tr>
<td>SERVICE_PAUSED</td>
<td>{Integer} used for</td>
</tr>
<tr>
<td>SERVICE_SUCCEEDED</td>
<td>{Integer} used for</td>
</tr>
<tr>
<td>SERVICE_FAILED</td>
<td>{Integer} used for</td>
</tr>
</tbody>
</table>

Functions

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>removeItem</td>
<td>Remove an object from an array.</td>
</tr>
<tr>
<td>indexOf</td>
<td>Copy all properties of a source object to a destination object.</td>
</tr>
<tr>
<td>extend</td>
<td>Given two objects representing points with geographic coordinates, this calculates the distance between those points on the surface of an ellipsoid.</td>
</tr>
<tr>
<td>rad</td>
<td>Method used to create ZOO classes.</td>
</tr>
<tr>
<td>distVincenty</td>
<td>Method used to update the status of the process</td>
</tr>
<tr>
<td>Class</td>
<td></td>
</tr>
<tr>
<td>UpdateStatus</td>
<td></td>
</tr>
</tbody>
</table>

Constants

SERVICE_ACCEPTED  {Integer} used for
SERVICE_STARTED  {Integer} used for
SERVICE_PAUSED  {Integer} used for
SERVICE_SUCCEEDED  {Integer} used for
SERVICE_FAILED  {Integer} used for

Functions
(removeItem)
removeItem: function(array, item)

Remove an object from an array. Iterates through the array to find the item, then removes it.

Parameters

array {Array}
item {Object}

Returns

{Array} A reference to the array

indexOf

indexOf: function(array, obj)

Parameters

array {Array}
obj {Object}

Returns

{Integer} The index at, which the first object was found in the array. If not found, returns -1.

extend

extend: function(destination, source)

Copy all properties of a source object to a destination object. Modifies the passed in destination object. Any properties on the source object that are set to undefined will not be (re)set on the destination object.

Parameters

destination {Object} The object that will be modified
source {Object} The object with properties to be set on the destination

Returns

{Object} The destination object.

rad

rad: function(x)

Parameters

x {Float}
Returns

{Float}

distVincenty

distVincenty: function(p1,p2)

Given two objects representing points with geographic coordinates, this calculates the distance between those points on the surface of an ellipsoid.

Parameters:

p1 [ZOO.Geometry.Point] (or any object with both .x, .y properties)
p2 [ZOO.Geometry.Point] (or any object with both .x, .y properties)

Class

Class: function()

Method used to create ZOO classes. Includes support for multiple inheritance.

UpdateStatus

UpdateStatus: function(env,value)

Method used to update the status of the process

Parameters

env {Object} The environment object
value {Float} The status value between 0 to 100

ZOO.Bounds

Instances of this class represent bounding boxes.

Properties

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>left</td>
<td>{Number} Minimum horizontal coordinate.</td>
</tr>
<tr>
<td>bottom</td>
<td>{Number} Minimum vertical coordinate.</td>
</tr>
<tr>
<td>right</td>
<td>{Number} Maximum horizontal coordinate.</td>
</tr>
<tr>
<td>top</td>
<td>{Number} Maximum vertical coordinate.</td>
</tr>
</tbody>
</table>
Functions

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZOO.Bounds</td>
<td>Construct a new bounds object.</td>
</tr>
<tr>
<td>clone</td>
<td>Create a cloned instance of this bounds.</td>
</tr>
<tr>
<td>equals</td>
<td>Test a two bounds for equivalence.</td>
</tr>
<tr>
<td>toString</td>
<td>{String} String representation of bounds object.</td>
</tr>
<tr>
<td>toBBOX</td>
<td>Create a new polygon geometry based on this bounds.</td>
</tr>
<tr>
<td>toGeometry</td>
<td>{Float} The width of the bounds</td>
</tr>
<tr>
<td>getWidth</td>
<td>{Float} The height of the bounds (top minus bottom)</td>
</tr>
<tr>
<td>getHeight</td>
<td></td>
</tr>
<tr>
<td>add</td>
<td></td>
</tr>
<tr>
<td>extend</td>
<td>Extend the bounds to include the point, lonlat, or bounds specified.</td>
</tr>
<tr>
<td>intersectsBounds</td>
<td>Determine whether the target bounds intersects this bounds.</td>
</tr>
<tr>
<td>containsBounds</td>
<td>Determine whether the target bounds is contained within this bounds.</td>
</tr>
</tbody>
</table>

Properties

left {Number} Minimum horizontal coordinate.

bottom {Number} Minimum vertical coordinate.

right {Number} Maximum horizontal coordinate.

top {Number} Maximum vertical coordinate.

Functions

ZOO.Bounds  Construct a new bounds object.

Parameters

left {Number} The left bounds of the box. Note that for width calculations, this is assumed to be less than the right value.

bottom {Number} The bottom bounds of the box. Note that for height calculations, this is assumed to be more than the top value.

right {Number} The right bounds.

top {Number} The top bounds.

close

close: function()  Create a cloned instance of this bounds.

Returns

/ZOO.Bounds/ A fresh copy of the bounds

equals

equals: function(bounds)  Test a two bounds for equivalence.

Parameters
bounds {ZOO.Bounds}

Returns

{Boolean} The passed-in bounds object has the same left, right, top, bottom components as this. Note that if bounds passed in is null, returns false.

toString

toString:function()

Returns

{String} String representation of bounds object. (ex. &lt;i&gt;"left-bottom=(5,42) right-top=(10,45)"&lt;/i&gt;)

toBBOX

toBBOX:function(decimal)

Parameters

decimal {Integer} How many significant digits in the bbox coords? Default is 6

Returns

{String} Simple String representation of bounds object. (ex. &lt;i&gt;"5.42,10.45"&lt;/i&gt;)

toGeometry

toGeometry: function()

Create a new polygon geometry based on this bounds.

Returns

{ZOO.Geometry.Polygon} A new polygon with the coordinates of this bounds.

getWidth

getWidth:function()

Returns

{Float} The width of the bounds

getHeight

getHeight:function()

Returns

{Float} The height of the bounds (top minus bottom).

add

add:function(x,y)

Parameters

x {Float}
y {Float}
Returns

\{ZOO.Bounds\} A new bounds whose coordinates are the same as this, but shifted by the passed-in x and y values.

extend

extend:function(object)

Extend the bounds to include the point, lonlat, or bounds specified. Note, this function assumes that left < right and bottom < top.

Parameters

object {Object} Can be Point, or Bounds.

intersectsBounds

intersectsBounds:function(bounds,inclusive)

Determine whether the target bounds intersects this bounds. Bounds are considered intersecting if any of their edges intersect or if one bounds contains the other.

Parameters

bounds \{ZOO.Bounds\} The target bounds.
inclusive {Boolean} Treat coincident borders as intersecting. Default is true. If false, bounds that do not overlap but only touch at the border will not be considered as intersecting.

Returns

{Boolean} The passed-in bounds object intersects this bounds.

containsBounds

containsBounds:function(bounds,partial,inclusive)

Determine whether the target bounds is contained within this bounds.

Parameters

bounds \{ZOO.Bounds\} The target bounds.
partial {Boolean} If any of the target corners is within this bounds consider the bounds contained. Default is false. If true, the entire target bounds must be contained within this bounds.
inclusive {Boolean} Treat shared edges as contained. Default is true.

Returns

{Boolean} The passed-in bounds object is contained within this bounds.

ZOO.Feature

Vector features use the ZOO.Geometry classes as geometry description.
Properties

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>fid</td>
<td>{String}</td>
</tr>
<tr>
<td>geometry</td>
<td>{ZOO.Geometry}</td>
</tr>
<tr>
<td>attributes</td>
<td>{Object} This object holds arbitrary properties that describe the feature.</td>
</tr>
<tr>
<td>bounds</td>
<td>{ZOO.Bounds} The box bounding that feature’s geometry, that property can be set by an ZOO.Format object when deserializing the feature, so in most cases it represents an information set by the server.</td>
</tr>
</tbody>
</table>

Functions

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZOO.Feature</td>
<td>Create a vector feature.</td>
</tr>
<tr>
<td>destroy</td>
<td>nullify references to prevent circular references and memory leaks</td>
</tr>
<tr>
<td>clone</td>
<td>Create a clone of this vector feature.</td>
</tr>
<tr>
<td>move</td>
<td>Moves the feature and redraws it at its new location</td>
</tr>
</tbody>
</table>

Properties

fid {String}

geometry {ZOO.Geometry}

attributes {Object} This object holds arbitrary properties that describe the feature.

bounds {ZOO.Bounds} The box bounding that feature’s geometry, that property can be set by an ZOO.Format object when deserializing the feature, so in most cases it represents an information set by the server.

Functions

ZOO.Feature Create a vector feature.

Parameters

geometry {ZOO.Geometry} The geometry that this feature represents.

attributes {Object} An optional object that will be mapped to the attributes property.

destroy

destroy: function()

nullify references to prevent circular references and memory leaks

clone

clone: function()

Create a clone of this vector feature. Does not set any non-standard properties.

Returns

{ZOO.Feature} An exact clone of this vector feature.

move Moves the feature and redraws it at its new location
ZOO.Format

Base class for format reading/writing a variety of formats.

Properties

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>{Object} A reference to options passed to the constructor.</td>
</tr>
<tr>
<td>externalProjection</td>
<td>{ZOO.Projection} When passed a externalProjection and internalProjection, the format will reproject the geometries it reads or writes.</td>
</tr>
<tr>
<td>internalProjection</td>
<td>{ZOO.Projection} When passed a externalProjection and internalProjection, the format will reproject the geometries it reads or writes.</td>
</tr>
<tr>
<td>data</td>
<td>{Object} When keepData is true, this is the parsed string sent to read.</td>
</tr>
<tr>
<td>keepData</td>
<td>{Object} Maintain a reference (data) to the most recently read data.</td>
</tr>
</tbody>
</table>

Functions

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZOO.Format</td>
<td>Instances of this class are not useful.</td>
</tr>
<tr>
<td>destroy</td>
<td>Clean up.</td>
</tr>
<tr>
<td>read</td>
<td>Read data from a string, and return an object whose type depends on the subclass.</td>
</tr>
<tr>
<td>data</td>
<td>{Object} When keepData is true, this is the parsed string sent to read.</td>
</tr>
<tr>
<td>write</td>
<td>Accept an object, and return a string.</td>
</tr>
</tbody>
</table>

Properties

options {Object} A reference to options passed to the constructor.

externalProjection {ZOO.Projection} When passed a externalProjection and internalProjection, the format will reproject the geometries it reads or writes. The externalProjection is the projection used by the content which is passed into read or which comes out of write. In order to reproject, a projection transformation function for the specified projections must be available. This support is provided via zoo-proj4js.

internalProjection {ZOO.Projection} When passed a externalProjection and internalProjection, the format will reproject the geometries it reads or writes. The internalProjection is the projection used by the geometries which are returned by read or which are passed into write. In order to reproject, a projection transformation function for the specified projections must be available. This support is provided via zoo-proj4js.

data {Object} When keepData is true, this is the parsed string sent to read.

keepData {Object} Maintain a reference (data) to the most recently read data. Default is false.

Functions

ZOO.Format Instances of this class are not useful. See one of the subclasses.

Parameters

options {Object} An optional object with properties to set on the format

Valid options

keepData {Boolean} If true, upon read, the data property will be set to the parsed object (e.g. the json or xml object).

Returns

An instance of ZOO.Format
destroy

    destroy: function()

    Clean up.

read

    read: function(data)

    Read data from a string, and return an object whose type depends on the subclass.

    Parameters
    
    data {string} Data to read-parse.

    Returns
    
    Depends on the subclass

write

    write: function(data)

    Accept an object, and return a string.

    Parameters
    
    object {Object} Object to be serialized

    Returns
    
    {String} A string representation of the object.

ZOO.Format.GeoJSON

Read and write GeoJSON.

Inherits from

  • ZOO.Format.JSON

Properties

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZOO.Format.GeoJSON</td>
<td>Create a new parser for GeoJSON. Deserialized a GeoJSON string.</td>
</tr>
<tr>
<td>read</td>
<td>Check if a GeoJSON object is a valid representative of the given type.</td>
</tr>
<tr>
<td>isValidType</td>
<td>Convert a feature object from GeoJSON into an ZOO.Feature.</td>
</tr>
<tr>
<td>parseFeature</td>
<td>Convert a geometry object from GeoJSON into an ZOO.Geometry.</td>
</tr>
<tr>
<td>parseGeometry</td>
<td></td>
</tr>
<tr>
<td>parseCoords</td>
<td>Object with properties corresponding to the GeoJSON geometry types.</td>
</tr>
</tbody>
</table>

parseCoords Properties

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>parseCoords</td>
<td>Object with properties corresponding to the GeoJSON geometry types.</td>
</tr>
</tbody>
</table>
### parseCoords Functions

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>parseCoords.point</td>
<td>Convert a coordinate array from GeoJSON into an ZOO.Geometry.Point.</td>
</tr>
<tr>
<td>parseCoords.multipoint</td>
<td>Convert a coordinate array from GeoJSON into an ZOO.Geometry.MultiPoint.</td>
</tr>
<tr>
<td>parseCoords.linestring</td>
<td>Convert a coordinate array from GeoJSON into an ZOO.Geometry.LineString.</td>
</tr>
<tr>
<td>parseCoords.multilinestring</td>
<td>Convert a coordinate array from GeoJSON into an ZOO.Geometry.MultiLineString.</td>
</tr>
<tr>
<td>parseCoords.polygon</td>
<td>Convert a coordinate array from GeoJSON into an ZOO.Geometry.Polygon.</td>
</tr>
<tr>
<td>parseCoords.multipolygon</td>
<td>Convert a coordinate array from GeoJSON into an ZOO.Geometry.MultiPolygon.</td>
</tr>
<tr>
<td>parseCoords.box</td>
<td>Convert a coordinate array from GeoJSON into an ZOO.Geometry.Polygon.</td>
</tr>
<tr>
<td>write</td>
<td>Serialize a feature, geometry, array of features into a GeoJSON string.</td>
</tr>
<tr>
<td>createCRSObject</td>
<td>Create the CRS object for an object.</td>
</tr>
</tbody>
</table>

### extract Properties

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>extract</td>
<td>Object with properties corresponding to the GeoJSON types.</td>
</tr>
</tbody>
</table>

### extract Functions

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>extract.feature</td>
<td>Return a partial GeoJSON object representing a single feature.</td>
</tr>
<tr>
<td>extract.geometry</td>
<td>Return a GeoJSON object representing a single geometry.</td>
</tr>
<tr>
<td>extract.point</td>
<td>Return an array of coordinates from a point.</td>
</tr>
<tr>
<td>extract.multipoint</td>
<td>Return an array of coordinates from a multipoint.</td>
</tr>
<tr>
<td>extract.linestring</td>
<td>Return an array of coordinate arrays from a linestring.</td>
</tr>
<tr>
<td>extract.multilinestring</td>
<td>Return an array of linestring arrays from a linestring.</td>
</tr>
<tr>
<td>extract.polygon</td>
<td>Return an array of linear ring arrays from a polygon.</td>
</tr>
<tr>
<td>extract.multipolygon</td>
<td>Return an array of polygon arrays from a multipolygon.</td>
</tr>
<tr>
<td>extract.collection</td>
<td>Return an array of geometries from a geometry collection.</td>
</tr>
</tbody>
</table>

### Functions

**ZOO.Format.GeoJSON** Create a new parser for GeoJSON.

**Parameters**

```javascript
options {Object} An optional object whose properties will be set on this instance.
```

**read**

```javascript
read: function(json,type,filter)
```

Deserialize a GeoJSON string.

**Parameters**

```javascript
json {String} A GeoJSON string
```

**type {String}** Optional string that determines the structure of the output. Supported values are “Geometry”, “Feature”, and “FeatureCollection”. If absent or null, a default of “FeatureCollection” is assumed.

```javascript
filter {Function} A function which will be called for every key and value at every level of the final result. Each value will be replaced by the result of the filter function. This can be used to reform generic objects into instances of classes, or to transform date strings into Date objects.
```
Returns

{Object} The return depends on the value of the type argument. If type is “FeatureCollection” (the default), the return will be an array of ZOO.Feature. If type is “Geometry”, the input json must represent a single geometry, and the return will be an ZOO.Geometry. If type is “Feature”, the input json must represent a single feature, and the return will be an ZOO.Feature.

isValidType

isValidType: function(obj,type)

Check if a GeoJSON object is a valid representative of the given type.

Returns

{Boolean} The object is valid GeoJSON object of the given type.

parseFeature

parseFeature: function(obj)

Convert a feature object from GeoJSON into a ZOO.Feature.

Parameters

obj {Object} An object created from a GeoJSON object

Returns

{ZOO.Feature} A feature.

parseGeometry

parseGeometry: function(obj)

Convert a geometry object from GeoJSON into a ZOO.Geometry.

Parameters

obj {Object} An object created from a GeoJSON object

Returns

{ZOO.Geometry} A geometry.

parseCoords Properties

parseCoords Object with properties corresponding to the GeoJSON geometry types. Property values are functions that do the actual parsing.

parseCoords Functions

parseCoords.point Convert a coordinate array from GeoJSON into a ZOO.Geometry.Point.

Parameters

array {Object} The coordinates array from the GeoJSON fragment.

Returns

{ZOO.Geometry.Point} A geometry.

parseCoords.multipoint Convert a coordinate array from GeoJSON into a ZOO.Geometry.MultiPoint.

Parameters

array {Object} The coordinates array from the GeoJSON fragment.

Returns


parseCoords.linestring  Convert a coordinate array from GeoJSON into a `ZOO.Geometry.LineString`.

Parameters

array {Object} The coordinates array from the GeoJSON fragment.

Returns

`{ZOO.Geometry.LineString}` A geometry.

parseCoords.multilinestring  Convert a coordinate array from GeoJSON into a `ZOO.Geometry.MultiLineString`.

Parameters

array {Object} The coordinates array from the GeoJSON fragment.

Returns

`{ZOO.Geometry.MultiLineString}` A geometry.

parseCoords.polygon  Convert a coordinate array from GeoJSON into a `ZOO.Geometry.Polygon`.

Parameters

array {Object} The coordinates array from the GeoJSON fragment.

Returns

`{ZOO.Geometry.Polygon}` A geometry.

parseCoords.multipolygon  Convert a coordinate array from GeoJSON into a `ZOO.Geometry.MultiPolygon`.

Parameters

array {Object} The coordinates array from the GeoJSON fragment.

Returns

`{ZOO.Geometry.MultiPolygon}` A geometry.

parseCoords.box  Convert a coordinate array from GeoJSON into a `ZOO.Geometry.Polygon`.

Parameters

array {Object} The coordinates array from the GeoJSON fragment.

Returns

`{ZOO.Geometry.Polygon}` A geometry.

write

write: function(obj, pretty)

Serialize a feature, geometry, array of features into a GeoJSON string.

Parameters

obj {Object} A `ZOO.Feature`, `ZOO.Geometry`, or an array of features.

pretty {Boolean} Structure the output with newlines and indentation. Default is false.

Returns

{String} The GeoJSON string representation of the input geometry, features, or array of features.
createCRSObject

createCRSObject: function(object)

Create the CRS object for an object.

Parameters

object {ZOO.Feature}

Returns

{Object} An object which can be assigned to the crs property of a GeoJSON object.

extract Properties

extract Object with properties corresponding to the GeoJSON types. Property values are functions that do the actual value extraction.

extract Functions

extract.feature Return a partial GeoJSON object representing a single feature.

Parameters

feature {ZOO.Feature}

Returns

{Object} An object representing the point.

extract.geometry Return a GeoJSON object representing a single geometry.

Parameters

geometry {ZOO.Geometry}

Returns

{Object} An object representing the geometry.

extract.point Return an array of coordinates from a point.

Parameters

point {ZOO.Geometry.Point}

Returns

{Array} An array of coordinates representing the point.

extract.multipoint Return an array of coordinates from a multipoint.

Parameters

multipoint {ZOO.Geometry.MultiPoint}

Returns

{Array} An array of point coordinate arrays representing the multipoint.

extract.linestring Return an array of coordinate arrays from a linestring.

Parameters

linestring {ZOO.Geometry.LineString}

Returns

{Array} An array of coordinate arrays representing the linestring.
extract.multilinestring  Return an array of linestring arrays from a linestring.

Parameters
multilinestring [ZOO.Geometry.MultiLineString]

Returns
{Array} An array of linestring arrays representing the multilinestring.

extract.polygon  Return an array of linear ring arrays from a polygon.

Parameters
polygon [ZOO.Geometry.Polygon]

Returns
{Array} An array of linear ring arrays representing the polygon.

extract.multipolygon  Return an array of polygon arrays from a multipolygon.

Parameters
multipolygon [ZOO.Geometry.MultiPolygon]

Returns
{Array} An array of polygon arrays representing the multipolygon

extract.collection  Return an array of geometries from a geometry collection.

Parameters
collection [ZOO.Geometry.Collection]

Returns
{Array} An array of geometry objects representing the geometry collection.

ZOO.Format.GML

Read/Write GML.

Inherits from
• ZOO.Format

Properties and Functions

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>schemaLocation</td>
<td>{String} Schema location for a particular minor version.</td>
</tr>
<tr>
<td>namespaces</td>
<td>{Object} Mapping of namespace aliases to namespace URIs.</td>
</tr>
<tr>
<td>defaultPrefix</td>
<td></td>
</tr>
<tr>
<td>collectionName</td>
<td>{String} Name of featureCollection element.</td>
</tr>
<tr>
<td>featureName</td>
<td>{String} Element name for features.</td>
</tr>
<tr>
<td>geometryName</td>
<td>{String} Name of geometry element.</td>
</tr>
<tr>
<td>xy</td>
<td>{Boolean} Order of the GML coordinate true:(x,y) or false:(y,x) Changing is not recommended, a new parser is needed.</td>
</tr>
<tr>
<td>extractAttributes</td>
<td>{Boolean} Could we extract attributes</td>
</tr>
<tr>
<td>ZOO.Format.GML</td>
<td>Create a new parser for GML.</td>
</tr>
<tr>
<td>read</td>
<td>Read data from a string, and return a list of features.</td>
</tr>
</tbody>
</table>
Table 5.1 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parseFeature</td>
<td>This function is the core of the GML parsing code in ZOO. Properties of this object are the functions that parse geometries based on their type.</td>
</tr>
<tr>
<td>parseGeometry</td>
<td>Properties of this object are the functions that parse geometries based on their type.</td>
</tr>
<tr>
<td>parseGeometry.point</td>
<td>Given a GML node representing a point geometry, create a ZOO point geometry.</td>
</tr>
<tr>
<td>parseGeometry.multipoint</td>
<td>Given a GML node representing a point geometry, create a ZOO multipoint geometry.</td>
</tr>
<tr>
<td>parseGeometry.linestring</td>
<td>Given a GML node representing a linestring geometry, create a ZOO linestring geometry.</td>
</tr>
<tr>
<td>parseGeometry.polygon</td>
<td>Given a GML node representing a polygon geometry, create a ZOO polygon geometry.</td>
</tr>
<tr>
<td>parseGeometry.multigeometry</td>
<td>Given a GML node representing a multigeometry, create a ZOO geometry collection.</td>
</tr>
<tr>
<td>parseAttributes</td>
<td>Parse ExtendedData from GML.</td>
</tr>
<tr>
<td>write</td>
<td>Accept Feature Collection, and return a string.</td>
</tr>
<tr>
<td>createFeature</td>
<td>Accept an ZOO.Feature, and build a GML node for it.</td>
</tr>
<tr>
<td>buildGeometryNode</td>
<td>Builds and returns a GML geometry node with the given geometry.</td>
</tr>
<tr>
<td>buildGeometry</td>
<td>Object containing methods to do the actual geometry node building based on geometry type.</td>
</tr>
<tr>
<td>buildGeometry.point</td>
<td>Given a ZOO point geometry, create a GML point.</td>
</tr>
<tr>
<td>buildGeometry.multipoint</td>
<td>Given a ZOO multipoint geometry, create a GML GeometryCollection.</td>
</tr>
<tr>
<td>buildGeometry.linestring</td>
<td>Given a ZOO linestring geometry, create a GML linestring.</td>
</tr>
<tr>
<td>buildGeometry.multilinestring</td>
<td>Given a ZOO multilinestring geometry, create a GML GeometryCollection.</td>
</tr>
<tr>
<td>buildGeometry.linearring</td>
<td>Given a ZOO linearring geometry, create a GML lineerring.</td>
</tr>
<tr>
<td>buildGeometry.polygon</td>
<td>Given a ZOO polygon geometry, create a GML polygon.</td>
</tr>
<tr>
<td>buildGeometry.multipolygon</td>
<td>Given a ZOO multipolygon geometry, create a GML GeometryCollection.</td>
</tr>
<tr>
<td>buildGeometry.collection</td>
<td>Given a ZOO geometry collection, create a GML MultiGeometry.</td>
</tr>
<tr>
<td>buildCoordinatesNode</td>
<td>builds the coordinates XmlNode</td>
</tr>
</tbody>
</table>

**ZOO.Format.GML**  Create a new parser for GML.

**Parameters**

- **options** {Object} An optional object whose properties will be set on this instance.

**read**

- **read** : function(data) Read data from a string, and return a list of features.

**Parameters**

- **data** {String} data to read/parse.

**Returns**

- {Array(ZOO.Feature)} An array of features.
parseFeature: function(node)

This function is the core of the GML parsing code in ZOO. It creates the geometries that are then attached to the returned feature, and calls parseAttributes() to get attribute data out.

**Parameters**
node {E4XElement}

**Returns**
{ZOO.Feature} A vector feature.

**parseGeometry** Properties of this object are the functions that parse geometries based on their type.

**parseGeometry.point** Given a GML node representing a point geometry, create a ZOO point geometry.

**Parameters**
node {E4XElement} A GML node.

**Returns**
{ZOO.Geometry.Point} A point geometry.

**parseGeometry.multipoint** Given a GML node representing a multipoint geometry, create a ZOO multipoint geometry.

**Parameters**
node {E4XElement} A GML node.

**Returns**
{ZOO.Geometry.MultiPoint} A multipoint geometry.

**parseGeometry.linestring** Given a GML node representing a linestring geometry, create a ZOO linestring geometry.

**Parameters**
node {E4XElement} A GML node.

**Returns**
{ZOO.Geometry.LineString} A linestring geometry.

**parseGeometry.multilinestring** Given a GML node representing a multilinestring geometry, create a ZOO multilinestring geometry.

**Parameters**
node {E4XElement} A GML node.

**Returns**
{ZOO.Geometry.MultiLineString} A multilinestring geometry.

**parseGeometry.polygon** Given a GML node representing a polygon geometry, create a ZOO polygon geometry.

**Parameters**
node {E4XElement} A GML node.

**Returns**
{ZOO.Geometry.Polygon} A polygon geometry.
**parseGeometry.multipolygon**  Given a GML node representing a multipolygon geometry, create a ZOO multipolygon geometry.

*Parameters*

- node {E4XElement} A GML node.

*Returns*

- {ZOO.Geometry.MultiPolygon} A multipolygon geometry.

**parseGeometry.envelope**  Given a GML node representing an envelope, create a ZOO polygon geometry.

*Parameters*

- node {E4XElement} A GML node.

*Returns*

- {ZOO.Geometry.Polygon} A polygon geometry.

**parseAttributes**

```javascript
parseAttributes: function(node)
```

*Parameters*

- node {E4XElement}

*Returns*

- {Object} An attributes object.

**write**

```javascript
write: function(features)
```

Generate a GML document string given a list of features.

*Parameters*

- features {Array(ZOO.Feature)} List of features to serialize into a string.

*Returns*

- {String} A string representing the GML document.

**createFeature**

```javascript
createFeature: function(feature)
```

Accept an ZOO.Feature, and build a GML node for it.

*Parameters*

- feature {ZOO.Feature} The feature to be built as GML.

*Returns*

- {E4XElement} A node reprensting the feature in GML.

**buildGeometryNode**

```javascript
buildGeometryNode: function(geometry)
```

*Parameters*

- geometry {ZOO.Geometry} The geometry to be built as GML.
Returns

{E4XElement} A node representing the geometry in GML.

buildGeometry Object containing methods to do the actual geometry node building based on geometry type.

buildGeometry.point Given a ZOO point geometry, create a GML point.

Parameters


Returns

{E4XElement} A GML point node.

buildGeometry.multipoint Given a ZOO multipoint geometry, create a GML multipoint.

Parameters


Returns

{E4XElement} A GML multipoint node.

buildGeometry.linestring Given a ZOO linestring geometry, create a GML linestring.

Parameters


Returns

{E4XElement} A GML linestring node.

buildGeometry.multilinestring Given a ZOO multilinestring geometry, create a GML multilinestring.

Parameters


Returns

{E4XElement} A GML multilinestring node.

buildGeometry.linearring Given a ZOO linearring geometry, create a GML linearring.

Parameters


Returns

{E4XElement} A GML linearring node.

buildGeometry.polygon Given a ZOO polygon geometry, create a GML polygon.

Parameters


Returns

{E4XElement} A GML polygon node.

buildGeometry.multipolygon Given a ZOO multipolygon geometry, create a GML multipolygon.

Parameters


5.3. ZOO API Documentation
Returns

{E4XElement} A GML multipolygon node.

buildCoordinatesNode

buildCoordinatesNode: function(geometry)

builds the coordinates XmlNode

<gml:coordinates decimal="." cs="," ts=" ">...</gml:coordinates>

Parameters

gameometry {ZOO.Geometry}

Returns

{E4XElement} created E4XElement

ZOO.Format.JSON

A parser to read/write JSON safely.

Inherits from

• ZOO.Format

Properties

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>indent</td>
<td>{String} For “pretty” printing, the indent string will be used once for each indentation level.</td>
</tr>
<tr>
<td>space</td>
<td>{String} For “pretty” printing, the space string will be used after the “:” separating a name/value pair.</td>
</tr>
<tr>
<td>newline</td>
<td>{String} For “pretty” printing, the newline string will be used at the end of each name/value pair or array item.</td>
</tr>
<tr>
<td>level</td>
<td>{Integer} For “pretty” printing, this is incremented/decremented during serialization.</td>
</tr>
<tr>
<td>pretty</td>
<td>{Boolean} Serialize with extra whitespace for structure.</td>
</tr>
</tbody>
</table>

Functions

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZOO.Format.JSON</td>
<td>Create a new parser for JSON.</td>
</tr>
<tr>
<td>read</td>
<td>Deserialize a json string.</td>
</tr>
<tr>
<td>write</td>
<td>Serialize an object into a JSON string.</td>
</tr>
<tr>
<td>writeIndent</td>
<td>Output an indentation string depending on the indentation level.</td>
</tr>
<tr>
<td>writeNewline</td>
<td>Output a string representing a newline if in pretty printing mode.</td>
</tr>
<tr>
<td>writeSpace</td>
<td>Output a string representing a space if in pretty printing mode.</td>
</tr>
</tbody>
</table>

Serialize Properties

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>serialize</td>
<td>Object with properties corresponding to the serializable data types.</td>
</tr>
</tbody>
</table>
### Serialize Functions

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>serialize.object</td>
<td>Transform an object into a JSON string.</td>
</tr>
<tr>
<td>serialize.array</td>
<td>Transform an array into a JSON string.</td>
</tr>
<tr>
<td>serialize.string</td>
<td>Transform a string into a JSON string.</td>
</tr>
<tr>
<td>serialize.number</td>
<td>Transform a number into a JSON string.</td>
</tr>
<tr>
<td>serialize.boolean</td>
<td>Transform a boolean into a JSON string.</td>
</tr>
<tr>
<td>serialize.date</td>
<td>Transform a date into a JSON string.</td>
</tr>
</tbody>
</table>

#### Properties

**indent**  
{String} For “pretty” printing, the indent string will be used once for each indentation level.

**space**  
{String} For “pretty” printing, the space string will be used after the “;” separating a name/value pair.

**newline**  
{String} For “pretty” printing, the newline string will be used at the end of each name/value pair or array item.

**level**  
{Integer} For “pretty” printing, this is incremented/decremented during serialization.

**pretty**  
{Boolean} Serialize with extra whitespace for structure. This is set by the `write` method.

#### Functions

**ZOO.Format.JSON**  
Create a new parser for JSON.

**Parameters**

- `options` {Object} An optional object whose properties will be set on this instance.

**read**

```javascript
read: function(json, filter)
```

Deserialize a json string.

**Parameters**

- `json` {String} A JSON string
- `filter` {Function} A function which will be called for every key and value at every level of the final result. Each value will be replaced by the result of the filter function. This can be used to reform generic objects into instances of classes, or to transform date strings into Date objects.

**Returns**

{Object} An object, array, string, or number.

**write**

```javascript
write: function(value, pretty)
```

Serialize an object into a JSON string.

**Parameters**

- `value` {String} The object, array, string, number, boolean or date to be serialized.
- `pretty` {Boolean} Structure the output with newlines and indentation. Default is false.
Returns

{String} The JSON string representation of the input value.

writeIndent

writeIndent: function()

Output an indentation string depending on the indentation level.

Returns

{String} An appropriate indentation string.

writeNewline

writeNewline: function()

Output a string representing a newline if in pretty printing mode.

Returns

{String} A string representing a new line.

writeSpace

writeSpace: function()

Output a string representing a space if in pretty printing mode.

Returns

{String} A space.

Serialize Properties

serialize Object with properties corresponding to the serializable data types. Property values are functions that do the actual serializing.

Serialize Functions

serialize.object Transform an object into a JSON string.

Parameters

object {Object} The object to be serialized.

Returns

{String} A JSON string representing the object.

serialize.array Transform an array into a JSON string.

Parameters

array {Array} The array to be serialized

Returns

{String} A JSON string representing the array.

serialize.string Transform a string into a JSON string.

Parameters

string {String} The string to be serialized

Returns

{String} A JSON string representing the string.
serialize.number  Transform a number into a JSON string.

Parameters
number {Number} The number to be serialized.

Returns
{String} A JSON string representing the number.

serialize.boolean  Transform a boolean into a JSON string.

Parameters
bool {Boolean} The boolean to be serialized.

Returns
{String} A JSON string representing the boolean.

serialize.date  Transform a date into a JSON string.

Parameters
date {Date} The date to be serialized.

Returns
{String} A JSON string representing the date.

ZOO.Format.KML
Read/Write KML.
Inherits from
• ZOO.Format
Properties and Functions

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>kmns</td>
<td>{String} KML Namespace to use. Defaults to 2.2 namespace.</td>
</tr>
<tr>
<td>foldersName</td>
<td>{String} Name of the folders. Default is “ZOO export”. If set to null, no name element will be created.</td>
</tr>
<tr>
<td>foldersDesc</td>
<td>{String} Description of the folders. Default is “Exported on [date].” If set to null, no description element will be created.</td>
</tr>
<tr>
<td>placemarksDesc</td>
<td>{String} Name of the placemarks. Default is “No description available”.</td>
</tr>
<tr>
<td>extractAttributes</td>
<td>{Boolean} Extract attributes from KML. Default is true. Extracting styleUrls requires this to be set to true</td>
</tr>
<tr>
<td>ZOO.Format.KML</td>
<td>Create a new parser for KML.</td>
</tr>
<tr>
<td>parseFeatures</td>
<td>Loop through all Placemark nodes and parse them.</td>
</tr>
<tr>
<td>parseFeature</td>
<td>This function is the core of the KML parsing code in ZOO.</td>
</tr>
<tr>
<td>parseGeometry</td>
<td>Properties of this object are the functions that parse geometries based on their type.</td>
</tr>
<tr>
<td>parseGeometry.point</td>
<td>Given a KML node representing a point geometry, create a ZOO point geometry.</td>
</tr>
<tr>
<td>parseGeometry.linestring</td>
<td>Given a KML node representing a linestring geometry, create a ZOO linestring geometry.</td>
</tr>
<tr>
<td>parseGeometry.polygon</td>
<td>Given a KML node representing a polygon geometry, create a ZOO polygon geometry.</td>
</tr>
<tr>
<td>parseGeometry.multigeometry</td>
<td>Given a KML node representing a multigeometry, create a ZOO geometry collection.</td>
</tr>
<tr>
<td>parseAttributes</td>
<td>Parse ExtendedData from KML.</td>
</tr>
<tr>
<td>parseExtendedData</td>
<td>Accept Feature Collection, and return a string.</td>
</tr>
<tr>
<td>write</td>
<td>Creates and returns a KML placemark node representing the given feature.</td>
</tr>
<tr>
<td>createPlacemark</td>
<td>Builds and returns a KML geometry node with the given geometry.</td>
</tr>
<tr>
<td>buildGeometryNode</td>
<td>Object containing methods to do the actual geometry node building based on geometry type.</td>
</tr>
<tr>
<td>buildGeometry</td>
<td>Given a ZOO point geometry, create a KML point.</td>
</tr>
<tr>
<td>buildGeometry.point</td>
<td>Given a ZOO multipoint geometry, create a KML GeometryCollection.</td>
</tr>
<tr>
<td>buildGeometry.multipoint</td>
<td>Given a ZOO linestring geometry, create a KML linestring.</td>
</tr>
<tr>
<td>buildGeometry.linestring</td>
<td>Given a ZOO multilinestring geometry, create a KML GeometryCollection.</td>
</tr>
<tr>
<td>buildGeometry.linearring</td>
<td>Given a ZOO linearring geometry, create a KML linearring.</td>
</tr>
<tr>
<td>buildGeometry.polygon</td>
<td>Given a ZOO polygon geometry, create a KML polygon.</td>
</tr>
<tr>
<td>buildGeometry.multipolygon</td>
<td>Given a ZOO multipolygon geometry, create a KML GeometryCollection.</td>
</tr>
<tr>
<td>buildGeometry.collection</td>
<td>Given a ZOO geometry collection, create a KML MultiPolygon.</td>
</tr>
<tr>
<td>buildCoordinatesNode</td>
<td>Builds and returns the KML coordinates node with the given geometry</td>
</tr>
</tbody>
</table>

**Parameters**

- **kmns**: {String} KML Namespace to use. Defaults to 2.2 namespace.
- **foldersName**: {String} Name of the folders. Default is “ZOO export”. If set to null, no name element will be created.
- **foldersDesc**: {String} Description of the folders. Default is “Exported on [date].” If set to null, no description element will be created.
- **placemarksDesc**: {String} Name of the placemarks. Default is “No description available”.
- **extractAttributes**: {Boolean} Extract attributes from KML. Default is true. Extracting styleUrls requires this to be set to true.

**ZOO.Format.KML**: Create a new parser for KML.
options {Object} An optional object whose properties will be set on this instance.

parseFeatures
parseFeatures: function(nodes)

Loop through all Placemark nodes and parse them. Will create a list of features

Parameters

nodes {Array} of {E4XElement} data to read/parse.
options {Object} Hash of options

parseFeature
parseFeature: function(node)

This function is the core of the KML parsing code in ZOO. It creates the geometries that are then attached to
the returned feature, and calls parseAttributes() to get attribute data out.

Parameters

node {E4XElement} A KML Point node.

Returns

{ZOO.Feature} A vector feature.

parseGeometry Properties of this object are the functions that parse geometries based on their type.

parseGeometry.point Given a KML node representing a point geometry, create a ZOO point geometry.

Parameters

node {E4XElement} A KML Point node.

Returns

{ZOO.Geometry.Point} A point geometry.

parseGeometry.linestring Given a KML node representing a linestring geometry, create a ZOO linestring geometry.

Parameters

node {E4XElement} A KML LineString node.

Returns

{ZOO.Geometry.LineString} A linestring geometry.

parseGeometry.polygon Given a KML node representing a polygon geometry, create a ZOO polygon geometry.

Parameters

node {E4XElement} A KML Polygon node.

Returns

{ZOO.Geometry.Polygon} A polygon geometry.

parseGeometry.multigeometry Given a KML node representing a multigeometry, create a ZOO geometry collection.

Parameters
node {E4XElement} A KML MultiGeometry node.

Returns

{ZOO.Geometry.Collection} A geometry collection.

parseAttributes

parseAttributes: function(node)

Parameters

node {E4XElement}

Returns

{Object} An attributes object.

parseExtendedData

parseExtendedData: function(node)

Parse ExtendedData from KML. Limited support for schemas/datatypes. See http://code.google.com/apis/kml/documentation/kmlreference.html#extendeddata for more information on extendedData.

Parameters

node {E4XElement}

Returns

{Object} An attributes object.

write

write: function(features)

Accept Feature Collection, and return a string.

Parameters

features {Array(ZOO.Feature)} An array of features.

Returns

{String} A KML string.

createPlacemark

createPlacemark: function(feature)

Creates and returns a KML placemark node representing the given feature.

Parameters

feature {ZOO.Feature}

Returns

{E4XElement}

buildGeometryNode

buildGeometryNode: function(geometry)
Builds and returns a KML geometry node with the given geometry.

**Parameters**

geometry [ZOO.Geometry]

**Returns**

{E4XElement}

**buildGeometry** Object containing methods to do the actual geometry node building based on geometry type.

**buildGeometry.point** Given a ZOO point geometry, create a KML point.

**Parameters**


**Returns**

{E4XElement} A KML point node.

**buildGeometry.multipoint** Given a ZOO multipoint geometry, create a KML GeometryCollection.

**Parameters**


**Returns**

{E4XElement} A KML GeometryCollection node.

**buildGeometry.linestring** Given a ZOO linestring geometry, create a KML linestring.

**Parameters**


**Returns**

{E4XElement} A KML linestring node.

**buildGeometry.multilinestring** Given a ZOO multilinestring geometry, create a KML GeometryCollection.

**Parameters**


**Returns**

{E4XElement} A KML GeometryCollection node.

**buildGeometry.linearring** Given a ZOO linearring geometry, create a KML linearring.

**Parameters**


**Returns**

{E4XElement} A KML linearring node.

**buildGeometry.polygon** Given a ZOO polygon geometry, create a KML polygon.

**Parameters**


**Returns**

{E4XElement} A KML polygon node.
buildGeometry.multipolygon  Given a ZOO multipolygon geometry, create a KML GeometryCollection.

Parameters


Returns

{E4XElement} A KML GeometryCollection node.

buildGeometry.collection  Given a ZOO geometry collection, create a KML MultiGeometry.

Parameters


Returns

{E4XElement} A KML MultiGeometry node.

buildCoordinatesNode

buildCoordinatesNode: function(geometry)

Builds and returns the KML coordinates node with the given geometry <coordinates>...</coordinates>

Parameters

geometry [ZOO.Geometry]

Return

{E4XElement}

ZOO.Format.WKT

Class for reading and writing Well-Known Text.

Inherits from

• ZOO.Format

Functions and Properties

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZOO.Format.WKT</td>
<td>Create a new parser for WKT</td>
</tr>
<tr>
<td>read</td>
<td>Deserialize a WKT string and return a vector feature or an array of vector features.</td>
</tr>
<tr>
<td>write</td>
<td>Serialize a feature or array of features into a WKT string.</td>
</tr>
<tr>
<td>extract</td>
<td>Object with properties corresponding to the geometry types.</td>
</tr>
<tr>
<td>parse</td>
<td>Object with properties corresponding to the geometry types.</td>
</tr>
<tr>
<td>parse.point</td>
<td>Return point feature given a point WKT fragment.</td>
</tr>
<tr>
<td>parse.multipoint</td>
<td>Return a multipoint feature given a multipoint WKT fragment.</td>
</tr>
<tr>
<td>parse.linestring</td>
<td>Return a linestring feature given a linestring WKT fragment.</td>
</tr>
<tr>
<td>parse.multilinestring</td>
<td>Return a multilinestring feature given a multilinestring WKT fragment.</td>
</tr>
<tr>
<td>parse.polygon</td>
<td>Return a polygon feature given a polygon WKT fragment.</td>
</tr>
<tr>
<td>parse.multipolygon</td>
<td>Return a multipolygon feature given a multipolygon WKT fragment.</td>
</tr>
<tr>
<td>parse.geometrycollection</td>
<td>Return an array of features given a geometrycollection WKT fragment.</td>
</tr>
</tbody>
</table>

ZOO.Format.WKT  Create a new parser for WKT

Parameters
options {Object} An optional object whose properties will be set on this instance

Returns


read

read: function(wkt)

Deserialize a WKT string and return a vector feature or an array of vector features. Supports WKT for POINT, MULTIPOINT, LINESTRING, MULTILINESTRING, POLYGON, MULTIPOLYGON, and GEOMETRYCOLLECTION.

Parameters

wkt {String} A WKT string

Returns

{<ZOO.Feature.Vector>|Array} A feature or array of features for GEOMETRYCOLLECTION WKT.

write

write: function(features)

Serialize a feature or array of features into a WKT string.

Parameters

features {<ZOO.Feature.Vector>|Array} A feature or array of features

Returns

{String} The WKT string representation of the input geometries

extract Object with properties corresponding to the geometry types. Property values are functions that do the actual data extraction.

parse Object with properties corresponding to the geometry types. Property values are functions that do the actual parsing.

parse.point Return point feature given a point WKT fragment.

Parameters

str {String} A WKT fragment representing the point

Returns

{ZOO.Feature} A point feature

parse.multipoint Return a multipoint feature given a multipoint WKT fragment.

Parameters

str {String} A WKT fragment representing the multipoint

Returns

{ZOO.Feature} A multipoint feature

parse.linestring Return a linestring feature given a linestring WKT fragment.

Parameters

str {String} A WKT fragment representing the linestring

Returns
A linestring feature

parse.multilinestring Return a multilinestring feature given a multilinestring WKT fragment.

Parameters
str {String} A WKT fragment representing the multilinestring

Returns
{ZOO.Feature} A multilinestring feature

parse.polygon Return a polygon feature given a polygon WKT fragment.

Parameters
str {String} A WKT fragment representing the polygon

Returns
{ZOO.Feature} A polygon feature

parse.multipolygon Return a multipolygon feature given a multipolygon WKT fragment.

Parameters
str {String} A WKT fragment representing the multipolygon

Returns
{ZOO.Feature} A multipolygon feature

parse.geometrycollection Return an array of features given a geometrycollection WKT fragment.

Parameters
str {String} A WKT fragment representing the geometrycollection

Returns
{Array} An array of ZOO.Feature

ZOO.Format.WPS

Read/Write WPS.
Inherits from
• ZOO.Format

Functions and Properties

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>schemaLocation</td>
<td>{String} Schema location for a particular minor version.</td>
</tr>
<tr>
<td>namespaces</td>
<td>{Object} Mapping of namespace aliases to namespace URIs.</td>
</tr>
<tr>
<td>read</td>
<td>Object containing methods to analyse data response.</td>
</tr>
<tr>
<td>parseExecuteResponse</td>
<td>Given an Object representing the WPS complex data response.</td>
</tr>
<tr>
<td>parseData</td>
<td>Given an Object representing the WPS literal data response.</td>
</tr>
<tr>
<td>parseData.complexdata</td>
<td>Given an Object representing the WPS reference response.</td>
</tr>
<tr>
<td>parseData.literaldata</td>
<td></td>
</tr>
<tr>
<td>parseData.reference</td>
<td></td>
</tr>
</tbody>
</table>

schemaLocation {String} Schema location for a particular minor version.
namespaces  

A Mapping of namespace aliases to namespace URIs.

read

read:function(data)

Parameters

data {String} A WPS xml document

Returns

{Object} Execute response.

parseExecuteResponse

parseExecuteResponse: function(node)

Parameters

node {E4XElement} A WPS ExecuteResponse document

Returns

{Object} Execute response.

parseData  Object containing methods to analyse data response.

parseData.complexdata  Given an Object representing the WPS complex data response.

Parameters

node {E4XElement} A WPS node.

Returns

{Object} A WPS complex data response.

parseData.literaldata  Given an Object representing the WPS literal data response.

Parameters

node {E4XElement} A WPS node.

Returns

{Object} A WPS literal data response.

parseData.reference  Given an Object representing the WPS reference response.

Parameters

node {E4XElement} A WPS node.

Returns

{Object} A WPS reference response.

ZOO.Geometry

A Geometry is a description of a geographic object.
Properties

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>{String} A unique identifier for this geometry.</td>
</tr>
<tr>
<td>parent</td>
<td>{ZOO.Geometry} This is set when a Geometry is added as component of another geometry</td>
</tr>
<tr>
<td>bounds</td>
<td>{ZOO.Bounds} The bounds of this geometry</td>
</tr>
</tbody>
</table>

Functions

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZOO.Geometry</td>
<td>Creates a geometry object.</td>
</tr>
<tr>
<td>destroy</td>
<td>nullify references to prevent circular references and memory leaks</td>
</tr>
<tr>
<td>clone</td>
<td>Create a clone of this vector feature.</td>
</tr>
<tr>
<td>extendBounds</td>
<td>Moves the feature and redraws it at its new location</td>
</tr>
<tr>
<td>clearBounds</td>
<td>Nullify this components bounds and that of its parent as well.</td>
</tr>
<tr>
<td>getBounds</td>
<td>Get the bounds for this Geometry.</td>
</tr>
<tr>
<td>calculateBounds</td>
<td>Recalculate the bounds for the geometry.</td>
</tr>
<tr>
<td>toString</td>
<td>Returns the Well-Known Text representation of a geometry</td>
</tr>
<tr>
<td>ZOO.Geometry.fromWKT</td>
<td>Generate a geometry given a Well-Known Text string.</td>
</tr>
</tbody>
</table>

Properties

- **id** {String} A unique identifier for this geometry.
- **parent** {ZOO.Geometry} This is set when a Geometry is added as component of another geometry
- **bounds** {ZOO.Bounds} The bounds of this geometry

Functions

- **ZOO.Geometry** Creates a geometry object.
  - **destroy**: function()
    - Destroy this geometry.
  - **clone**: function()
    - Create a clone of this geometry. Does not set any non-standard properties of the cloned geometry.
    - **Returns**
      - **ZOO.Geometry** An exact clone of this geometry.
  - **extendBounds**: function(newBounds)
    - Extend the existing bounds to include the new bounds. If geometry’s bounds is not yet set, then set a new Bounds.
    - **Parameters**
      - newBounds {ZOO.Bounds}
clearBounds: function()

Nullify this component's bounds and that of its parent as well.

getBounds

getBounds: function()

Get the bounds for this Geometry. If bounds is not set, it is calculated again, this makes queries faster.

Returns

newBounds {ZOO.Bounds}

calculateBounds

calculateBounds: function()

Recalculate the bounds for the geometry.

toString

toString: function()

Returns the Well-Known Text representation of a geometry

Returns

{String} Well-Known Text

ZOO.Geometry.fromWKT

ZOO.Geometry.fromWKT = function(wkt)

Generate a geometry given a Well-Known Text string.

Parameters

wkt {String} A string representing the geometry in Well-Known Text.

Returns

{ZOO.Geometry} A geometry of the appropriate class.

ZOO.Geometry.Collection

A Collection is exactly what it sounds like: A collection of different Geometries. These are stored in the local parameter components (which can be passed as a parameter to the constructor).

As new geometries are added to the collection, they are NOT cloned. When removing geometries, they need to be specified by reference (i.e., you have to pass in the exact geometry to be removed).

The <getArea> and getLength functions here merely iterate through the components, summing their respective areas and lengths.

Create a new instance with the ZOO.Geometry.Collection constructor.

 Inherits from

• ZOO.Geometry
Properties

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>components</td>
<td>{Array(ZOO.Geometry)} The component parts of this geometry</td>
</tr>
<tr>
<td>componentTypes</td>
<td>{Array(String)} An array of class names representing the types of components that the collection can include.</td>
</tr>
</tbody>
</table>

Functions

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZOO.Geometry.Collection</td>
<td>Creates a Geometry Collection – a list of geoms.</td>
</tr>
<tr>
<td>destroy</td>
<td>Destroy this geometry.</td>
</tr>
<tr>
<td>clone</td>
<td>Clone this geometry.</td>
</tr>
<tr>
<td>getComponentsString</td>
<td>Get a string representing the components for this collection.</td>
</tr>
<tr>
<td>calculateBounds</td>
<td>Recalculate the bounds by iterating through the components and calling extendBounds() on each item.</td>
</tr>
<tr>
<td>addComponent</td>
<td>Add a new component (geometry) to the collection.</td>
</tr>
<tr>
<td>removeComponents</td>
<td>Remove components from this geometry.</td>
</tr>
<tr>
<td>removeComponent</td>
<td>Remove a component from this geometry.</td>
</tr>
<tr>
<td>getLength</td>
<td>Calculate the length of this geometry.</td>
</tr>
<tr>
<td>getGeodesicLength</td>
<td>Calculate the approximate length of the geometry were it projected onto the earth.</td>
</tr>
<tr>
<td>move</td>
<td>Moves a geometry by the given displacement along positive x and y axes.</td>
</tr>
<tr>
<td>rotate</td>
<td>Rotate a geometry around some origin.</td>
</tr>
<tr>
<td>resize</td>
<td>Resize a geometry relative to some origin.</td>
</tr>
<tr>
<td>equals</td>
<td>Determine whether another geometry is equivalent to this one.</td>
</tr>
<tr>
<td>transform</td>
<td>Reproject the components geometry from source to dest.</td>
</tr>
<tr>
<td>intersects</td>
<td>Determine if the input geometry intersects this one.</td>
</tr>
<tr>
<td>getVertices</td>
<td>Return a list of all points in this geometry.</td>
</tr>
</tbody>
</table>

Properties

components  {Array(ZOO.Geometry)} The component parts of this geometry

componentTypes  {Array(String)} An array of class names representing the types of components that the collection can include. A null value means the component types are not restricted.

Functions

ZOO.Geometry.Collection  Creates a Geometry Collection – a list of geoms.

Parameters

components  {Array(ZOO.Geometry)} Optional array of geometries

destroy

destroy: function ()

Destroy this geometry.

close

close: function ()

Clone this geometry.

Returns
An exact clone of this collection.

getComponentsString

getComponentsString: function()

Get a string representing the components for this collection.

Returns

{String} A string representation of the components of this geometry.

calculateBounds

calculateBounds: function()

Recalculate the bounds by iterating through the components and calling extendBounds() on each item.

addComponent

addComponent: function(component, index)

Add a new component (geometry) to the collection. If this.componentTypes is set, then the component class name must be in the componentTypes array. The bounds cache is reset.

Parameters

component {ZOO.Geometry} A geometry to add
index {int} Optional index into the array to insert the component

Returns

{Boolean} The component geometry was successfully added

removeComponents

removeComponents: function(components)

Remove components from this geometry.

Parameters

components {Array(ZOO.Geometry)} The components to be removed

removeComponent

removeComponent: function(component)

Remove a component from this geometry.

Parameters

component {ZOO.Geometry}

getLength

getLength: function()

Calculate the length of this geometry.

Returns

{Float} The length of the geometry
getCentroid

centroid: function()

Returns

[ZOO.Geometry.Point] The centroid of the collection

getGeodesicLength

calculateGeodesicLength: function(projection)

Calculate the approximate length of the geometry were it projected onto the earth.

Parameters

projection [ZOO.Projection] The spatial reference system for the geometry coordinates. If not provided, Geographic/WGS84 is assumed.

Returns

{Float} The approximate geodesic length of the geometry in meters.

move

move: function(x,y)

Moves a geometry by the given displacement along positive x and y axes. This modifies the position of the geometry and clears the cached bounds.

Parameters

x {Float} Distance to move geometry in positive x direction.
y {Float} Distance to move geometry in positive y direction.

rotate

rotate: function(angle,origin)

Rotate a geometry around some origin

Parameters

angle {Float} Rotation angle in degrees (measured counterclockwise from the positive x-axis)
origin [ZOO.Geometry.Point] Center point for the rotation

resize

resize: function(scale,origin,ratio)

Resize a geometry relative to some origin. Use this method to apply a uniform scaling to a geometry.

Parameters

scale {Float} Factor by which to scale the geometry. A scale of 2 doubles the size of the geometry in each dimension (lines, for example, will be twice as long, and polygons will have four times the area).
origin {ZOO.Geometry.Point} Point of origin for resizing
ratio {Float} Optional x:y ratio for resizing. Default ratio is 1.

>Returns
{ZOO.Geometry} The current geometry.

equals
equals: function(geometry)
Determine whether another geometry is equivalent to this one. Geometries are considered equivalent if all components have the same coordinates.

>Parameters
geom {ZOO.Geometry} The geometry to test.

>Returns
{Boolean} The supplied geometry is equivalent to this geometry.

transform
transform: function(source, dest)
Reproject the components geometry from source to dest.

>Parameters
source {ZOO.Projection}
dest {ZOO.Projection}

(Returns
{ZOO.Geometry}

intersects
intersects: function(geometry)
Determine if the input geometry intersects this one.

>Parameters
geometry {ZOO.Geometry} Any type of geometry.

>Returns
{Boolean} The input geometry intersects this one.

getVertices
getVertices: function(nodes)
Return a list of all points in this geometry.

>Parameters
nodes {Boolean} For lines, only return vertices that are endpoints. If false, for lines, only vertices that are not endpoints will be returned. If not provided, all vertices will be returned.
Returns

{Array} A list of all vertices in the geometry.

ZOO.Geometry.Curve

A Curve is a MultiPoint, whose points are assumed to be connected. To this end, we provide a “getLength()” function, which iterates through the points, summing the distances between them.

Inherits from

- ZOO.Geometry.MultiPoint

Properties

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>componentTypes</td>
<td>{Array(String)} An array of class names representing the types of components that the collection can include.</td>
</tr>
</tbody>
</table>

Functions

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZOO.Geometry.Curve</td>
<td></td>
</tr>
<tr>
<td>getLength</td>
<td>{Float} The length of the curve</td>
</tr>
</tbody>
</table>

Properties

componentTypes {Array(String)} An array of class names representing the types of components that the collection can include. A null value means the component types are not restricted.

Functions

ZOO.Geometry.Curve

Parameters

point {Array(ZOO.Geometry.Point)}

getLength

gLength: function()

Returns

{Float} The length of the curve

ZOO.Geometry.LinearRing

A Linear Ring is a special LineString which is closed. It closes itself automatically on every addPoint/removePoint by adding a copy of the first point as the last point.

Also, as it is the first in the line family to close itself, a getArea() function is defined to calculate the enclosed area of the linearRing

Inherits from

- ZOO.Geometry.LineString
Properties

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>componentTypes</td>
<td>{Array(String)} An array of class names representing the types of components that the collection can include.</td>
</tr>
</tbody>
</table>

Functions

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZOO.Geometry.LinearRing</td>
<td>Linear rings are constructed with an array of points.</td>
</tr>
<tr>
<td>addComponent</td>
<td>Adds a point to geometry components.</td>
</tr>
<tr>
<td>move</td>
<td>Moves a geometry by the given displacement along positive x and y axes.</td>
</tr>
<tr>
<td>rotate</td>
<td>Rotate a geometry around some origin.</td>
</tr>
<tr>
<td>resize</td>
<td>Resize a geometry relative to some origin.</td>
</tr>
<tr>
<td>transform</td>
<td>Reproject the components geometry from source to dest.</td>
</tr>
<tr>
<td>getCentroid</td>
<td>{ZOO.Geometry.Point} The centroid of the ring</td>
</tr>
<tr>
<td>getArea</td>
<td>Calculate the approximate area of the polygon were it projected onto the earth.</td>
</tr>
<tr>
<td>getGeodesicArea</td>
<td>Test if a point is inside a linear ring.</td>
</tr>
<tr>
<td>containsPoint</td>
<td></td>
</tr>
</tbody>
</table>

Properties

componentTypes {Array(String)} An array of class names representing the types of components that the collection can include. A null value means the component types are not restricted.

Functions

ZOO.Geometry.LinearRing Linear rings are constructed with an array of points. This array can represent a closed or open ring. If the ring is open (the last point does not equal the first point), the constructor will close the ring. If the ring is already closed (the last point does equal the first point), it will be left closed.

Parameters

points {Array(ZOO.Geometry.Point)} points

addComponent

addComponent: function(point,index)

Adds a point to geometry components. If the point is to be added to the end of the components array and it is the same as the last point already in that array, the duplicate point is not added. This has the effect of closing the ring if it is not already closed, and doing the right thing if it is already closed. This behavior can be overridden by calling the method with a non-null index as the second argument.

Parameter

point /ZOO.Geometry.Point
index {Integer} Index into the array to insert the component

Returns

{Boolean} Was the Point successfully added?

move
move: function(x,y)

Moves a geometry by the given displacement along positive x and y axes. This modifies the position of the geometry and clears the cached bounds.

Parameters

x {Float} Distance to move geometry in positive x direction.
y {Float} Distance to move geometry in positive y direction.

rotate

rotate: function(angle,origin)

Rotate a geometry around some origin

Parameters

angle {Float} Rotation angle in degrees (measured counterclockwise from the positive x-axis)
origin [ZOO.Geometry.Point] Center point for the rotation

resize

resize: function(scale,origin,ratio)

Resize a geometry relative to some origin. Use this method to apply a uniform scaling to a geometry.

Parameters

scale {Float} Factor by which to scale the geometry. A scale of 2 doubles the size of the geometry in each dimension (lines, for example, will be twice as long, and polygons will have four times the area).
origin [ZOO.Geometry.Point] Point of origin for resizing
ratio {Float} Optional x:y ratio for resizing. Default ratio is 1.

Returns

{ZOO.Geometry} The current geometry.

transform

transform: function(source,dest)

Reproject the components geometry from source to dest.

Parameters

source [ZOO.Projection]
dest [ZOO.Projection]
Returns

{ZOO.Geometry}

gCentroid

gCentroid: function()

Returns

{ZOO.Geometry.Point} The centroid of the ring

gArea

gArea: function()

Note: The area is positive if the ring is oriented CW, otherwise it will be negative.

Returns

{Float} The signed area for a ring.

gGeodesicArea

gGeodesicArea: function(projection)

Calculate the approximate area of the polygon were it projected onto the earth. Note that this area will be positive if ring is oriented clockwise, otherwise it will be negative.

Parameters

projection {ZOO.Projection} The spatial reference system for the geometry coordinates. If not provided, Geographic/WGS84 is assumed.

Reference


Returns

{float} The approximate signed geodesic area of the polygon in square meters.

containsPoint

containsPoint: function(point)

Test if a point is inside a linear ring. For the case where a point is coincident with a linear ring edge, returns 1. Otherwise, returns boolean.

Parameters

point {ZOO.Geometry.Point}

Returns

{Boolean | Number} The point is inside the linear ring. Returns 1 if the point is coincident with an edge. Returns boolean otherwise.
ZOO.Geometry.LineString

A LineString is a Curve which, once two points have been added to it, can never be less than two points long. Inherits from

- ZOO.Geometry.Curve

Functions

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZOO.Geometry.LineString</td>
<td>Create a new LineString geometry</td>
</tr>
<tr>
<td>removeComponent</td>
<td>Only allows removal of a point if there are three or more points in the linestring. Test for intersection between two geometries. {Array} An array of segment objects.</td>
</tr>
<tr>
<td>intersects</td>
<td>Test for intersection between two geometries. This is a cheapo implementation of the Bently-Ottmann algorithm. It doesn’t really keep track of a sweep line data structure. It is closer to the brute force method, except that segments are sorted and potential intersections are only calculated when bounding boxes intersect. {Boolean} The input geometry intersects this geometry.</td>
</tr>
<tr>
<td>getSortedSegments</td>
<td>Returns {Array} An array of segment objects. Segment objects have properties x1, y1, x2, and y2. The start point is represented by x1 and y1. The end point is represented by x2 and y2. Start and end are ordered so that x1 &lt; x2.</td>
</tr>
</tbody>
</table>

Parameters

points {Array(ZOO.Geometry.Point)} An array of points used to generate the linestring

point {ZOO.Geometry.Point} The point to be removed

geometry {ZOO.Geometry}
splitWithSegment

`splitWithSegment: function(seg, options)`  

Split this geometry with the given segment.

**Parameters**

- `seg` **Object** An object with `x1`, `y1`, `x2`, and `y2` properties referencing segment endpoint coordinates.
- `options` **Object** Properties of this object will be used to determine how the split is conducted.

**Valid options**

- `edge` **Boolean** Allow splitting when only edges intersect. Default is true. If false, a vertex on the source segment must be within the tolerance distance of the intersection to be considered a split.
- `tolerance` **Number** If a non-null value is provided, intersections within the tolerance distance of one of the source segment’s endpoints will be assumed to occur at the endpoint.

**Returns**

An object with lines and points properties. If the given segment intersects this linestring, the lines array will reference geometries that result from the split. The points array will contain all intersection points. Intersection points are sorted along the segment (in order from `x1`, `y1` to `x2`, `y2`).

split

`split: function(target, options)`  

Use this geometry (the source) to attempt to split a target geometry.

**Parameters**

- `target` **ZOO.Geometry** The target geometry.
- `options` **Object** Properties of this object will be used to determine how the split is conducted.

**Valid options**

- `mutual` **Boolean** Split the source geometry in addition to the target geometry. Default is false.
- `edge` **Boolean** Allow splitting when only edges intersect. Default is true. If false, a vertex on the source must be within the tolerance distance of the intersection to be considered a split.
- `tolerance` **Number** If a non-null value is provided, intersections within the tolerance distance of an existing vertex on the source will be assumed to occur at the vertex.

**Returns**

An array of geometries (of this same type as the target) that result from splitting the target with the source geometry. The source and target geometry will remain unmodified. If no split results, null will be returned. If mutual is true and a split results, return will be an array of two arrays - the first will be all geometries that result
from splitting the source geometry and the second will be all geometries that result from splitting the target geometry.

**splitWith**

```javascript
splitWith: function(geometry, options)
```

Split this geometry (the target) with the given geometry (the source).

**Parameters**

- `geometry` **{ZOO.Geometry}** A geometry used to split this geometry (the source).
- `options` **{Object}** Properties of this object will be used to determine how the split is conducted.

**Valid options**

- `mutual` **{Boolean}** Split the source geometry in addition to the target geometry. Default is false.
- `edge` **{Boolean}** Allow splitting when only edges intersect. Default is true. If false, a vertex on the source must be within the tolerance distance of the intersection to be considered a split.
- `tolerance` **{Number}** If a non-null value is provided, intersections within the tolerance distance of an existing vertex on the source will be assumed to occur at the vertex.

**Returns**

**{Array}** A list of geometries (of this same type as the target) that result from splitting the target with the source geometry. The source and target geometry will remain unmodified. If no split results, null will be returned. If mutual is true and a split results, return will be an array of two arrays - the first will be all geometries that result from splitting the source geometry and the second will be all geometries that result from splitting the target geometry.

**getVertices**

```javascript
getVertices: function(nodes)
```

Return a list of all points in this geometry.

**Parameters**

- `nodes` **{Boolean}** For lines, only return vertices that are endpoints. If false, for lines, only vertices that are not endpoints will be returned. If not provided, all vertices will be returned.

**Returns**

**{Array}** A list of all vertices in the geometry.

**ZOO.Geometry.MultiLineString**

A MultiLineString is a geometry with multiple ZOO.Geometry.LineString components.

Inherits from

- **ZOO.Geometry.Collection**
## Functions

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
</table>

**ZOO.Geometry.MultiLineString** Constructor for a MultiLineString Geometry.

**Parameters**

- components {Array(ZOO.Geometry.LineString)}

## ZOO.Geometry.MultiPoint

MultiPoint is a collection of Points. Create a new instance with the `ZOO.Geometry.MultiPoint` constructor.

Inherits from

- `ZOO.Geometry.Collection`

### Properties

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>componentTypes</td>
<td>{Array(String)} An array of class names representing the types of components that the collection can include.</td>
</tr>
</tbody>
</table>

### Functions

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZOO.Geometry.MultiPoint</td>
<td>Create a new MultiPoint Geometry</td>
</tr>
</tbody>
</table>

**Parameters**

- components {Array(ZOO.Geometry.Point)}

**Returns**

- [ZOO.Geometry.MultiPoint]

**addPoint**

- addPoint: function(point, index)

Wrapper for `ZOO.Geometry.Collection.addComponent`

**Parameters**

- point [ZOO.Geometry.Point] Point to be added
index (Integer) Optional index

removePoint

removePoint: function(point)

Wrapper for ZOO.Geometry.Collection.removeComponent

Parameters

point (ZOO.Geometry.Point) Point to be removed

ZOO.Geometry.MultiPolygon

MultiPolygon is a geometry with multiple ZOO.Geometry.Polygon components. Create a new instance with the ZOO.Geometry.MultiPolygon constructor.

Inherits from

• ZOO.Geometry.Collection

Functions

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZOO.Geometry.MultiPolygon</td>
<td>Create a new MultiPolygon geometry</td>
</tr>
</tbody>
</table>

ZOO.Geometry.MultiPolygon Create a new MultiPolygon geometry

Parameters

components (Array(ZOO.Geometry.Polygon)) An array of polygons used to generate the MultiPolygon

ZOO.Geometry.Point

Point geometry class.

Inherits from

• ZOO.Geometry

Properties

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>{float}</td>
</tr>
<tr>
<td>y</td>
<td>{float}</td>
</tr>
</tbody>
</table>
Functions

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZOO.Geometry.Point clone</td>
<td>{ZOO.Geometry.Point} An exact clone of this ZOO.Geometry.Point</td>
</tr>
<tr>
<td>calculateBounds</td>
<td>Create a new Bounds based on the x/y</td>
</tr>
<tr>
<td>equals</td>
<td>Determine whether another geometry is equivalent to this one.</td>
</tr>
<tr>
<td>toShortString</td>
<td>{String} Shortened String representation of Point object.</td>
</tr>
<tr>
<td>move</td>
<td>Moves a geometry by the given displacement along positive x and y axes.</td>
</tr>
<tr>
<td>rotate</td>
<td>Rotate a point around another.</td>
</tr>
<tr>
<td>getCentroid</td>
<td>{ZOO.Geometry.Point} The centroid of the collection</td>
</tr>
<tr>
<td>resize</td>
<td>Resize a point relative to some origin.</td>
</tr>
<tr>
<td>intersects</td>
<td>Determine if the input geometry intersects this one.</td>
</tr>
<tr>
<td>transform</td>
<td>Translate the x,y properties of the point from source to dest.</td>
</tr>
<tr>
<td>getVertices</td>
<td>Return a list of all points in this geometry.</td>
</tr>
</tbody>
</table>

Properties

x {float}
y {float}

Functions

ZOO.Geometry.Point  Construct a point geometry.

Parameters

x {float}
y {float}

cloned

cloned: function(obj)

Returns

{ZOO.Geometry.Point} An exact clone of this ZOO.Geometry.Point

calculateBounds

calculateBounds: function ()

Create a new Bounds based on the x/y

equals

equals: function(geom)

Determine whether another geometry is equivalent to this one. Geometries are considered equivalent if all components have the same coordinates.

Parameters

geom {ZOO.Geometry.Point} The geometry to test.

Returns

{Boolean} The supplied geometry is equivalent to this geometry.
toShortString

toShortString: function()

Returns

{String} Shortened String representation of Point object. (ex. "5, 42")

move

move: function(x,y)

Moves a geometry by the given displacement along positive x and y axes. This modifies the position of the
geometry and clears the cached bounds.

Parameters

x {Float} Distance to move geometry in positive x direction.
y {Float} Distance to move geometry in positive y direction.

rotate

rotate: function(angle,origin)

Rotate a point around another.

Parameters

angle {Float} Rotation angle in degrees (measured counterclockwise from the positive x-axis)
origin {ZOO.Geometry.Point} Center point for the rotation

gCentroid

gCentroid: function()

Returns

{ZOO.Geometry.Point} The centroid of the collection

resize

resize: function(scale,origin,ratio)

Resize a point relative to some origin. For points, this has the effect of scaling a vector (from the origin to the
point). This method is more useful on geometry collection subclasses.

Parameters

scale {Float} Ratio of the new distance from the origin to the old distance from the origin. A scale of 2
doubles the distance between the point and origin.
origin {ZOO.Geometry.Point} Point of origin for resizing
ratio {Float} Optional x:y ratio for resizing. Default ratio is 1.
Returns

{ZOO.Geometry} The current geometry.

intersects

intersects: function(geometry)

Determine if the input geometry intersects this one.

Parameters

geometry {ZOO.Geometry} Any type of geometry.

Returns

{Boolean} The input geometry intersects this one.

transform

transform: function(source, dest)

Translate the x,y properties of the point from source to dest.

Parameters

source {ZOO.Projection}
dest {ZOO.Projection}

Returns

{ZOO.Geometry}

getVertices

getVertices: function(nodes)

Return a list of all points in this geometry.

Parameters

nodes {Boolean} For lines, only return vertices that are endpoints. If false, for lines, only vertices that are not endpoints will be returned. If not provided, all vertices will be returned.

Returns

{Array} A list of all vertices in the geometry.

ZOO.Geometry.Polygon

Polygon is a collection of ZOO.Geometry.LinearRing.

Inherits from

- ZOO.Geometry.Collection
### Functions

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZOO.Geometry.Polygon</td>
<td>Constructor for a Polygon geometry. The first ring (<em>this.component[0]</em>) is the outer bounds of the polygon and all subsequent rings (<em>this.component[1-n]</em>) are internal holes.</td>
</tr>
<tr>
<td>getArea</td>
<td>Calculated by subtracting the areas of the internal holes from the area of the outer hole.</td>
</tr>
<tr>
<td>containsPoint</td>
<td>Test if a point is inside a polygon. Points on a polygon edge are considered inside.</td>
</tr>
<tr>
<td>createRegularPolygon</td>
<td>Create a regular polygon around a radius. Useful for creating circles and the like.</td>
</tr>
</tbody>
</table>

**getArea**

```javascript
getArea: function()
```

Calculated by subtracting the areas of the internal holes from the area of the outer hole.

**Returns**

*float* The area of the geometry

**containsPoint**

```javascript
containsPoint: function(point)
```

Test if a point is inside a polygon. Points on a polygon edge are considered inside.

**Parameters**

- **point** [ZOO.Geometry.Point]

**Returns**

*{Boolean | Number}* The point is inside the polygon. Returns 1 if the point is on an edge. Returns boolean otherwise.

**createRegularPolygon**

```javascript
ZOO.Geometry.Polygon.createRegularPolygon = function( origin, radius, sides, rotation )
```

Create a regular polygon around a radius. Useful for creating circles and the like.

**Parameters**

- **origin** [ZOO.Geometry.Point] center of polygon.
- **radius** [Float] distance to vertex, in map units.
- **sides** [Integer] Number of sides. 20 approximates a circle.
- **rotation** [Float] original angle of rotation, in degrees.

---

**ZOO.Geometry.Surface**

Surface geometry class.
Inherits from

- ZOO.Geometry

ZOO.Process

Used to query OGC WPS process defined by its URL and its identifier. Useful for chaining localhost process.

Properties and Functions

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>schemaLocation</td>
<td>{String} Schema location for a particular minor version.</td>
</tr>
<tr>
<td>namespaces</td>
<td>{Object} Mapping of namespace aliases to namespace URIs.</td>
</tr>
<tr>
<td>url</td>
<td>{String} The OGC’s Web Processing Service URL, default is <a href="http://localhost/zoo">http://localhost/zoo</a>.</td>
</tr>
<tr>
<td>identifier</td>
<td>{String} Process identifier in the OGC’s Web Processing Service.</td>
</tr>
<tr>
<td>ZOO.Process</td>
<td>Create a new Process</td>
</tr>
<tr>
<td>Execute</td>
<td>Query the OGC’s Web Processing Service to Execute the process.</td>
</tr>
<tr>
<td>buildInput</td>
<td>Object containing methods to build WPS inputs.</td>
</tr>
<tr>
<td>buildInput.complex</td>
<td>Given an E4XElement representing the WPS complex data input.</td>
</tr>
<tr>
<td>buildInput.reference</td>
<td>Given an E4XElement representing the WPS reference input.</td>
</tr>
<tr>
<td>buildInput.literal</td>
<td>Given an E4XElement representing the WPS literal data input.</td>
</tr>
<tr>
<td>buildDataInputsNode</td>
<td>Method to build the WPS DataInputs element.</td>
</tr>
</tbody>
</table>

schemaLocation  {String} Schema location for a particular minor version.

namespaces    {Object} Mapping of namespace aliases to namespace URIs.

url    {String} The OGC’s Web Processing Service URL, default is http://localhost/zoo.

identifier    {String} Process identifier in the OGC’s Web Processing Service.

ZOO.Process  Create a new Process

Parameters

url    {String} The OGC’s Web Processing Service URL.

identifier    {String} The process identifier in the OGC’s Web Processing Service.

Execute

Execute: function(inputs)

Query the OGC’s Web Processing Service to Execute the process.

Parameters

inputs    {Object}

Returns

{String} The OGC’s Web processing Service XML response. The result needs to be interpreted.

buildInput  Object containing methods to build WPS inputs.

buildInput.complex  Given an E4XElement representing the WPS complex data input.

Parameters
identifier {String} the input identifier
data {Object} A WPS complex data input.

Returns
{E4XElement} A WPS Input node.

buildInput.reference Given an E4XElement representing the WPS reference input.

Parameters

identifier {String} the input identifier
data {Object} A WPS reference input.

Returns
{E4XElement} A WPS Input node.

buildInput.literal Given an E4XElement representing the WPS literal data input.

Parameters

identifier {String} the input identifier
data {Object} A WPS literal data input.

Returns
{E4XElement} The WPS Input node.

buildDataInputsNode

buildDataInputsNode:function(inputs)

Method to build the WPS DataInputs element.

Parameters
inputs {Object}

Returns
{E4XElement} The WPS DataInputs node for Execute query.

ZOO.Projection

Class for coordinate transforms between coordinate systems.

Properties

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>proj</td>
<td>{Number} Proj4js.Proj instance.</td>
</tr>
<tr>
<td>projCode</td>
<td>{String}</td>
</tr>
</tbody>
</table>
Functions

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZOO.Projection</td>
<td>This class offers several methods for interacting with a wrapped zoo-pro4js projection object.</td>
</tr>
<tr>
<td>getCode</td>
<td>Get the string SRS code.</td>
</tr>
<tr>
<td>getUnits</td>
<td>Get the units string for the projection – returns null if zoo-proj4js is not available.</td>
</tr>
<tr>
<td>toString</td>
<td>Convert projection to string (getCode wrapper).</td>
</tr>
<tr>
<td>equals</td>
<td>Test equality of two projection instances.</td>
</tr>
<tr>
<td>destroy</td>
<td>Destroy projection object.</td>
</tr>
<tr>
<td>transform</td>
<td>Transform a point coordinate from one projection to another.</td>
</tr>
</tbody>
</table>

Properties

proj `{Object}` Proj4js.Proj instance.
projCode `{String}`

Functions

ZOO.Projection  This class offers several methods for interacting with a wrapped zoo-pro4js projection object.

Parameters

projCode `{String}` A string identifying the Well Known Identifier for the projection.
options `{Object}` An optional object to set additional properties.

Returns

[ZOO.Projection] A projection object.

getCode

getCode: function()

Get the string SRS code.

Returns

{String} The SRS code.

ggetUnits

ggetUnits: function()

Get the units string for the projection – returns null if zoo-proj4js is not available.

Returns

{String} The units abbreviation.

toString

toString: function()

Convert projection to string (getCode wrapper).

Returns

{String} The projection code.

equals
equals: function(projection)

Test equality of two projection instances. Determines equality based solely on the projection code.

Returns

{Boolean} The two projections are equivalent.

destroy

destroy: function()

Destroy projection object.

transform

ZOO.Projection.transform = function(point, source, dest)

Transform a point coordinate from one projection to another. Note that the input point is transformed in place.

Parameters

point {{ZOO.Geometry.Point> | Object} An object with x and y properties representing coordinates in those dimensions.
sourceProj {ZOO.Projection} Source map coordinate system
destProj {ZOO.Projection} Destination map coordinate system

Returns

point {object} A transformed coordinate. The original point is modified.

ZOO.Request

Contains convenience methods for working with ZOORequest which replace XMLHttpRequest.

Functions

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Send an HTTP GET request.</td>
</tr>
<tr>
<td>POST</td>
<td>Send an HTTP POST request.</td>
</tr>
</tbody>
</table>

GET  Send an HTTP GET request.

Parameters

url {String} The URL to request.
params {Object} Params to add to the url

Returns

{String} Request result.
POST  Send an HTTP POST request.

  Parameters

  url {String}  The URL to request.
  body {String}  The request’s body to send.
  headers {Object}  A key-value object of headers to push to the request’s head

  Returns

  {String}  Request result.

**ZOO.String**

Contains convenience methods for string manipulation:

**Functions and Properties**

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>startsWith</td>
<td>Test whether a string starts with another string.</td>
</tr>
<tr>
<td>contains</td>
<td>Test whether a string contains another string.</td>
</tr>
<tr>
<td>trim</td>
<td>Removes leading and trailing whitespace characters from a string.</td>
</tr>
<tr>
<td>camelize</td>
<td>Camel-case a hyphenated string.</td>
</tr>
<tr>
<td>tokenRegEx</td>
<td>Used to find tokens in a string.</td>
</tr>
<tr>
<td>numberRegEx</td>
<td>Used to test strings as numbers.</td>
</tr>
<tr>
<td>isNumeric</td>
<td>Determine whether a string contains only a numeric value.</td>
</tr>
<tr>
<td>numericIf</td>
<td>Converts a string that appears to be a numeric value into a number.</td>
</tr>
</tbody>
</table>

**startsWith**

`startsWith: function(str,sub)`

Test whether a string starts with another string.

  Parameters

  str {String}  The string to test.
  sub {String}  The substring to look for.

  Returns

  {Boolean}  The first string starts with the second.

**contains**

`contains: function(str,sub)`

Test whether a string contains another string.

  Parameters

  str {String}  The string to test.
sub (String) The substring to look for.

Returns
{Boolean} The first string contains the second.

trim
trim: function(str)
Removes leading and trailing whitespace characters from a string.

Parameters
str {String} The (potentially) space-padded string. This string is not modified.

Returns
{String} A trimmed version of the string with all leading and trailing spaces removed.

camelize
camelize: function(str)
Camel-case a hyphenated string. Ex. “chicken-head” becomes “chickenHead”, and “-chicken-head” becomes “ChickenHead”.

Parameters
str {String} The string to be camelized. The original is not modified.

Returns
{String} The string, camelized

tokenRegEx Used to find tokens in a string. Examples: ${a}, ${a.b.c}, ${a-b}, ${5}

numberRegEx Used to test strings as numbers.

isNumeric
isNumeric: function(value)
Determine whether a string contains only a numeric value.

Examples
ZOO.String.isNumeric("6.02e23") // true
ZOO.String.isNumeric("12 dozen") // false
ZOO.String.isNumeric("4") // true
ZOO.String.isNumeric(" 4 ") // false

Returns
{Boolean} String contains only a number.

numericIf
numericIf: function(value)
Converts a string that appears to be a numeric value into a number.

Returns
{Number|String} a Number if the passed value is a number, a String otherwise.
5.3.3 Examples

In this page you can find some small examples on how to use the JavaScript ZOO-API on the server side. ZOO-API contains many classes and functions. You can find the description list here.

ZOO.Process example of use

```javascript
function SampleService(conf, inputs, outputs) {
    var myProcess = new ZOO.Process('http://localhost/cgi-bin-new1/zoo_loader_new1.cgi', 'Boundary');
    var myInputs = {InputPolygon: { type: 'complex', value: '{"type": "Polygon", "coordinates": [[-106.9,...
    var myExecuteResult = myProcess.Execute(myInputs);
    return {result: ZOO.SERVICE_SUCCEEDED, outputs: [{name: "Result", value: myExecuteResult}]};
}
```

In this really short example you can see how to create ZOO.Process class instance and call the Execute method on such an instance. Then you’ll just need to return a JavaScript object containing the attributes result and outputs, which I’m sure you already know what is about. The first is about the status of the process (can be ZOO.SERVICE_SUCCEEDED, ZOO.SERVICE_FAILED and so on), the last is obviously the resulting maps (take a look at the maps internal data structure used by ZOO Kernel in service.h).

ZOO.UpdateStatus

```javascript
function SampleLongService(conf, inputs, outputs) {
    var my_i = 0;
    while (my_i < 100) {
        try {
            conf['lenv']['status'] = my_i;
        } catch (e) {
        }
        ZOOUpdateStatus(conf, my_i);
        SampleService(conf, inputs, outputs);
        my_i += 10;
    }
    return SampleService(conf, inputs, outputs);
}
```

You can see in this sample code how to use the ZOOUpdateStatus function to update the current status of your running process. This information will be really helpful when the ZOO Kernel will run your JavaScript Service in background mode (if the user set to true the storeExecuteResponse parameter in his request).

5.4 ZOO-Client Documentation

The following sections will assist you with the ZOO-Client:

5.4.1 How To Use ZOO-Client

**Authors** Nicolas Bozon, Luca Delucchi, Gérald Fenoy, Jeff McKenna

**Last Updated** $Id: introduction.txt 562 2015-02-09 15:26:31Z djay$

ZOO-Client is a client-side Javascript API which ease the use and integration of WPS in your web applications.
Table of Contents

- How To Use ZOO-Client
  - Requirements
    * Compile mustache templates
    * Build the API documentation
  - Create your first application
    * Loading the modules from your web application

Requirements

The ZOO-Client is depending on various Javascript libraries, they are exhaustively listed below:

- jQuery
- x2js
- requirejs
- Hogan.js
- query-string

Compile mustache templates

For being able to use the ZOO-Client API from your application you will need to compile the mustache templates files located in the tpl directory. This compilation process imply that you have setup node.js on your computer.

```
sudo npm install hogan
hulk zoo-client/lib/tpl/*mustache > \
  zoo-client/lib/js/wps-client/payloads.js
```

Note: the Hogan version used to compile the template should be the same as the one used from your web application, in other case you may face compatibility issue.

Build the API documentation

You can build the ZOO-Client API documentation by using jsDoc, to build the documentation use the following command:

```
~/.node_modules/.bin/jsdoc zoo-client/lib/js/wps-client/* -p
```

This will build the documentation in a directory named `out` in your current working directory.

Create your first application

For this first application, we will suppose that you have setup a directory named `zoo-client-demo` accessible from your server by using `http://localhost/zoo-client-demo`. In this directory, you should have the following subdirectories:
You will need to copy your node_modules javascript files copied in the hogan and query-string directories. First, you will need to install query-string.

`npm install query-string`

Then you will copy `query-string.js` and `hogan-3.0.2.js` files in your `zoo-client-demo` web directory. Those files are located in your `~/node_modules` directory.

For other libraries, you will need to download them from their official web sites and uncompress them in the corresponding directories.

### Loading the modules from your web application

Before using the ZOO-Client, you will first have to include the javascript files from your web page. With the use of requirejs you will need only one line in your HTML page to include everything at once. This line will look like the following:

```html
<script data-main="assets/js/first" src="assets/js/lib/require.js"></script>
```

In this example, we suppose that you have created a `first.js` file in the `assets/js` directory containing your main application code. First, you define there the required JavaScript libraries and potentially their configuration, then you can add any relevant code.

```javascript
requirejs.config({
  baseUrl: 'assets/js',
  paths: {
    jquery: 'lib/jquery/jquery-1.11.0.min',
    hogan: 'lib/hogan/hogan-3.0.2',
    xml2json: 'lib/xml2json/xml2json.min',
    queryString: 'lib/query-string/query-string',
    wpsPayloads: 'lib/zoo/payloads',
    wpsPayload: 'lib/zoo/wps-payload',
    utils: 'lib/zoo/utils',
    zoo: 'lib/zoo/zoo',
    domReady: 'lib/domReady',
    app: 'first-app',
  },
  shim: {
    wpsPayloads: {
      deps: ['hogan'],
    },
    wpsPayload: {
      deps: ['wpsPayloads'],
      exports: 'wpsPayload',
    },
    hogan: {
      exports: 'Hogan',
    },
  },
});
```
requirejs.config({
  config: {
    app: {
      url: '/cgi-bin/zoo_loader.cgi',
      delay: 2000,
    }
  }
});

require(["domReady", 'app'], function(domReady, app) {
  domReady(function() {
    app.initialize();
  });
});

On line 2, you define the url where your files are located on the web server, in assets/js. From line 3 to 14, you define the JavaScript files to be loaded. From line 15 to 21, you configure the dependencies and exported symbols. From line 35 to 42, you configure your main application.

In this application, we use the domReady module to call the initialize function defined in the app module, which is defined in the first-app.js file as defined on line 13.

define(['
  module', 'zoo', 'wpsPayload'
], function(module, ZooProcess, wpsPayload) {
  var myZooObject = new ZooProcess({
    url: module.config().url,
    delay: module.config().delay,
  });

  var initialize = function() {
    self = this;
    myZooObject.getCapabilities({
      type: 'POST',
      success: function(data){
        console.log(data);
      }
    });

    myZooObject.describeProcess({
      type: 'POST',
      identifier: "all",
      success: function(data){
        console.log(data);
      }
    });

    myZooObject.execute({
      identifier: "Buffer",
    });

    });

    myZooObject.describeProcess({
      type: 'POST',
      identifier: "all",
      success: function(data){
        console.log(data);
      }
    });

    myZooObject.execute({
      identifier: "Buffer",
    });
});
On line 5 you create a “global” ZooProcess instance named myZooObject, you set the url and delay to the values defined in first.js on line 35. From line 10 to 40, you define a simple initialize function which will invoke the getCapabilities (line 12 to 18), describeProcess (from line 20 to 26) and execute (from line 28 to 39) methods. For each you define a callback function which will simply display the resulting data in the browser’s console.

5.5 Deploying Web Processing Services using ZOO-Project – Examples of Python based WPS using PgRouting

**Author** Gérald Fenoy, Nicolas Bozon, Venkatesh Raghavan

**Contact** gerald.fenoy at geolabs.fr, nicolas.bozon at gmail.com, venka at osgeo.org

**Last Updated** $Date: 2015-02-09 19:21:38 +0100 (lun. 09 févr. 2015) $
5.5.2 Sponsored By

5.5.3 Special thanks to our Knowledge Partners

5.5.4 WorkShop table of content

Introduction

Table of Contents
- Introduction
  - What is ZOO?
  - How does ZOO works?
  - What are we going to do in this workshop?
  - Useful tips for reading

What is ZOO?

ZOO-Project is a WPS (Web Processing Service) open source project released under a MIT/X-11 style license. It provides an OGC WPS compliant developer-friendly framework to create and chain WPS Web services. ZOO is made of three parts:

- **ZOO Kernel**: A powerful server-side C Kernel which makes it possible to manage and chain Web services coded in different programming languages.
- **ZOO Services**: A growing suite of example Web Services based on various open source libraries.
- **ZOO API**: A server-side JavaScript API able to call and chain the ZOO Services, which makes the development and chaining processes easier and faster.

ZOO was designed to make the service creation and deployment easy, by providing a powerful system able to understand and execute WPS compliant queries. It supports seven programming languages, thus allowing you to create Web Services using the one you prefer. It also lets you use an existing code and to turn it as a WPS Service. The current supported programming languages are the following:

- C/C++
- Python
- Perl
• Java
• Fortran
• PHP
• JavaScript

More information on the project is available on the ZOO-Project official website.

How does ZOO works?

ZOO is based on a C Kernel which is the ZOO-Project core system (aka ZOO-Kernel). The latter is able to dynamically load libraries and to handle them as on-demand Web services.

A ZOO-Service is a link composed of a ZOO metadata file (.zcfg) and the code for the corresponding implementation. The metadata file describes the function that can be called using a WPS Execute Request, as well as the supported input/output formats. Service contains the algorithms and functions, and can be implemented using any of the supported languages.

ZOO-Kernel works as CGI through Apache and can communicate with cartographic engines and Web mapping clients. It simply adds the WPS support to your spatial data infrastructure and your webmapping applications. It can use every GDAL/OGR supported formats as input data and create suitable vector or raster output for your cartographic engine and/or your web-mapping client application.

What are we going to do in this workshop?

You will learn how to use ZOO-Kernel and how to create ZOO-Services using the OSGeoLiveDVD 7.9 iso image provided at the beginning of this workshop on MapMint USB Sticks. Despite a pre-compiled ZOO 1.3.0 package is provided inside the official OSGeoLive, some optional supports are not available in the default setup and the ZOO-Kernel made available correspond to the 1.4-dev version, so let say the current version. So you will use the binary version of ZOO-Kernel available on the iso image provided. Configuration file and basic ways to use ZOO-Kernel and ZOO-Service will be presented. Then you will be invited to start programming your own simple service using Python language. Some ZOO-Services will be presented and individually tested inside a ready-to-use OpenLayers application. Finally, these services will be chained using the server-side Javascript ZOO-API.

The whole workshop is organized step-by-step and numerous code snippets are available. The instructors will check the ZOO-Kernel is functioning on each machine and will assist you while coding. Technical questions are of course welcome during the workshop.

Usefull tips for reading

this is a code block

<table>
<thead>
<tr>
<th>Warning:</th>
<th>This is a warning message.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Note:</th>
<th>This is an important note.</th>
</tr>
</thead>
</table>

Let’s go!
Configuration and ZOO-Kernel use

Table of Contents
- Configuration and ZOO-Kernel use
  - ZOO-Kernel Configuration
  - Testing the ZOO installation with GetCapabilities

ZOO-Kernel Configuration

As already said in introduction, an OSGeoLive Virtual Machine has been booted on your computer, allowing you to use ZOO-Kernel in a development environment directly.

Note: we will use ZOO-Kernel or zoo_loader.cgi script without any distinction in this document.

General ZOO-Kernel settings are set in the main.cfg file located in the same directory as the ZOO-Kernel, so in /usr/lib/cgi-bin/. This informations will be accessible from each services at runtime, so when you will use Execute requests. You can see a typical main.cfg content in the following:

```
[headers]
X-Powered-By=ZOO-Project@MapMint

[main]
encoding=utf-8
dataPath=/var/data
tmpPath=/var/www/temp
cacheDir=/var/www/cache
version=1.0.0
sessPath=/tmp
msOgcVersion=1.0.0
serverAddress=http://localhost/cgi-bin/mm/zoo_loader.cgi
lang=fr-FR,ja-JP
language=en-US
mapserverAddress=http://localhost/cgi-bin/masserv.cgi
tmpUrl=http://127.0.0.1/temp/

[identification]
keywords=WPS,GIS,buffer,MapMint,ZOO-Project
title=ZOO-Project Workshop - FOSS4G 2014
abstract= Deploying Web Processing Services using ZOO-Project - Examples of Python based WPS using PostgreSQL/PostGIS
accessConstraints=none
fees=None

[provider]
positionName=Developer
providerName=GeoLabs SARL
addressAdministrativeArea=False
addressDeliveryPoint=1280, avenue des Platanes
addressCountry=fr
phoneVoice=+33467430995
addressPostalCode=34970
role=Dev
providerSite=http://www.geolabs.fr
phoneFacsimile=False
```
The main.cfg file contains metadata informations about the identification and provider but also some important settings. The file is composed of various sections, namely [main], [identification] and [provider] per default.

From the [main] section settings are as follow:

- **lang**: the supported languages separated by a coma (the first is the default one),
- **version**: the supported WPS version,
- **encoding**: the default encoding of WPS Responses,
- **serverAddress**: the url to access your ZOO-Kernel instance,
- **dataPath**: the path to store data files (when MapServer support was activated, this directory is used to store mapfiles and data).
- **tmpPath**: the path to store temporary files (such as ExecuteResponse when storeExecuteResponse was set to true),
- **tmpUrl**: a url relative to serverAddress to access the temporary file,
- **cachedir**: the path to store cached request files \(^1\) (optional),
- **mapservAddress**: your local MapServer address (optional),
- **msOgcVersion**: the version for all supported OGC Web Services output \(^2\) (optional).

The [identification] and [provider] section are specific to OGC metadata and should be set \(^3\).

Obviously, you are free to add new sections to this file if you need more \(^4\). Nevertheless, you have to know that there is some specific names you should use only for specific purposes: [headers], [mapserver], [env], [lenv] and [senv].

**Warning:** [senv] and [lenv] are used / produced on runtime internally by the ZOO-Kernel and should be defined only from the Service code.

The headers section is used to define your own HTTP Response headers. You may take a look at headers returned by web site such as http://www.zoo-project.org by using curl command line tool for instance and notice the specific header X-Powered-By: Zoo-Project@Trac.

**Warning:** There is no reason to define basic headers such as Content-Type or encoding as they will be overwritten at runtime by the ZOO-Kernel.

The mapserver section is used to store specific mapserver configuration parameters such as PROJ_LIB and GDAL_DATA or any other you want to be set to make your MapServer working.

**Note:** the mapserver section is mainly used on WIN32 platform

\(^1\) when you use GET requests passed through xlink:href the ZOO-Kernel will execute the request only once, the first time you will ask for this resource and it will store on disk the result. The next time you will need the same feature, the cached file will be used which make your process running faster. If cachedir was not specified in the main.cfg then tmpPath value will be used.

\(^2\) since version 1.3.0, when MapServer is activated, your service can automatically return a WMS, WFS or WCS request to expose your data. Your can set here the specific version number you want to use to request your local MapServer setup. It depends mostly on the client capability to deal with specific OGC Web Services version.

\(^3\) since version 1.3.0, when MapServer is activated, the same metadata will be used for setting metadata for OGC Web Services.

\(^4\) you can take a quick look into the mapmint main.cfg file which is available in /usr/lib/cgi-bin/mm/ directory to have example of sections use.

5.5. Deploying Web Processing Services using ZOO-Project – Examples of Python based WPS 113 using PgRouting
The **env** section is used to store specific environment variables you want to be set prior to load your Services Provider and run your Service. A typical example, is when your Service requires to access to a X server running on framebuffer, then you will have to set the `DISPLAY` environment variable, in this case you would add `DISPLAY=:1` line in your `[env]` section.

The **lenv** is used to store runtime informations automatically set by the ZOO-Kernel before running your service and can be accesses / updated from it:

- **sid (r):** the service unique identifier,
- **status (rw):** the current progress value (value between 0 and 100, percent),
- **cwd (r):** the current working directory of the ZOO-Kernel,
- **message (rw):** an error message when returning `SERVICE_FAILED` (optional),
- **cookie (rw):** the cookie your service want to return to the client (for authentication purpose or tracking).

The **senv** is used to store session informations on the server side. You can then access them automatically from service if the server is requested using a valid cookie (as defined in `lenv > cookie`). The ZOO-Kernel will store on disk the values set in the `senv` maps, then load it and dynamically add its content to the one available in the `main.cfg`. The `senv` section should contain at least:

- **XXX: the session unique identifier where XXX is the name included in the returned cookie.**

For instance, if you get the following in your Service source code:

```python
conf["lenv"]["cookie"]="XXX=XXX1000000; path=/"
conf["senv"]={"XXX": "XXX1000000","login": "demoUser"}
```

That means that the ZOO-Kernel will create a file `sess_XXX1000000.cfg` in the `cacheDir` and return the specified cookie to the client. Each time the client will request the ZOO-Kernel using the Cookie, it will automatically load the value stored before running your service. You can then easily access this informations from your service source code. This functionality won’t be used in the following presentation.

### Testing the ZOO installation with GetCapabilities

Once you have a `main.cfg` file available in the same directory as your ZOO-Kernel, then you can use `GetCapabilities`. Indeed, to answer such kind of requests, the ZOO-Kernel will simply parse the `main.cfg` file (to gather global informations), then parse individually each `zcfg` files (if any) contained in the same directory or in sub-directories, then return a well formed `Capabilities` document.

You can request ZOO-Kernel using the following link from your Internet browser:

```
http://localhost/cgi-bin/zoo_loader.cgi?Request=GetCapabilities&Service=WPS
```

You should get a valid Capabilities XML document, looking like the following:

---

5 If you’re not familiar with ZOO-Project, you can pass this part and come to it after the next section.

6 sub-directories listing is available from revision 469.
Please note that some Process nodes are returned in the ProcessOfferings section, as some are available already on OSGeoLive DVD. You can also run a GetCapabilities request from the command line, using the following command:

```bash
cd /usr/lib/cgi-bin
./zoo_loader.cgi "request=GetCapabilities&service=WPS" | less
```

The same result as in your browser will be returned, as shown in the following screenshot:

![ZOO Project Workshop - FOSS4G 2014](image)

Invoking ZOO Kernel from the command line can be helpful during development process of new Services for debugging purpose. If you need to simulate POST request from the command line, then you can use the following:

```bash
cd /usr/lib/cgi-bin
# Download the sample GetCapabilities request
curl -o /tmp/10_wpsGetCapabilities_request.xml http://schemas.opengis.net/wps/1.0.0/examples/10_wpsGetCapabilities_request.xml
# Define required environment settings
export REQUEST_METHOD=POST
```

5.5. Deploying Web Processing Services using ZOO-Project – Examples of Python based WPS using PgRouting
You should have the same result as presented before.

Creating your first ZOO Service

Table of Contents

- Creating your first ZOO Service
  - Introduction
  - Service and publication process overview
  - Creating your first ZCFG file
  - Test requests
    - Test the GetCapabilities request
    - Test the DescribeProcess request
    - Test the Execute request
  - Implementing the Python Service
    - General Principles
    - The Hello Service
  - Interacting with your service using Execute requests
  - Conclusion

Introduction

In this part, you will create and publish a simple ZOO-Service named Hello which will simply return a hello message containing the input value provided. It will be useful to present in deeper details general concept on how ZOO-Kernel works and handles Execute requests.

Service and publication process overview

Before starting developing a ZOO Service, you should remember that in ZOO-Project, a Service is a couple made of:

- a metadata file: a ZOO Service Configuration File (ZCFG) containing metadata informations about a Service (providing informations about default / supported inputs and outputs for a Service)
- a Services Provider: it depends on the programming language used, but for Python it is a module and for JavaScript a script file.

To publish your Service, which means make your ZOO Kernel aware of its presence, you should copy a ZCFG file in the directory where zoo_loader.cgi is located (in this workshop, /usr/lib/cgi-bin) or in any subdirectory.

**Warning:** only the ZCFG file is required for the Service to be considerate as available. So if you don’t get the Service Provider, obviously your Execute request will fail as we will discuss later.

Before publication, you should store your ongoing work, so you’ll start by creating a directory to store the files of your Services Provider:

mkdir -p /home/user/zoo-ws/ws_sp/cgi-env
Once the ZCFG and the Python module are both ready, you can publish simply by copying the corresponding files in the same directory as the ZOO-Kernel.

**Creating your first ZCFG file**

You will start by creating the ZCFG file for the Hello Service. Edit the `/home/user/zoo-ws/ws_sp/cgi-env/Hello.zcfg` file and add the following content:

```xml
[Hello]
Title = Return a hello message.
Abstract = Create a welcome string.
processVersion = 2
storeSupported = true
statusSupported = true
serviceProvider = test_service
serviceType = Python
<DataInputs>
  [name]
  Title = Input string
  Abstract = The string to insert in the hello message.
  minOccurs = 1
  maxOccurs = 1
  <LiteralData>
    dataType = string
    <Default />
  </LiteralData>
</DataInputs>
<DataOutputs>
  [Result]
  Title = The resulting string
  Abstract = The hello message containing the input string
  <LiteralData>
    dataType = string
    <Default />
  </LiteralData>
</DataOutputs>
```

**Note:** the name of the ZCFG file and the name between bracket (here [Hello]) should be the same and correspond to the function name you will define in your Services provider.

As you can see in the ZOO Service Configuration File presented above it is divided into three distinct sections:

1. Main Metadata information (from line 2 to 8)
2. List of Inputs metadata information (from 9 line to 19)
3. List of Outputs metadata information (from line 20 to 28)

You can get more informations about ZCFG from the reference documentation.

If you copy the Hello.zcfg file in the same directory as your ZOO Kernel then you will be able to request for DescribeProcess using the Hello Identifier. The Hello service should also be listed from Capabilities document.
Test requests

In this section you will tests each WPS requests : GetCapabilities, DescribeProcess and Execute. Note that only GetCapabilities and DescribeProcess should work at this step.

Test the GetCapabilities request  If you run the GetCapabilities request:

http://localhost/cgi-bin/zoo_loader.cgi?request=GetCapabilities&service=WPS

Now, you should find your Hello Service in a Process node in ProcessOfferings:

```xml
<wps:Process wps:processVersion="2">
    <ows:Identifier>Hello</ows:Identifier>
    <ows:Title>Return a hello message.</ows:Title>
    <ows:Abstract>Create a welcome string.</ows:Abstract>
</wps:Process>
```

Test the DescribeProcess request  You can access the ProcessDescription of the Hello service using the following DescribeProcess request:

http://localhost/cgi-bin/zoo_loader.cgi?request=DescribeProcess&service=WPS&version=1.0.0&Identifier=Hello

You should get the following response:

```xml
    <ProcessDescription wps:processVersion="2" storeSupported="true" statusSupported="true">
        <ows:Identifier>Hello</ows:Identifier>
        <ows:Title>Return a hello message.</ows:Title>
        <ows:Abstract>Create a welcome string.</ows:Abstract>
        <DataInputs>
            <Input minOccurs="1" maxOccurs="1">
                <ows:Identifier>name</ows:Identifier>
                <ows:Title>Input string</ows:Title>
                <ows:Abstract>The string to insert in the hello message.</ows:Abstract>
                <LiteralData>
                    <ows:DataType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:ows="http://www.opengis.net/ows/1.1" xsi:schemaLocation="http://schemas.opengis.net/ows/1.1/owsInputLiteralData_common.xsd">string</ows:DataType>
                    <ows:AnyValue/>
                </LiteralData>
            </Input>
        </DataInputs>
        <ProcessOutputs>
            <Output>
                <ows:Identifier>Result</ows:Identifier>
                <ows:Title>The resulting string</ows:Title>
                <ows:Abstract>The hello message containing the input string</ows:Abstract>
                <LiteralOutput>
                    <ows:DataType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:ows="http://www.opengis.net/ows/1.1" xsi:schemaLocation="http://schemas.opengis.net/ows/1.1/owsOutputLiteralData_common.xsd">string</ows:DataType>
                    <ows:AnyValue/>
                </LiteralOutput>
            </Output>
        </ProcessOutputs>
    </ProcessDescription>
</wps:ProcessDescriptions>
```

Test the Execute request  Obviously, you cannot run your Service because the Python file was not published yet. If you try the following Execute request:
You should get an ExceptionReport similar to the one provided in the following, which is normal behavior:

```xml
  <ows:Exception exceptionCode="NoApplicableCode">
    <ows:ExceptionText>Python module test_service cannot be loaded.</ows:ExceptionText>
  </ows:Exception>
</ows:ExceptionReport>
```

Implementing the Python Service

**General Principles**  The most important thing you must know when implementing a new ZOO-Services using the Python language is that the function corresponding to your Service returns an integer value representing the status of execution (SERVICE_FAILED\(^7\) or SERVICE_SUCCEEDED\(^8\)) and takes three arguments (Python dictionaries):

- `conf`: the main environment configuration (corresponding to the `main.cfg` content)
- `inputs`: the requested / default inputs (used to access input values)
- `outputs`: the requested / default outputs (used to store computation result)

**Note:** when your service return SERVICE_FAILED you can set `conf["lenv"]["message"]` to add a personalized message in the ExceptionReport returned by the ZOO Kernel in such case.

You get in the following a sample `conf` value based on the `main.cfg` file you saw before.

```json
{
  "main": {
    "language": "en-US",
    "lang": "fr-FR,ja-JP",
    "version": "1.0.0",
    "encoding": "utf-8",
    "serverAddress": "http://localhost/cgi-bin/zoo_loader.cgi",
    "dataPath": "/var/data",
    "tmpPath": "/var/www/temp",
    "tmpUrl": "./.temp",
    "cacheDir": "/var/www/temp/"
  },
  "identification": {
    "title": "ZOO-Project Workshop - FOSS4G 2014",
    "keywords": "WPS,GIS,buffer",
    "abstract": "Deploying Web Processing Services using ZOO-Project - Examples of Python based WPS using PgRouting",
    "accessConstraints": "none",
    "fees": "None"
  },
  "provider": {
    "positionName": "Developer",
    "providerName": "ZOO-Project",
    "addressAdministrativeArea": "Lattes",
    "addressCountry": "fr",
    "phoneVoice": "False",
    "addressPostalCode": "34970",
    "role": "Dev"
  }
}
```

\(^7\) SERVICE_FAILED=4  \(^8\) SERVICE_SUCCEEDED=3

5.5. Deploying Web Processing Services using ZOO-Project – Examples of Python based WPS using PgRouting
In the following you get a sample outputs value passed to a Python or a JavaScript Service:

```json
{
  'Result': {
    'mimeType': 'application/json',
    'inRequest': 'true',
    'encoding': 'UTF-8'
  }
}
```

**Note:** the inRequest value is set internally by the ZOO-Kernel and can be used to determine from the Service if the key was provided in the request.

ZOO-Project provide a ZOO-API which was originally only available for JavaScript services, but thanks to the work of the ZOO-Project community, now you have also access to a ZOO-API when using the Python language. Thanks to the Python ZOO-API you don’t have to remember anymore the value of SERVICE_SUCCEEDED and SERVICE_FAILED, you have the capability to translate any string from your Python service by calling the `_` function (ie: `zoo._('My string to translate')`) or to update the current status of a running service by using the `update_status` function the same way you use it from JavaScript or C services.

### The Hello Service

You can copy and paste the following into the `/home/user/zoo-ws/ws_sp/cgi-env/test_service.py` file.

```python
import zoo

def Hello(conf, inputs, outputs):
    outputs['Result']['value'] = "Hello " + inputs['name']['value'] + " from the ZOO-Project Python world !"
    return zoo.SERVICE_SUCCEEDED
```

Once you finish editing the file, you should copy it in the `/usr/lib/cgi-bin` directory:

```bash
sudo cp /home/user/zoo-ws/ws_sp/cgi-env/* /usr/lib/cgi-bin
```

### Interacting with your service using Execute requests

Now, you can request for Execute using the following basic url:

```bash
http://localhost/cgi-bin/zoo_loader.cgi?request=Execute&service=WPS&version=1.0.0&Identifier=Hello&DataInputs=name=toto
```

You can request the WPS Server to return a XML WPS Response containing the result of your computation, requesting for ResponseDocument or you can access the data directly requesting for RawDataOutput.

- Sample request using the RawDataOutput parameter:

  ```bash
  http://localhost/cgi-bin/zoo_loader.cgi?request=Execute&service=WPS&version=1.0.0&Identifier=Hello&DataInputs=name=toto&RawDataOutput=Result
  ```

- Sample request using the default ResponseDocument parameter:

  ```bash
  http://localhost/cgi-bin/zoo_loader.cgi?request=Execute&service=WPS&version=1.0.0&Identifier=Hello&DataInputs=name=toto
  ```

---

9 sample use of update_status is available [here](#)
When you are using ResponseDocument there is specific attribute you can use to ask the ZOO Kernel to store the result: 

```
http://localhost/cgi-bin/zoo_loader.cgi?request=Execute&service=WPS&version=1.0.0&Identifier=Hello&DataInputs=name=toto&ResponseDocument=Result@asReference=true
```

When computation take long time, the client should request the execution of a Service by setting both 
```
storeExecuteResponse
status
```

parameter to true to force asynchronous execution. This will make the ZOO-Kernel return, without waiting for the Service execution completion but after starting another ZOO-Kernel process responsible of the Service execution, a ResponseDocument containing a 
```
statusLocation
```

attribute which can be used to access the status of an ongoing service or the result when the process ended. 

```
http://localhost/cgi-bin/zoo_loader.cgi?request=Execute&service=WPS&version=1.0.0&Identifier=Hello&DataInputs=name=toto&ResponseDocument=Result&storeExecuteResponse=true&status=true
```

**Conclusion**

Even if this first service was really simple it was useful to illustrate how the ZOO-Kernel fill 
```
conf, inputs and outputs
```

parameter prior to load and run your function service, how to write a ZCFG file, how to publish a Services Provider by placing the ZCFG and Python files in the same directory as the ZOO-Kernel, then how to interact with your service using both 
```
GetCapabilities, DescribeProcess and Execute
```

requests. We will see in the next section how to write similar requests using the XML syntax.

**Building blocks presentation - Using OGR and PgRouting based Web Services**

<table>
<thead>
<tr>
<th>Table of Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Building blocks presentation - Using OGR and PgRouting based Web Services</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

**Introduction**

In this section, you will use basic ZOO-Services: Buffer, Intersection and DifferencePy which use OGR and psycopg Python modules. The intended goal of this section is to present and interact with your new building blocks before chaining them in the next section.

First of all, you should use the following links to access the user interfaces and interact with your services, the first one is used to access basic spatial-tools demo client interface and the second to access the routing application:

- [http://localhost/zoo-ws/spatialtools-py.html](http://localhost/zoo-ws/spatialtools-py.html)
- [http://localhost/zoo-ws/](http://localhost/zoo-ws/)

To get on-going status [url in statusLocation](http://localhost/zoo-ws/status), you’ll need to setup the utils/status service. If you don’t get this service available, the ZOO-Kernel will simply give the url to a flat XML file stored on the server which will contain, at the end of the execution, the result of the Service execution. For more informations please take a look into the reference [documentation](http://localhost/zoo-ws/status).

5.5. Deploying Web Processing Services using ZOO-Project – Examples of Python based WPS 121 using PgRouting
Services Provider and configuration files

First you may verify if the ZOO-Services are available from your current setup. You can take a look at the Buffer.zcfg, Intersection.zcfg and DifferencePy.zcfg to get details about parameters. As you can see from the ZCFG files, you will use ZOO-Services provided by the service Python service provider. So if you want to modify the Python code you will have to edit the corresponding file (so service.py). You are invited to use similar requests as the one used in previous sections to learn about each services individually.

The Buffer Service

First click on a street then once the street is displayed in blue, click the ‘Buffer’ button on top, you should get similar result as displayed in the following.

Since version ZOO-Project 1.2.0, you can run automatically some basic tests to make sure that you wrote a correct ZCFG file and your service is validating.

Note: the current testing is really simple and should be adapted to each Services Provider, mainly to define input names.

You can use the following command:

cd /home/user/zoo/testing
./run.sh http://localhost/cgi-bin/zoo_loader.cgi Buffer

Note: During or after the test run, you can take a look inside the tmp directory which contains both the XML requests send to the ZOO Kernel (*.xml) and the responses it gave (output*.xml).
The Intersection Service

Using the same client interface as before, once you get a Buffer, you can then select a street intersecting the Buffer geometry to compute intersection by clicking on the Intersection button.

The DifferencePy Service

Using the same instructions as for Intersection, you can get the following result.
The Routing and Profile Services

First click on the blue flag then place your starting point on the map, do the same with the red flag to get the shortest path computed by the dedicated service and then display its profile. Note that when you pass the mouse over the profile display then you will see its corresponding position displayed on the map. You can also click on a specific step in your path to get the corresponding line displayed.

If you take a look in the file: /usr/lib/cgi-bin/routing/do.zcfg you may notice something new in the supported format which is reproduced here after.

```xml
<Supported>
  mimeType = image/png
  asReference = true
  msStyle = STYLE COLOR 125 0 105 OUTLINECOLOR 0 0 0 WIDTH 3 END
  useMapServer = true
  extension = json
</Supported>
```

The `mimeType` is defined as `image/png` and there is two new parameter which are both optional:

- `useMapServer`: which make you able to inform the ZOO-Kernel that it have to use MapServer to publish your result as WMS / WFS or WCS (this last won’t be used in this workshop) rather than simply storing the result as a file.

- `msStyle`: which let you define your own MapServer STYLE block definition.

When you need to access a result many time or for different purpose accross other services then it is really useful to ask ZOO-Kernel to publish your result as WMS, WFS or WCS.

Note that no modification of the code of the do service was required to handle automatic pubilcation of the result as it is a vector format supported by OGR, only modification of the zcfg was required.

This routing example is here to illustrate how easy it is to publish your result as WMS, WFS or WCS ressource. Indeed, when your routing service was called, ZOO-Kernel published the result as WMS and WFS ressources which was both used first for the UI purpose, the WMS to display the resulting path on the map, the WFS for displaying details about
each steps. The WFS protocol is also used as the input value for the profile computation. So, the computation was run once and accessed three times for different purposes and from different clients.

For more informations about the MapServer support, please refer to the official ZOO-Project Documentation.

**Conclusion**

Now you know this three services, and you get a simple interface to interact with your MapServer WFS and your ZOO-Project WPS Servers, you are ready to use the Services in a different way, by chaining them using the JavaScript ZOO-API to build more complexe and powerfull services.

**Playing with buildign blocks - Creating JavaScript Web Services**

<table>
<thead>
<tr>
<th>Table of Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Playing with buildign blocks - Creating JavaScript Web Services</td>
</tr>
<tr>
<td>- Introduction</td>
</tr>
<tr>
<td>- ZOO-API Overview</td>
</tr>
<tr>
<td>- The Mask Service</td>
</tr>
<tr>
<td>* The ZCFG</td>
</tr>
<tr>
<td>* The JavaScript service</td>
</tr>
<tr>
<td>* Publish and use your Service</td>
</tr>
<tr>
<td>- BufferMask Service</td>
</tr>
<tr>
<td>* The ZCFG</td>
</tr>
<tr>
<td>* The JavaScript service</td>
</tr>
<tr>
<td>* Publish and use your Service</td>
</tr>
<tr>
<td>- BufferRequest service</td>
</tr>
<tr>
<td>* The ZCFG</td>
</tr>
<tr>
<td>* The JavaScript code</td>
</tr>
<tr>
<td>* Publish and use your Service</td>
</tr>
<tr>
<td>- Add Union into the chain</td>
</tr>
<tr>
<td>- Conclusion</td>
</tr>
</tbody>
</table>

**Introduction**

This section illustrate how you can use JavaScript on the server-side to chain services together to build new ones. You will create a ZOO Services Provider using the services you seen before and the WFS server using the ZOO-API. The final goal is to query all POIs included in a buffer around a feature and to highlight them using a mask around this buffer. The following screenshot show you the expected result:
For the routing interface result should look like this:

You can decompose the result above in two different ones: the mask around the buffer and the points included in the buffer. So you will create two different Services: one called `BufferMask` and another one called `BufferRequest`.

But before implementing any JavaScript Service, you will get an overview of how to use ZOO-API from your ZOO-Project installation in the next section.

As before, you first create a new directory to store files for your new Services Provider:
ZOO-API Overview

ZOO-API and ZOO-Kernel JavaScript support make you able to run services implemented in JavaScript on the server side. JavaScript is a popular programming language but mostly used on the client side. Let say from a browser, but here it is a bit different.

To support JavaScript language ZOO-Kernel use the SpiderMonkey API to create a javascript runtime environment from which it will load your JS file then extract the function corresponding to the service to run it using the prefilled parameters. The JavaScript runtime environment created by the ZOO-Kernel depend on your setup. If you placed the ZOO-api.js and ZOO-proj4js.js in the same directory as your ZOO-Kernel it means that your environment will contains ZOO-API and Proj4js which will be loaded before your service. In such case you can access to the Classes defined in the JavaScript ZOO-API to manipulate geographic data, for more informations please refer to the ZOO-API Documentation.

Even if it can be useful to run JavaScript on the server side, you should remember that some basic JavaScript functions you are familiar with does not exist or get a different behavior. For instance the simple alert function will display messages in apache error logs rather than in a window when used from a browser. The alert function can be used as follow:

```javascript
alert("My alert message");
```

There is no XMLHttpRequest available in the JavaScript environment your service will run into. Hopefully, the ZOO-Kernel expose a C function to the JavaScript world named: JSRequest. This function make you able from your JavaScript services to call other WPS services (locally or remotely) or other kind OGC services such as WFS. When you are using the ZOO-API it is possible to call Services using a ZOO.Process instance, to parse WPS Responses using ZOO.Format.WPS (cf. ref).

As for Python services you already seen in previous sections, the functions corresponding to a Service should take three arguments: conf, inputs and outputs. Nevertheless, as the ZOO-Kernel is not able to access the values modified by the Service code, rather than returning an integer as in Python, here you’ll need to return both the integer value representing the Status of your Service in a JavaScript Object and the resulting outputs values as an Object. You can see in the following an example of a JavaScript Service code:

```javascript
function SampleService(conf,inputs,outputs){
    var resultValue=someComputation(inputs);
    return
    {
        result: ZOO.SERVICE_SUCCEEDED,
        outputs: { "Result": { "mimeType": "application/json", "value": resultValue } }
    };
}
```

Before starting to implement the Services we will need to get our final BufferRequest service, let start with a simpler one.

The Mask Service

In this section you will learn how to create your first JavaScript service which will simply return a rectangular mask around a selected feature. To build this mask you will use the Buffer service to create a buffer big enough around a

---

11 The ZOO.Process class uses JSRequest (cf. ref). You will get example of use later.
12 So conf, inputs and outputs are simple JavaScript objects, similar to the Python dictionaries used in the previous section.
13 Such as conf, inputs and outputs.
14 You can also return a conf Object if you get any informations updated from your JavaScript service (such as cookie for instance)
selected geometry to cover a significant part of your map. You can see the expected result in the following screenshot:

![Map Screenshot]

As before, you will first start by writing the ZCFG, then you will write the JavaScript source code and finish by publishing your Services Provider.

**The ZCFG**  Open a file named `~/zoo-ws/jschains/cgi-env/Mask.zcfg` with your favorite text editor and add the following content:

```xml
<Mask>
  Title = Compute mask
  Abstract = Compute mask around a geometry
  processVersion = 1
  storeSupported = true
  statusSupported = true
  serviceProvider = foss4gws.js
  serviceType = JS
  <DataInputs>
    [InputData]
    Title = The feature
    Abstract = The feature to run the service with
    minOccurs = 1
    maxOccurs = 1
    <ComplexData>
      <Default>
        mimeType = text/xml
        encoding = utf-8
      </Default>
    </ComplexData>
  </DataInputs>
  <DataOutputs>
    [Result]
    Title = The resulting feature
    Abstract = The feature created by the service.
    <ComplexOutput>
      <Default>
      </Default>
    </ComplexOutput>
  </DataOutputs>
</Mask>
```
Here you simply define one default ComplexData for both inputData and Result: a GML and a GeoJSON respectively.

The JavaScript service As you will have to request the Buffer service many times from your service, you will first define a Buffer function as follow. It uses the ZOO.Process to request the Buffer service you seen in the previous section.

Open a file named ~/.zoo-ws/jschains/cgi-env/foss4gws.js and add the following content:

```javascript
var zoo_url='http://localhost/cgi-bin/zoo_loader.cgi';
var mapfile="/var/data/maps/project_WS2014.map";
var mapserv_url="http://localhost/cgi-bin/mapserv?map=mapfile";

function Buffer(inputData,bDist){
    // Create all required ZOO.formats
    var fJ=new ZOO.Format.JSON();
    var fGJ=new ZOO.Format.GeoJSON();
    var fWPS=new ZOO.Format.WPS();
    
    // Pass the value as json
    var myInputs = {
        InputPolygon: { type: 'complex', value: fGJ.write(inputData), mimeType: "application/json"},
        BufferDistance: {type: 'float', "value": bDist }
    };
    var myOutputs= { Result: { type: 'RawDataOutput', "mimeType": "application/json" } };
    var myProcess = new ZOO.Process(zoo_url,'BufferPy');
    var myExecuteResult=myProcess.Execute(myInputs,myOutputs);
    return fGJ.read(myExecuteResult);
}
```

From line 12 to 16, you give a GeoJSON string (created from inputData) for InputPolygon and, on line 15, you set the BufferDistance value to bDist. On line 17, you define Result as a RawDataOutput, so you won’t have to parse the WPS response using the ZOO.Format.WPS.

On line 18, you create a ZOO.Process instance providing the ZOO-Kernel url and the Service name. Then, on line 19, you run the request passing inputs and outputs previously defined (from line 12 to 15). On line 21 you return the GeoJSON red.

Now, you get your Buffer function, it is time to create your first JavaScript service. So, edit your foss4gws.js file you created before and add the following content:

```javascript
function Mask(conf,inputs,outputs){
    // Create all required ZOO.formats
    var fGML=new ZOO.Format.GML();
    var fGJ=new ZOO.Format.GeoJSON();
    
    // Read the input GML
```

15 Using one of the available ZOO.formats you are also able to support various ComplexData for both input and output of the service. To simplify the presentation here, you will use only this default ones.

5.5. Deploying Web Processing Services using ZOO-Project – Examples of Python based WPS using PgRouting
```javascript
var inputData = fGML.read(inputs["InputData"]['value']);

// Compute Buffer
var bufferResultAsJSON = Buffer(inputData, 0.015);

// Create the Buffer result BBOX and store its geometry in a ZOO.Feature
var bbox = new ZOO.Bounds();
var bounds = bufferResultAsJSON[0].geometry.getVertices();
for (var t in bounds) {
    bbox.extend(bounds[t]);
}
var finalG = bbox.toGeometry();
var result = new ZOO.Feature(finalG, { "name": "Result1000" });

// Return the created feature
return {
    result: ZOO.SERVICE_SUCCEEDED,
    outputs: { "Result": { mimeType: "application/json", value: fGJ.write(result) } }
};
```

**Publish and use your Service**  
Now you get both your ZCFG and your service code ready, you need to deploy your new Services Provider using the following command:

```
sudo cp ~/zoo-ws/jschains/cgi-env/* /usr/lib/cgi-bin
```

Now you are ready to use your JavaScript service by loading the following url, click on a street then click on the “Mask” button.

**BufferMask Service**

In this section you will implement a simple JavaScript service which will be able create a hole in the mask you created in previous section. This service will be used to highlight the buffer zone around a selected feature. You get a preview of the expected result in the following screenshot:
The ZCFG  Open the file named `~/zoo-ws/jscchains/cgi-env/BufferMask.zcfg` with your favorite text editor and copy / paste the following content:

```xml
[BufferMask]
  Title = Compute buffer mask
  Abstract = Compute buffer mask around a geometry
  processVersion = 1
  storeSupported = true
  statusSupported = true
  serviceProvider = foss4gws.js
  serviceType = JS
<DataInputs>
  [InputData]
    Title = The feature
    Abstract = The feature to run the service with
    minOccurs = 1
    maxOccurs = 1
    <ComplexData>
      <Default>
        mimeType = text/xml
        encoding = utf-8
      </Default>
    </ComplexData>
  </DataInputs>
<DataOutputs>
  [Result]
    Title = The resulting feature
    Abstract = The feature created by the service.
    <ComplexOutput>
      <Default>
        mimeType = application/json
      </Default>
    </ComplexOutput>
  </DataOutputs>
```

5.5. Deploying Web Processing Services using ZOO-Project – Examples of Python based WPS 131 using PgRouting
This ZCFG is similar to the previous one. Please, refer to comments in the previous section for more informations.

**The JavaScript service**  In this Service you will use same source code (until line 19) you used in the previous section. Indeed, you should compute the Mask as you did before then compute Buffer for creating a hole in the mask (on line 22) to run the Difference service (from line 25 to 40).

```javascript
function BufferMask(conf, inputs, outputs) {

    // Create all required ZOO.formats
    var fGML = new ZOO.Format.GML();
    var fGJ = new ZOO.Format.GeoJSON();

    // Read the input GML
    var inputData = fGML.read(inputs["InputData"]['value']);

    // Compute Buffer
    var bufferResultAsJSON = Buffer(inputData, 0.015);

    // Create the Buffer result BBOX
    var bbox = new ZOO.Bounds();
    var bounds = bufferResultAsJSON[0].geometry.getVertices();
    for (var t in bounds) {
        bbox.extend(bounds[t]);
    }
    var finalG = bbox.toGeometry();

    // Compute Buffer standard buffer
    var bufferResultAsJSON = Buffer(inputData, 0.0015);

    // Request Difference service using Buffer result and features in the BBOX
    var result = new ZOO.Feature(finalG, {"name": "Result1000"});
    var myProcess2 = new ZOO.Process(zoo_url, 'DifferencePy');
    var myInputs2 = {
        InputEntity1: {
            type: 'complex',
            value: fGJ.write(finalG),
            mimeType: "application/json"
        },
        InputEntity2: {
            type: 'complex',
            value: fGJ.write(bufferResultAsJSON),
            mimeType: "application/json"
        }
    };
    var myOutputs2 = {Result: {type: 'RawDataOutput', mimeType: "application/json"}};
    var myExecuteResult4 = myProcess2.Execute(myInputs2, myOutputs2);

    // Return the bbox
    var result = new ZOO.Feature(finalG, {"name": "Result1000"});
    return {
        result: ZOO.SERVICE_SUCCEEDED,
        outputs: { "Result": {mimeType: "application/json", value: myExecuteResult4 } }
    };
}
```
Publish and use your Service  
Now, you can publish your service as you did before. To use your service, please use the following url.

BufferRequest service

In this section, you will create a new Service: BufferRequest which will request POIs included in the Buffer around a selected feature. You will use the poi layer served as WFS through your local mapserver installation. You can see in the following screenshot the expected result:

The ZCFG  
Open the file named `~/zoo-ws/jscchains/cgi-env/BufferRequest.zcfg` with your favorite text editor and copy / paste the following content:

```
[BufferRequest]
Title = Compute buffer request
Abstract = Compute buffer request around a geometry
processVersion = 1
storeSupported = true
statusSupported = true
serviceProvider = foss4gws.js
serviceType = JS
<DataInputs>
[InputData]
Title = The feature
Abstract = The feature to run the service with
minOccurs = 1
maxOccurs = 1
<ComplexData>
<Default>
mimeType = text/xml
encoding = utf-8
</Default>
```

16 So in the hole you created in the previous section.

5.5. Deploying Web Processing Services using ZOO-Project – Examples of Python based WPS using PgRouting
As in the previous Service, you will compute a buffer around the input feature. But then you will request POIs available in the Buffer extent using a WFS request to use them to run the Intersection service using the initial Buffer. The WFS request is useful to limit the number of points to use when requesting the Intersection Service.

```javascript
function BufferRequest(conf, inputs, outputs) {
    // Create all required ZOO.formats
    var fGJ = new ZOO.Format.GeoJSON();
    var fGML = new ZOO.Format.GML();

    // Read the input GML
    var inputData = fGML.read(inputs["InputData"]['value']);

    // Compute Buffer
    var bufferResultAsJSON = Buffer(inputData, 0.0015);

    // Create the Buffer result BBOX
    var bbox = new ZOO.Bounds();
    var bounds = bufferResultAsJSON[0].geometry.getVertices();
    for (var t in bounds) {
        bbox.extend(bounds[t]);
    }

    // Request Intersection service using Buffer result and WFS request using the BBOX
    var myProcess2 = new ZOO.Process(zoo_url, 'Intersection');
    var req = "&SERVICE=WFS&version=1.0.0&request=GetFeature&typeName=poi1&
               SRS=EPSG:4326&BBOX=" + bbox.left +"," + bbox.bottom +"," + bbox.right +"," + bbox.top + "";
    var myInputs2 = {
        InputEntity1: {
            type: 'complex',
            value: fGJ.write(bufferResultAsJSON),
            mimeType: "application/json"
        },
        InputEntity2: {
            type: 'complex',
            xlink: mapserv_url + req + bbox.left +"," + bbox.bottom +"," + bbox.right +"," + bbox.top + ",
            mimeType: "text/xml"
        }
    };

    var myOutputs2 = {Result: {type: 'RawDataOutput', "mimeType": "application/json"}};
    var myExecuteResult4 = myProcess2.Execute(myInputs2, myOutputs2);
```
return {
    result: ZOO.SERVICE_SUCCEEDED,
    outputs: [ {name:"Result", mimeType: "application/json", value: myExecuteResult4} ]
};

**Warning:** to take advantage of the ZOO-Kernel cache system, you directly use the WFS request as xlink:href rather than value for InputEntity2 (from line 31 to 34) and use text/xml mimeType (on line 40). Indeed, the ZOO-API doesn’t use the internal cache mechanisms.

**Publish and use your Service**   Now, you can publish your service as you did before. To use your service, please use the following url.

**Note:** You can click on “Buffer Request and Mask” to get the same result as presented in the initial screenshot.

**Add Union into the chain**

As you can see in the following screenshot, when using the Buffer service using a feature collection containing more than one geometry, the result is made of multiple geometries. So, running Buffer service on the routing interface will result in multiple buffer:

![Buffer Request and Mask](image)

So, to get the same result as you got when selecting a single road, you should use Union of geometry (input or the one outputed by the Buffer Service). As you are using the JavaScript ZOO-API, you can simply update the Buffer JavaScript function you defined earlier, to first call the Union of each geometry avaliable in a feature collection prior to request (or after requesting) the Buffer Service. Hopefully, there is already this Python Service available, its name is UnionOne1, so you just need to add it in your Service chain.

Here is the final code for the Buffer JavaScript function:

```javascript
// Code continues...
```
function Buffer(inputData, bDist) {

    // Create all required ZOO.formats
    var fJ = new ZOO.Format.JSON();
    var fGJ = new ZOO.Format.GeoJSON();
    var fWPS = new ZOO.Format.WPS();

    // Call the UnionOnel Service
    var myInputs0 = {
        InputPolygon: { type: 'complex', value: fGJ.write(inputData), mimeType: "application/json" },
        BufferDistance: { type: 'float', "value": bDist }
    };
    var myOutputs0 = { Result: { type: 'RawDataOutput', "mimeType": "application/json" } };  
    var myProcess0 = new ZOO.Process(zoo_url, 'UnionOnel');
    var myExecuteResult0 = myProcess0.Execute(myInputs0, myOutputs0);

    // Call the BufferPy Service
    var myInputs = {
        InputPolygon: { type: 'complex', value: myExecuteResult0, mimeType: "application/json" },
        BufferDistance: { type: 'float', "value": bDist }
    };
    var myOutputs = { Result: { type: 'RawDataOutput', "mimeType": "application/json" } };  
    var myProcess = new ZOO.Process(zoo_url, 'BufferPy');
    var myExecuteResult = myProcess.Execute(myInputs, myOutputs);

    return fGJ.read(myExecuteResult);
}

Conclusion

After understanding how basic Geometric Operation Services works, here you built step by step new JavaScript services which reuse the previous ones and combine them in different ways. This was achieved using the ZOO-API, composed by C functions exposed by the ZOO-Kernel to the JavaScript services runtime environement and the JS files which can be optionally installed.

5.6 ZOO Community

The following sections will help you interact with the ZOO community.

5.6.1 ZOO-Project Committer Guildlines

Authors  Nicolas Bozon, Gérald Fenoy, Jeff McKenna

Last Updated  $Date: 2015-02-09 16:26:31 +0100 (lun. 09 févr. 2015) $
Election to SVN Commit Access

Permission for SVN commit access shall be provided to new developers only if accepted by the ZOO-Project Project Steering Committee. A proposal should be written to the PSC for new committers and voted.

Removal of SVN commit access should be handled by the same process.

The new committer should have demonstrated commitment to ZOO-Project and knowledge of the ZOO-Project source code and processes to the committee’s satisfaction, usually by reporting bugs, submitting patches, and/or actively participating in the ZOO-Project mailing list(s).

The new committer should also be prepared to support any new feature or changes that he/she commits to the ZOO-Project source tree in future releases, or to find someone to which to delegate responsibility for them if he/she stops being available to support the portions of code that he/she is responsible for.

All committers should also be a member of the zoo-discuss mailing list so they can stay informed on policies, technical developments and release preparation.

New committers are responsible for having read, and understood this document.

Committer Tracking

A list of all project committers will be kept in the main zoo-project directory (called COMMITTERS) listing for each SVN committer:

- Userid: the id that will appear in the SVN logs for this person.
- Full name: the user's actual name.
- Email address: A current email address at which the committer can be reached. It may be altered in normal ways to make it harder to auto-harvest.

SVN Administrator

One member of the Project Steering Committee will be designated the SVN Administrator. That person will be responsible for giving SVN commit access to folks, updating the COMMITTERS file, and other SVN related management. That person will need login access on the SVN server of course.

Initially Gérald Fenoy will be the SVN Administrator.

SVN Commit Practices

The following are considered good SVN commit practices for the ZOO-Project project.

- Use meaningful descriptions for SVN commit log entries.
• Add a bug reference like “(#1234)” at the end of SVN commit log entries when committing changes related to a ticket in Trac. The ‘#’ character enables Trac to create a hyperlink from the changeset to the mentioned ticket.

• After committing changes related to a ticket in Trac, write the tree and revision in which it was fixed in the ticket description. Such as “Fixed in trunk (r12345) and in branches/1.7 (r12346)”. The ‘r’ character enables Trac to create a hyperlink from the ticket to the changeset.

• Changes should not be committed in stable branches without a corresponding bug id. Any change worth pushing into the stable version is worth a bug entry.

• Never commit new features to a stable branch without permission of the PSC or release manager. Normally only fixes should go into stable branches.

• New features go in the main development trunk.

• Only bug fixes should be committed to the code during pre-release code freeze, without permission from the PSC or release manager.

• Significant changes to the main development version should be discussed on the zoo-discuss list before you make them, and larger changes will require a to be discussed and approved on zoo-psc list by the PSC.

• Do not create new branches without the approval of the PSC. Release managers are assumed to have permission to create a branch.

• All source code in SVN should be in Unix text format as opposed to DOS text mode.

• When committing new features or significant changes to existing source code, the committer should take reasonable measures to insure that the source code continues to build and work on the most commonly supported platforms (currently Linux and Windows), either by testing on those platforms directly, running Buildbot tests, or by getting help from other developers working on those platforms. If new files or library dependencies are added, then the configure.in, Makefile.in, Makefile.vc and related documentations should be kept up to date.

Legal

Committers are the front line gatekeepers to keep the code base clear of improperly contributed code. It is important to the ZOO-Project users, developers and the OSGeo foundation to avoid contributing any code to the project without it being clearly licensed under the project license.

Generally speaking the key issues are that those providing code to be included in the repository understand that the code will be released under the MIT/X license, and that the person providing the code has the right to contribute the code. For the committer themselves understanding about the license is hopefully clear. For other contributors, the committer should verify the understanding unless the committer is very comfortable that the contributor understands the license (for instance frequent contributors).

If the contribution was developed on behalf of an employer (on work time, as part of a work project, etc) then it is important that an appropriate representative of the employer understand that the code will be contributed under the MIT/X license. The arrangement should be cleared with an authorized supervisor/manager, etc.

The code should be developed by the contributor, or the code should be from a source which can be rightfully contributed such as from the public domain, or from an open source project under a compatible license.

All unusual situations need to be discussed and/or documented.

Committers should adhere to the following guidelines, and may be personally legally liable for improperly contributing code to the source repository:

• Make sure the contributor (and possibly employer) is aware of the contribution terms.

• Code coming from a source other than the contributor (such as adapted from another project) should be clearly marked as to the original source, copyright holders, license terms and so forth. This information can be in the file
headers, but should also be added to the project licensing file if not exactly matching normal project licensing (zoo-project/zoo-kernel/LICENSE).

- Existing copyright headers and license text should never be stripped from a file. If a copyright holder wishes to give up copyright they must do so in writing to the foundation before copyright messages are removed. If license terms are changed it has to be by agreement (written in email is ok) of the copyright holders.

- Code with licenses requiring credit, or disclosure to users should be added to /trunk/zoo-project/zoo-kernel/LICENSE.

- When substantial contributions are added to a file (such as substantial patches) the author/contributor should be added to the list of copyright holders for the file.

- If there is uncertainty about whether a change it proper to contribute to the code base, please seek more information from the project steering committee, or the foundation legal counsel.

**Bootstrapping**

The following existing committers will be considered authorized ZOO-Project committers as long as they each review the committer guidelines, and agree to adhere to them. The SVN administrator will be responsible for checking with each person.

- David SAGGIORATO (aka david)
- Gérald FENOY (aka djay)
- Jeff MCKENNA (aka jmckenna)
- Angelos TZOTSOS (aka kalxas)
- Luca DELUCCHI (aka lucadelu)
- Nicolas BOZON (aka nbozon)
- Markus NETELER (aka neteler)
- Marco NEGRETTI (aka nmarco)
- Knut LANDMARK (aka knut)
- René-Luc D’HONT (aka reluc)
- Trevor CLARKE (aka tclarke)

### 5.6.2 Mailing Lists

**ZOO-Discuss**

The zoo-discuss list is your main source of support for the ZOO-Project.

- **Subscribing to zoo-discuss**
  
  To subscribe to the zoo-discuss listserv visit [http://lists.osgeo.org/cgi-bin/mailman/listinfo/zoo-discuss](http://lists.osgeo.org/cgi-bin/mailman/listinfo/zoo-discuss). You can later change your subscription information or leave the list at this website.

- **Submitting Questions to zoo-discuss**
  
  To submit questions to the zoo-discuss listserv, first join the list by following the subscription procedure above. Then post questions to the list by sending an email message to zoo-discuss@lists.osgeo.org.

- **Searching the Archives**
  
  All zoo-discuss archives are located in [http://lists.osgeo.org/pipermail/zoo-discuss/](http://lists.osgeo.org/pipermail/zoo-discuss/).
5.6.3 IRC Chat

Authors Nicolas Bozon, Gérald Fenoy, Jeff McKenna, Luca Delucchi

Last Updated $Date: 2013-03-28 11:00:07 +0100 (Thu, 28 Mar 2013) $

Some of the development of the ZOO-Project is coordinated through IRC. This page describes how you log on to chat, ask questions, and hack around with the developers.

Server and Channel Information

Server: irc.freenode.net
Channel: #zoo-project

Why IRC?

IRC is a primary medium where Open Source GIS hackers congregate, collaborate, and hack. It makes it easy to communicate things like compilation issues, where immediate, iterative feedback allows folks to make a lot of progress. Something that might take days of heavily-quoted emails through a mailinglist might only take fifteen minutes on IRC.

IRC is a great way to coordinate on-line meetings. The ZOO PSC conducts their monthly meetings through IRC.

How do I join?

- Chatzilla is probably the easiest way to get going. Chatzilla works with Mozilla or Firefox, and once you have it installed, you can log on to the channel by pointing your browser at:
  irc://irc.freenode.net/#zoo-project

- There are many other IRC clients available. This page provides a good listing for many different platforms.

- You can also connect to IRC directly in your Web browser without a client:
  - goto http://webchat.freenode.net/
  - enter a nickname (usually your first initial plus your last name)
  - in “Channels” enter: zoo-project
  - Connect!

IRC logs

It is possible to see the logs (history) of ZOO-Project IRC channel here

5.7 Documentation Development Guide

Authors Nicolas Bozon, Gérald Fenoy, Jeff McKenna

Last Updated $Date: 2015-02-25 13:23:00 +0100 (mer. 25 févr. 2015) $
5.7.1 License

The ZOO-Project documentation is released under the Creative Commons Attribution-ShareAlike 4.0 International Public License (CC-BY-SA). This license must be kept when editing or creating a new documentation file.

5.7.2 Background

The current structure of the ZOO Project documentation process is for developers with SVN commit access to maintain their documents in reStructuredText format, and therefore all documents live in the /docs directory in SVN. The Sphinx documentation generator is used to convert the reStructuredText files to html, and the live website is then updated on an hourly basis.

5.7.3 reStructuredText Reference Guides

- Docutils Quick reStructuredText
- Docutils reStructuredText Directives
- Sphinx’s reStructuredText Primer
- search Sphinx’s mailing list

5.7.4 reStructuredText Formatting

- All text should be hard breaks at or around the 80 column mark, just as the source code.

5.7.5 Installing and Using Sphinx for rst-html Generation

Note: You can browse the versions of the Sphinx packages here, and then install the exact version such as:

easy_install Sphinx==1.0.7

On Windows:
1. install Python 2.X

2. download setuptools

3. make sure that the C:/Python2X/Scripts directory is your path

4. execute the following at commandline:
   
   ```python
   easy_install Sphinx
   ```

   ...you should see message: “Finished processing dependencies for Sphinx”

   **Note:** Make sure you install Sphinx 1.0 or more recent. See note above.

5. install MiKTeX if you want to build pdfs

6. checkout the /docs directory from SVN, such as:
   
   ```bash
   svn checkout http://svn.zoo-project.org/svn/trunk zoo-project
   ```

7. inside the /docs directory, execute:
   
   ```bash
   make html
   ```
   
   or
   
   ```bash
   make latex
   ```

   the HTML output will be written to the _build/html sub-directory.

**On Linux:**

1. make sure you have the Python dev and setuptools packages installed. On Ubuntu:
   
   ```bash
   sudo apt-get install python-dev
   sudo apt-get install python-setuptools
   ```

2. install sphinx using easy_install:
   
   ```bash
   sudo easy_install Sphinx
   ```

   **Note:** Make sure you install Sphinx 1.0 or more recent. See note above.

3. checkout the /docs directory from SVN, such as:
   
   ```bash
   svn checkout http://svn.zoo-project.org/svn/trunk zoo-project
   ```

4. to process the docs, from the ZOO /docs directory, run:
   
   ```bash
   make html
   ```
   
   or
   
   ```bash
   make latex
   ```

   the HTML output will be written to the build/html sub-directory.

   **Note:** If there are more than one translation, the above commands will automatically build all translations.

**On Mac OS X:**
1. install sphinx using easy_install:
   ```
   sudo easy_install Sphinx
   ```

   **Note:** Make sure you install Sphinx 1.0 or more recent. See note above.

2. install MacTex if you want to build pdfs

3. checkout the /docs directory from SVN, such as:
   ```
   svn checkout http://svn.zoo-project.org/svn/trunk zoo-project
   ```

4. to process the docs, from the ZOO /docs directory, run:
   ```
   make html
   
or
   make latex
   ```
   the HTML output will be written to the build/html sub-directory.