ZOO-Project Documentation

Release 1.5

ZOO-Project team

November 24, 2017

CONTENTS

CHAPTER

ONE

INTRODUCTION

This is an introduction to the ZOO-Project[1] open source software documentation.

1.1 What is ZOO-Project ?

ZOO-Project[2] is a WPS (Web Processing Service) implementation written in C, Python and JavaScript. It is an open source platform which implements the WPS 1.0.0[3] and WPS 2.0.0[4] standards edited by the Open Geospatial Consortium[5] (OGC).

ZOO-Project[6] provides a developer-friendly framework for creating and chaining WPS compliant Web Services. Its main goal is to provide generic and standard-compliant methods for using existing open source librairies and algorithms as WPS. It also offers efficient tools for creating new innovative web services and applications.

ZOO-Project[7] is able to process geospatial or non geospatial data online. Its core processing engine (aka *ZOO-Kernel*) lets you execute a number of existing *ZOO-Services* based on reliable software and libraries. It also gives you the ability to create your own WPS Services from new or existing source code, which can be written in seven different programming languages. That lets you compose or turn code as WPS Services simply, with straightforward configuration and standard coding methods.

ZOO-Project[8] is very flexible with data input and output so you can process almost any kind of data stored locally or accessed from remote servers and databases. ZOO-Project excels in data processing and integrates new or existing spatial data infrastructures, as it is able to communicate with map servers and can integrate webmapping clients.

1.2 ZOO-Project components

The ZOO-Project[9] platform is made up of the following components:

- *ZOO-Kernel*: A WPS compliant implementation written in C offering a powerful WPS server able to manage and chain WPS services. by loading dynamic libraries and code written in different languages.

[1] http://zoo-project.org
[2] http://zoo-project.org
[3] http://www.opengeospatial.org/standards/wps/
[4] http://www.opengeospatial.org/standards/wps/
[5] http://www.opengeospatial.org/
[6] http://zoo-project.org
[7] http://zoo-project.org
[8] http://zoo-project.org
[9] http://zoo-project.org

- *ZOO-Services*: A growing collection of ready to use Web Processing Services built on top of reliable open source libraries such as GDAL, GRASS GIS, OrfeoToolbox, CGAL and SAGA GIS.
- *ZOO-API*: A server-side JavaScript API for creating, chaining and orchestrating the available WPS Services.
- *ZOO-Client*: A client side JavaScript API for interacting with WPS servers and executing standard requests from web applications.

1.3 Open Source

ZOO-Project[10] is open source and released under the terms of the MIT/X-11[11] license[12] . ZOO-Project activities are directed by the Project Steering Committee (PSC) and the software itself is being developed, maintained and documented by an international community of users and developers (aka ZOO-Tribe[13]).

Please refer to the ZOO-Project *Contributor Guide* if you want to participate and contribute. It is easy to *get involved* on source code, documentation or translation. Everybody is welcome to join the ZOO-Tribe[14].

ZOO-Project[15] is an incubating software at the Open Source Geospatial Foundation (OSGeo[16]).

[10] http://zoo-project.org
[11] http://opensource.org/licenses/MITlicense
[12] http://zoo-project.org/trac/browser/trunk/zoo-project/LICENSE
[13] http://zoo-project.org/new/ZOO-Project/ZOO%20Tribe
[14] http://zoo-project.org/new/ZOO-Project/ZOO%20Tribe/
[15] http://zoo-project.org
[16] http://osgeo.org

CHAPTER

TWO

ZOO-PROJECT INSTALLATION

The following sections will help you to install the ZOO-Project[1] Open WPS Platform on various operating systems.

2.1 Prerequisites

2.1.1 Mandatory

The following libraries are required to install *ZOO-Kernel*. Please make sure they are available on your system before anything else.

- autoconf (http://www.gnu.org/software/autoconf/)
- gettext (https://www.gnu.org/software/gettext/)
- cURL (http://curl.haxx.se)
- FastCGI (http://www.fastcgi.com)
- Flex & Bison (http://flex.sourceforge.net/ | http://www.gnu.org/software/bison/)
- libxml2 (http://xmlsoft.org)
- OpenSSL (http://www.openssl.org)
- GDAL (http://gdal.org/)

> **Warning:** It is mandatory to install every library listed above before compiling and installing ZOO-Kernel

2.1.2 Optional

You may also consider the following optional libraries:

- MapServer (for ZOO-Kernel optional WMS, WFS and WCS support) (http://mapserver.org)
- Python (http://www.python.org)
- PHP Embedded (for ZOO-Kernel optional PHP support) (http://www.php.net)
- Java SDK (for ZOO-Kernel optional Java support) (http://java.sun.com)
- SpiderMonkey (for ZOO-Kernel optional Javascript support) (http://www.mozilla.org/js/spidermonkey/)

[1] http://zoo-project.org

- SAGA GIS (for ZOO-Kernel optional SAGA support) (http://www.saga-gis.org/en/index.html/)
- OrfeoToolbox (for ZOO-Kernel optional OTB support) (https://www.orfeo-toolbox.org/)
- GRASS GIS (for using it through WPSGrassBridge) (http://grass.osgeo.org)
- PostgreSQL support activated in GDAL to *Use a Database Backend (Optional)*

2.2 Download

Several ways to download the ZOO-Project[2] source code are available and explained in this section.

Warning: The ZOO-Project svn is the place where developement happens. Checking out svn is the best way to be always up-to-date.

2.2.1 ZOO-Project releases archives

Each new ZOO-Project[3] major release are available on the project official website as .zip and .tar.bz2 archives. Head to the Downloads[4] section to get the latest or older ZOO-Project releases.

Warning: Don't use older versions of ZOO-Project if you want to use new features and avoid older code issues. Prefer svn or github instead.

2.2.2 ZOO-Project SVN

Download the latest[5] ZOO-Project[6] source code using the following *svn* command:

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

Registered ZOO-Project developers would prefer the following:

```
sed "s:\[tunnels\]:\[tunnels\]\nzoosvn = /usr/bin/ssh -p 1046:g" -i ~/.subversion/config
svn co svn+zoosvn://svn.zoo-project.org/var/svn/repos/trunk zoo-src
```

Note: The ZOO-Project svn server listens on the 1046 (1024+22) port (instead of 22 by default), so please use a specific tunnel to access the svn server, as shown in the command above.

2.2.3 ZOO-Project Github

The ZOO-Project svn is mirrored in this Github repository[7] in case you would like to fork it.

[2] http://zoo-project.org
[3] http://zoo-project.org
[4] http://zoo-project.org/site/Downloads
[5] http://zoo-project.org/trac/browser/trunk
[6] http://zoo-project.org
[7] https://github.com/kalxas/zoo-project/

2.3 Installation on Unix/Linux

To build and install ZOO-Project on your Web Server you will need 4 steps :

2.3.1 Build cgic

Run the following commands from the `thirds/cgic` directory to build the cgic library.

```
cd thirds/cgic
make
```

The cgic library originaly come from http://www.boutell.com/cgic.

Warning: You may need to edit the `Makefile` in case you are using a 64 bits platform for building and your fcgi library is not located in `/usr/lib64`.

2.3.2 Install ZOO-Kernel

For the impatient

Run the following commands from the directory where you *Download* and extracted the ZOO Kernel source code in order to build the `zoo_loader.cgi` CGI program with default options.

```
cd zoo-project/zoo-kernel
autoconf
./configure
make
make install
```

This should produce executables for the *zoo_loader.cgi* CGI program (located per default in `/usr/lib/cgi-bin/`) and a shared library `libzoo_service` (located per default in `/usr/local/lib`).

Warning: Edit ZOO-Kernel installation settings in the `main.cfg` file (set `tmpPath` and `tmpUrl` to fit your web server configuration).

Configure options

This section provides information on *ZOO-Kernel* configure options. It is recommanded to also read the *ZOO-Kernel configuration* section for configuration technical details.

Here is the list of available options in the same order as returned by `./configure --help` command:

Specific CGI Directory

In the case your `cgi-bin` is not located in `/usr/lib/` as it is assumed per default, then you can specify a specific target location by using the following option:

```
./configure --with-cgi-dir=/Library/WebServer/CGI-Executables
```

This way, when you will run the `make install` command, the ZOO-Kernel will be deployed in the specified directory (so, */Library/WebServer/CGI-Executables'* in this example).

Specific main.cfg location (Optional)

Per default, the ZOO-Kernel search for the `main.cfg` file from its installation directory but, in case you want to store this file in another place, then you can use the `--with-etc-dir` option so it will search for the `main.cfg` file in the `sysconfdir` directory.

For instance, you can define that the directory to store the `main.cfg` file is the `/etc/zoo-project` directory, by using the following command:

```
./configure --with-etc-dir=yes --sysconfdir=/etc/zoo-project
```

Use a Database Backend (Optional)

If you want to share the ongoing informations of running services between various ZOO-Kernel instances then you should use this option: `--with-db-backend`. This way, both the *GetStatus*, *GetResult* and *Dismiss* requests can be run from any host accessing the same database. Obviously, this will require that the ZOO-Kernel is able to access the Database server. To learn how to configure this connection and how to create this database please refer to *[1]* and *[2]* respectively.

Note: By now, the ZOO-Kernel is not able to handle correctly the *Dismiss* request from any host. Never-

theless, it will provide valid response from any host, but only the host which is really handling the service will be able to stop it and remove all the linked files.

To create a new database to be used by the ZOO-Kernel, you have to load the schema.sql[8] file. For instance, you may run the following:

```
createdb zoo_project
psql zoo_project -f zoo-project/zoo-kernel/sql/schema.sql
```

Note: You can choose another schema to store ZOO-Kernel specific informations. In such a case, you would need to edit the schema.sql file to uncomment line 33[9] and 34[10].

YAML Support (Optional)

If `yaml.h` file is not found in your `/usr/include` directory and `libyaml.so` is not found in `/usr/lib`, a `--with-yaml` option can be used to specify its location. For instance, if the headeer file lies in `/usr/local/include` and the shared library is located in `/usr/local/lib`, you may use the following command:

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

FastCGI Support (Required)

If your FastCGI library is not available in the default search path, a `--with-fastcgi` option can be used to specify its location. For instance, if `libfcgi.so` lies in `/usr/local/lib` which is not in your `LD_SEARCH_PATH`, you may use the following command:

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

GDAL Support (Required)

If gdal-config program is not found in your `PATH`, a `--with-gdal-config` option can be used to specify its location. For instance, if `gdal-config` lies in `/usr/local/bin` which is not in your `PATH`, you may use the following command:

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

GEOS Support (Optional)

If `geos-config` program is not found in your `PATH`, a `--with-geosconfig` option can be used to specify its location. For instance, if `geos-config` lies in `/usr/local/bin` which is not in your `PATH`, you may use the following command:

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

[8] http://zoo-project.org/trac/browser/trunk/zoo-project/zoo-kernel/sql/schema.sql
[9] http://zoo-project.org/trac/browser/trunk/zoo-project/zoo-kernel/sql/schema.sql#L33
[10] http://zoo-project.org/trac/browser/trunk/zoo-project/zoo-kernel/sql/schema.sql#L34

CGAL Support (Optional)

If `CGAL/Delaunay_triangulation_2.h` program is not found in your `/usr/include` directory, a `--with-cgal` option can be used to specify its location. For instance, if the file lies in `/usr/local/include` which is not in your PATH, you may use the following command:

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

MapServer Support (Optional)

In order to activate the WMS, WFS and WCS output support using MapServer, the `--with-mapserver` option must be used. The path to `mapserver-config` which is located in the source code of MapServer must also be set, using the following command:

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

Read more about the *Optional MapServer support*.

XML2 Support (Required)

If xml2-config program is not found in PATH, a *–with-xml2config* option can be used to specify its location. For instance, if xml2-config is installed in `/usr/local/bin` which is not in PATH, you may use the following command:

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

Python Support (Optional)

The `--with-python=yes` option is required to activate the *ZOO-Kernel* Python support, using the following command:

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

This assumes that python-config is found in your `PATH`. If not, then you can specify the Python installation directory using the following command (with Python installed in the `/usr/local` directory):

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

Python Version If multiple Python versions are available and you want to use a specific one, then you can use the `--with-pyvers` option as shown bellow:

```
$ ./configure --with-pyvers=2.7
```

JavaScript Support (Optional)

In order to activate the JavaScript support for ZOO-Kernel, the `--with-js=yes` configure option must be specified. 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. The following command is only needed (assuming that js_api.h and libmozjs.so are found in default directories):

```
$ ./configure --with-js=yes
```

If you want to use a custom installation of SpiderMonkey[11] , or if you are not using a Debian packaging system, then you'll have to specify the directory where it is installed. For instance, if SpiderMonkey is in `/usr/local/`, then the following command must be used:

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

PHP Support (Optional)

The `--with-php=yes` option is required to activate the *ZOO-Kernel* PHP support', using the following command:

```
$ ./configure --with-php=yes
```

This assumes that `php-config` can be found in the `<PATH>/bin` directory . So, supposing the your `php-config` can be found in `/usr/local/bin`, then use the following command:

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

Warning: ZOO-Kernel optional PHP support requires a local PHP Embedded installation. Read more here[a].

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

Java Support (Optional)

In order to activate the Java support for ZOO-Kernel, the *–with-java* configure option must be specified and sets the installation path of your Java SDK. For instance, if Java SDK is installed in the `/usr/lib/jvm/java-6-sun-1.6.0.22/` directory, then the following command can be used:

```
$ ./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/`, and that the `include/linux` directory contains the `jni.h` headers file and that the `jre/lib/i386/client/` directory contains the `libjvm.so` file.

Note: You can use the *–with-java-rpath* option to produce a binary aware of the libjvm location.

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:

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

Perl Support (Optional)

The `--with-perl=yes` option can be used for activating the ZOO-Kernel Perl support, as follow:

```
$ ./configure --with-perl=yes
```

[11] https://developer.mozilla.org/en/SpiderMonkey

This assumes that perl is found in your PATH. For instance, if Perl is installed in `/usr/local` and `/usr/local/bin` is not found in your `PATH`, then the following command can be used (this assumes that `/usr/local/bin/perl` exists):

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

Orfeo Toolbox Support (Optional)

In order to activate the optional Orfeo Toolbox support, the `--with-otb` option must be used, using the following command:

```
$ ./configure --with-otb=/path/to/your/otb/
```

Read more about the *Optional Orfeo Toolbox support*.

Warning: To build the Orfeo Toolbox support you will require ITK, the default version of ITK is 4.5, in case you use another version, please make sure to use the `--with-itk-version` to specificy what is the version available on your system.

SAGA GIS Support (Optional)

In order to activate the optional SAGA GIS support, the *–with-saga* option must be used, using the following command:

```
$ ./configure --with-saga=/path/to/your/saga/
```

Read more about the *Optional SAGA GIS support*.

Warning: In case wx-config is not in your `PATH` please, make sure to use the `--with-wx-config` to specify its location.

Translation support (Optional)

The ZOO-Kernel is able to translate the messages it produces in different natural languages. This requires that you download the messages file[12] translated in your language, if any. Then, for this translation support to work, you have to generate manually the requested file on your system. For instance for the French translation, you may use the following command:

```
msgfmt messagespo_fr_FR.utf8.po -o /usr/share/locale/fr/LC_MESSAGES/zoo-kernel.mo
```

The ZOO-Kernel is also able to handle translation of ZOO-Services. Please, refer to *this document* for more details on the procedure to add new ZOO-Service translation files.

Warning: The location of the final `.mo` file may vary depending on your system setup.

[12] https://www.transifex.com/projects/p/zoo-kernel-internationalization/

2.3.3 Install ZOO-Services

Warning: We present here a global installation procedure for basics ZOO-Services, for details about automatic installation of services provided by *Optional Orfeo Toolbox support* or *Optional SAGA GIS support*, please refer to there specific documentations.

Depending on the programming language used to implement the ZOO-Services you want to install, you will need to build a Services Provider. In the case of *C* and *Fotran*, you would create a shared library exporting the functions corresponding to all the ZOO-Services provided by this Services Provider. In case of *Java*, you will need to build a Java Class. In any other programming language, you should simply have to install the ServiceProvider and the zcfg files.

If building a Shared library or a Java class is required, then you should find a `Makefile` in the service directory which is responsible to help you build this Services Provider. So you should simply run the *make* command from the Service directory to generate the required file.

Then you simply need to copy the content of the `cgi-env` directory in `cgi-bin`.

To install the `ogr/base-vect-ops` Services Provider, supposing that your `cgi-bin` directory is `/usr/local/lib` use the following commands:

```
cd zoo-project/zoo-services/ogr/base-vect-ops
make
cp cgi-env/*.* /usr/lib/cgi-bin
```

Note: You may also run `make install` directly after `make`.

To install the hello-py Services Provider, use the following commands:

```
cd zoo-project/zoo-services/hello-py/
cp cgi-env/* /usr/lib/cgi-bin
```

2.3.4 Testing your installation

To test your installation yous should first be able to run the following command from the `cgi-bin` directory:

```
./zoo_loader.cgi "request=GetCapabilities&service=WPS"
```

2.4 Installation on Debian / Ubuntu

Use the following instructions to install ZOO-Project[13] on Debian or Ubuntu distributions.

2.4.1 Prerequisites

Using Debian

The following command should install all the required dependancies on Debian. See the *Prerequisites* section for additional information.

[13] http://zoo-project.org

```
apt-get install flex bison libfcgi-dev libxml2 libxml2-dev curl openssl autoconf apache2 python-softw
```

Using Ubuntu

On Ubuntu, use the following command first to install the required dependancies :

```
sudo apt-get install flex bison libfcgi-dev libxml2 libxml2-dev curl openssl autoconf apache2 python-
```

Then add the *UbuntuGIS* repository in order to get the latest versions of libraries

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

Install the geographic library as follow:

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

2.4.2 Installation

Download ZOO-Project latest version from svn using the following command:

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

Install the *cgic* library from packages using the following command:

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

Head to the *ZOO-Kernel* directory

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

Create a configure file as follow:

```
autoconf
```

Run configure with the desired options, for example with the following command:

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

Note: Refer to the installation section for the full list of available options

Compile ZOO-Kernel as follow:

```
make
```

Install the `libzoo_service.so.1.5` by using the following command:

```
sudo make install
```

Copy the necessary files to the *cgi-bin* directory (as administrator user):

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

Install ZOO ServiceProviders, for example the basic Python service (as administrator user)

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

Edit the *main.cfg* file as follow (example configuration):

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

Test the ZOO-Kernel installation with the following requests:

```
http://127.0.0.1/cgi-bin/zoo_loader.cgi?ServiceProvider=&metapath=&Service=WPS&Request=GetCapabilitie
```

```
http://127.0.0.1/cgi-bin/zoo_loader.cgi?ServiceProvider=&metapath=&Service=WPS&Request=DescribeProces
```

```
http://127.0.0.1/cgi-bin/zoo_loader.cgi?ServiceProvider=&metapath=&Service=WPS&Request=Execute&Versio
```

Note: Such request should return well formed XML documents (OWS documents responses).

Warning: The URLs provided here suppose that you have previously setup a web server and defined cgi-bin as a location where you can run cgi application.

Warning: If ZOO-Kernel returns an error please check the *ZOO-Kernel configuration* and beware of the *Prerequisites*.

2.5 Install on OpenSUSE

ZOO-Kernel is maintained as a package in OpenSUSE Build Service (OBS)[14]. RPM are thus provided for all versions of OpenSUSE Linux (11.2, 11.3, 11.4, Factory).

2.5.1 Stable release

Use the following instructions to install ZOO-Project latetst release on OpenSUSE distribution.

One-click installer

A one-click installer is available here[15]. For openSUSE 11.4, follow this direct link[16].

Yast software manager

Add the Application:Geo[17] repository to the software repositories and then ZOO-Kernel can then be found in Software Management using the provided search tool.

[14] https://build.opensuse.org/package/show?package=zoo-kernel&project=Application%3AGeo
[15] http://software.opensuse.org/search?q=zoo-kernel&baseproject=openSUSE%3A11.4&lang=en&exclude_debug=true
[16] http://software.opensuse.org/ymp/Application:Geo/openSUSE_11.4/zoo-kernel.ymp?base=openSUSE%3A11.4&query=zoo-kernel
[17] http://download.opensuse.org/repositories/Application:/Geo/

Command line (as root for openSUSE 11.4)

Install ZOO-Kernel package by yourself using the following command:

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

Developement version

The latest development version of ZOO-Kernel can be found in OBS under the project home:tzotsos[18]. ZOO-Kernel packages are maintained and tested there before being released to the Application:Geo repository. Installation methods are identical as for the stable version. Make sure to use this[19] repository instead.

Command line (as root for openSUSE 11.4)

Install latest ZOO-Kernel trunk version with the following command:

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

Note that there is the option of adding the zoo-wps-grass-bridge package. This option will automatically install grass7 (svn trunk).

2.6 Installation on CentOS

Use the following instructions to install ZOO-Project[20] on CentOS distributions.

2.6.1 Prerequisites

First you should add the ELGIS Repository[21] then install the dependencies by using *yum* commands.

```
rpm -Uvh http://elgis.argeo.org/repos/6/elgis-release-6-6_0.noarch.rpm
rpm -Uvh \
  http://download.fedoraproject.org/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm
wget\
  http://proj.badc.rl.ac.uk/cedaservices/raw-attachment/ticket/670/armadillo-3.800.2-1.el6.x86_64.rpm
yum install armadillo-3.800.2-1.el6.x86_64.rpm
yum install hdf5.so.6
yum install gcc-c++ zlib-devel libxml2-devel bison openssl \
  python-devel subversion libxslt-devel libcurl-devel \
  gdal-devel proj-devel libuuid-devel openssl-devel fcgi-devel
yum install java-1.7.0-openjdk-devel
```

[18] https://build.opensuse.org/project/show?project=home%3Atzotsos
[19] http://download.opensuse.org/repositories/home:/tzotsos/
[20] http://zoo-project.org
[21] http://elgis.argeo.org

2.6.2 Installation

Now refer to general instructions from *Installation on Unix/Linux* to setup your ZOO-Kernel and the ZOO-Services of your choice.

Note: In case you use the Java support, please, make sure to use the correct version of both java and javac using the following commands:

```
update-alternatives --config java
update-alternatives --config javac
```

Also, make sure to add the following to your *main.cfg* file before trying to execute any Java service:

```
[javax]
ss=2m
```

2.7 Installation on Windows ™

2.7.1 Install ZOO-Project binaries

Note: The content of the ZOO-Project Windows-Binaries is based on GISInternals SDK[22], make sure to refer to license informations.

Note: When using the ZOO-Project Windows-Binaries, you can decide if you want the Java support activated or not (which is the case per default). Indeed, once your installation has been done, you will have both a *zoo_loader.cgi* and *zoo_loader_java.cgi* which correspond respectively to the ZOO-Kernel without and with Java support activated. So, in case you want to use the Java support, simply rename the *zoo_loader_jave.cgi* file located in *c:\inetpub\cgi-bin* to *zoo_loader.cgi* and make sure the *jvm.dll* can be found.

Using the installer

Prior to run the ZOO-Project-Installer, please make sure you have IIS and Python[23] setup on your machine. Then download the ZOO-Project-Installer[24] corresponding to your platform. The first time you will run the installer binary, you may be prompted to authorize it to run. Once the installer has been run, simply access the following link: http://localhost/zoo-demo/ to access your local demo application.

Install by hand

Prior to run the ZOO-Project-Installer, please make sure you have IIS and Python[25] setup on your machine. Then download the ZOO-Project[26] archive corresponding to your platform. Uncompress it, then move *cgi-bin*, *data* and *tmp* from uncompressed folder to *c:\inetpub*, also move *wwwroot\zoo-demo* and *wwwroot\tmp* to *c:\inetpub\wwwroot*. To finish the installation, run the folllowing command as administrator to allow the *zoo_loader.cgi* to run from http://localhost/cgi-bin/zoo_loader.cgi:

[22] http://www.gisinternals.com/release.php
[23] https://www.python.org/downloads/windows/
[24] https://bintray.com/gfenoy/ZOO-Project/Windows-Binaries/view
[25] https://www.python.org/downloads/windows/
[26] https://bintray.com/gfenoy/ZOO-Project/Windows-Binaries/view

```
cd C:\Windows\System32\inetsrv
appcmd.exe add vdirs /app.name:"Default Web Site/" /path:/cgi-bin /physicalPath:c:\inetpub\cgi-b
appcmd set config /section:handlers /+[name='CGI-exe1',path='*.cgi',verb='*',modules='CgiModule'
appcmd.exe set config /section:isapiCgiRestriction /+[path='c:\inetpub\cgi-bin\zoo_loader.cgi',d
```

2.7.2 Compile ZOO-Project from source

Warning: Ensure to first 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 2010).

1. Make sure the gnuwin32 tools `bison.exe` and `flex.exe` are found in your path. You can download the GNUwin32 tools here[27].
2. Modify the `nmake.opt` file to point to your local libraries. Note that you can also use definition directly in the command line if you prefer. See *Build options* for details about this options.
3. Execute:

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

4. A file `zoo_loader.cgi` and `libzoo_service.dll` should be created. Note that if another file named `zoo_loader.cgi.manifest` is also created, you will have to run another command:

```
nmake /f makefile.vc embed-manifest
```

5. Copy the files `zoo_loader.cgi`, `libzoo_service.dll` 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:ExceptionReport xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:xsi="http://www.w3.org/200
  <ows:Exception exceptionCode="MissingParameterValue">
    <ows:ExceptionText>Parameter &lt;request&gt; was not specified</ows:ExceptionText>
  </ows:Exception>
</ows:ExceptionReport>
```

7. Edit the `main.cfg` file 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/
```

[27] http://www.zoo-project.org/dl/tool-win32.zip

```
[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
addressCity=Lunenburg
addressAdministrativeArea=False
addressPostalCode=B0J 2C0
addressCountry=ca
addressElectronicMailAddress=info@gatewaygeomatics.com
phoneVoice=False
phoneFacsimile=False
```

8. Open a web browser window, and execute a GetCapababilites 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.ne
<ows:ServiceIdentification>
  <ows:Title>The Zoo WPS Development Server</ows:Title>
  <ows:Abstract>
    Development version of ZooWPS. See http://www.zoo-project.org
  </ows:Abstract>
  <ows:Keywords>
    <ows:Keyword>WPS</ows:Keyword>
    <ows:Keyword>GIS</ows:Keyword>
    <ows:Keyword>buffer</ows:Keyword>
  </ows:Keywords>
  <ows:ServiceType>WPS</ows:ServiceType>
  <ows:ServiceTypeVersion>1.0.0</ows:ServiceTypeVersion>
  ...
```

Build options

Various build options can be set in the `nmake.opt` file to define the location of the built libraries you want to use to build your ZOO-Kernel. Some are optional and some are required, they are listed below exhaustively:

gettext (Required)

The location of the libintl (built when building gettext) should be specified by defining the `INTL_DIR` environment variable. It supposes that the header and the `intl.lib` file are available.

So for instance, in case you build the gettext in `\buildkit\srcs\gettext-0.14.6`, you may define the following before running `nmake /f makefile.vc`:

```
set INTL_DIR=\buildkit\srcs\gettext-0.14.6\gettext-runtime\intl
```

libCURL (Required)

The location of the libCURL should be specified by defining the `CURL_DIR` environment variable. It supposes that there are 2 sub-directory `include` containing the libCURL header and `lib` which contains the `libcurl.lib` file.

So for instance, in case you build the libCURL in `\buildkit\srcs\curl-7.38.0`, you may define the following before running `nmake /f makefile.vc`:

```
set CURL_DIR=\buildkit\srcs\curl-7.38.0\builds\libcurl-vc10-x86-release-dll-ssl-dll-zlib-dll-ipvs6-ss
```

libFCGI (Required)

The location of the libFCGI should be specified by defining the `FCGI_DIR` environment variable. It supposes that there are 2 sub-directory `include` containing the FastCGI header and `libfcgi/Release` which contains the `libfcgi.lib` file.

So for instance, in case you build the libXML2 library in `\buildkit\srcs\fcgi-2.4.1`, you may define the following before running `nmake /f makefile.vc`:

```
set FCGI_DIR=\buildkit\srcs\fcgi-2.41.1
```

libXML2 (Required)

The location of the libXML2 should be specified by defining the `XML2_DIR` environment variable. It supposes that there are 2 sub-directory `include` containing the libXML2 header and `win32\bin.msvc` which contains the `libxml2.lib` file.

So for instance, in case you build the libXML2 library in `\buildkit\srcs\libxml2-2.9.0`, you may define the following before running `nmake /f makefile.vc`:

```
set XML2_DIR=\buildkit\srcs\libxml2-2.9.0
```

OpenSSL (Required)

The location of the OpenSSL library should be specified by defining the `SSL_DIR` environment variable. It supposes that there are 2 sub-directory `inc32` containing the header files and `out32dll` which contains the `ssleay32.lib` file.

So for instance, in case you build the libXML2 library in `\buildkit\srcs\openssl-1.0.2c`, you may define the following before running `nmake /f makefile.vc`:

```
set SSL_DIR=\buildkit\srcs\openssl-1.0.2c
```

GDAL (Required)

The location of the GDAL library should be specified by defining the `GDAL_DIR` environment variable. It corresponds to the path where you uncompress and built GDAL, it supposes that you have the `gdal_i.lib` file available in this directory.

So for instance, in case you build the libXML2 library in `\buildkit\srcs\gdal-1.10.1`, you may define the following before running `nmake /f makefile.vc`:

```
set GDAL_DIR=\buildkit\srcs\gdal-1.10.1
```

MapServer (Optional)

The location of the MapServer library path should be specified by defining the `MS_DIR` environment variable. It corresponds to the path where you build MapServer on your system, this directory should contain the `nmake.opt` file used.

So for instance, in case you build Python in `\buildkit\srcs\mapserver-6.2.0`, you may define the following before running `nmake /f makefile.vc`:

```
set MS_DIR=\buildkit\srcs\mapserver-6.2.0
```

Python (Optional)

The location of the Python binaries path should be specified by defining the `PY_DIR` environment variable. It corresponds to the path where you build Python on your system. The location of the `pythonXX.lib` files should be specified by setting the `PY_LIBRARY` environment variable.

So for instance, in case you build Python in `\buildkit\srcs\Python-2.7`, you may define the following before running `nmake /f makefile.vc`:

```
set PY_DIR=\buildkit\srcs\Python-2.7
set PY_LIBRARY=\buildkit\srcs\Python-2.7\PCBuild\python27.lib
```

JavaScript (Optional)

The location of libmozjs should be specified by defining the `JS_DIR` environment variable. It corresponds to the path where you build libmozjs on your system, it supposes that the header and the `mozjs185-1.0.lib` file are available in this directory.

So for instance, in case you build libmozjs in `\buildkit\srcs\js-1.8.5`, you may define the following before running `nmake /f makefile.vc`:

```
set JS_DIR=\buildkit\srcs\js-1.8.5
```

PHP (Optional)

The location of PHP should be specified by defining the `PHP_DIR` environment variable. It corresponds to the path where you build PHP on your system. The location of the `php5embed.lib` files should be specified by setting the `PHP_LIB` environment variable.

So for instance, in case you build PHP in `\buildkit\srcs\php-5.5.10`, you may define the following before running `nmake /f makefile.vc`:

```
set PHP_DIR=\buildkit\srcs\php-5.5.10
set PHP_LIB=\buildkit\srcs\php-5.5.10\Release_TS\php5embed.lib
```

Database backend (Optional)

ZOO-Kernel can use a database backend to store ongoing status informations of running services, for activating this operation mode, you should define the evironment variable `DB` and set it to any value. So, to activate this option, you may use the following before running `nmake /f makefile.vc`:

```
set DB=activated
```

Note: To learn how to setup the corresponding database, please refer to *this section*.

Optionally Compile Individual Services

An example could be the *OGR base-vect-ops* provider located in the `zoo-project\zoo-services\ogr\base-vect-ops` directory.

1. First edit the *makefile.vc* located in that directory, and execute:

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

   Inside that same directory, the *ogr_service.zo* file should be created.

2. Copy all 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=B
project.org%3A8082%2Fgeoserver%2Fows%3FSERVICE%3DWFS%26REQUEST%3DGetFeature%26VERSION%3D1.0.0%2

   The response displayed in your browser should contain:

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

CHAPTER

THREE

ZOO-KERNEL

This section provides information on **ZOO-Kernel** , the ZOO-Project[1] WPS server. It will help you to configure and compile ZOO-Kernel.

3.1 What is ZOO-Kernel ?

ZOO-Kernel is the heart of the ZOO-Project[2] WPS platform. It is a WPS compliant implementation written in C language which provides a powerful and extensible WPS server.

ZOO-Kernel is an extensible WPS server that makes your system more powerful. It provides a full-featured processing engine which runs on Linux, Mac OSX ™ and Windows ™ operating systems. ZOO-Kernel is a CGI program which works on common web servers (namely Apache[3] or IIS[4] ™). It can be seamlessly integrated to new or existing web platforms.

ZOO-Kernel lets you process geospatial or non geospatial data using well formed WPS requests. The WPS server is able to manage and chain WPS Services (see ZOO-Services for examples) by loading dynamic libraries and source code written in different programming languages.

3.1.1 First class WPS server

Simple

The ZOO-Kernel rely on simple principles and tends to ease the implementation of new services by sharing similar data structures for every supported programming languages. The ZOO-Kernel is responsible to parse the requests it receives and return the corresponding WPS response.

In case of an *Execute* request, the ZOO-Kernel stores informations in a basic KVP data structure for the programming language used to implement the service, dynamically load the Service Provider defined in the zcfg file and run a specific function corresponding to the service, passing three arguments. Once the function return, ZOO-Kernel knows if the service run succeessfuly or failed by checking the returned value. In the case it succeeded, the ZOO-Kernel then parse the third arguments containing the result and produce the output in the desired format.

[1] http://zoo-project.org
[2] http://zoo-project.org
[3] http://httpd.apache.org/
[4] http://www.iis.net/

Compliant

ZOO-Kernel implements and complies with the WPS 1.0.0[5] and the WPS 2.0.0[6] standards edited by the Open Geospatial Consortium[7]. It is able to perform the WPS operations defined in the OpenGIS ® specification, such as:

- **GetCapablities**: Returns service-level metadata information.It provides the list of available processing services.
- **DescribeProcess**: Returns a description of a process, including its supported input and output.
- **Execute**: Launches computation and returns the output produced by a particular process.
- **GetStatus**: only available in WPS 2.0.0, it lets the client fetch the ongoing status of a running service.
- **GetResult**: only available in WPS 2.0.0, it lets the client fetch the final result of a running service.
- **Dismiss**: only available in WPS 2.0.0, it lets the client ask the server to stop a running service and remove any file it created.

ZOO-Kernel compliancy and performances can be tested using the following tools:

- cptesting[8]
- WPS Test Suite provided by the OGC compliancy program[9]
- XML responses validity can also be simply tested using XMLint[10].

Polyglot

ZOO-Kernel is a **polyglot**. The software is written in a valid form of multiple programming languages, which performs the same operations independent of the programming language used to compile or interpret it. The supported programming languages are listed bellow:

Language	Service-Provider	DataStructure	Return
C / C++	Shared Library	maps* M	integer
Java	Class File	HashMap[11]	integer
Python	Module File	Dictionary[12]	integer
PHP	Script File	Array[13]	integer
Perl	Script File		integer
Ruby	Script File	Hash[14]	integer
Fortran	Shared Library	CHARACTER*(1024) M(10,30)	integer
JavaScript	Script file	Object[15] or Array	Object/Array

[5] http://www.opengeospatial.org/standards/wps/
[6] http://www.opengeospatial.org/standards/wps/
[7] http://www.opengeospatial.org/
[8] https://github.com/WPS-Benchmarking/cptesting
[9] http://cite.opengeospatial.org/
[10] http://xmlsoft.org/xmllint.html/
[11] http://download.oracle.com/javase/6/docs/api/java/util/HashMap.html
[12] http://docs.python.org/tutorial/datastructures.html#dictionaries
[13] http://php.net/manual/language.types.array.php
[14] http://ruby-doc.org/core-2.2.0/Hash.html
[15] http://www.json.org/

3.2 ZOO-Kernel configuration

3.2.1 Main configuration file

ZOO-Kernel general settings are defined in a configuration file called `main.cfg`. This file is stored in the same directory as ZOO-Kernel (`/usr/lib/cgi-bin/` in most cases). It provides usefull metadata information on your ZOO-Kernel installation.

> **Warning:** ZOO-Kernel (`/usr/lib/cgi-bin/zoo_loader.cgi`) and its configuration file (`/usr/lib/cgi-bin/main.cfg`) must be in the same directory.

Note: Information contained by `/usr/lib/cgi-bin/main.cfg` is accessible from WPS Services at runtime, so when *Execute* requests are used.

Default main.cfg

An example *main.cfg* file is given here as reference.

```
[headers]
X-Powered-By=ZOO@ZOO-Project

[main]
version=1.0.0
encoding=utf-8
dataPath=/var/data
tmpPath=/var/www/temp
cacheDir=/var/www/cache
sessPath=/tmp
serverAddress=http://localhost/cgi-bin/zoo_loader.cgi
lang=fr-FR,ja-JP
language=en-US
mapserverAddress=http://localhost/cgi-bin/mapserv.cgi
msOgcVersion=1.0.0
tmpUrl=http:/localhost/temp/
cors=false

[identification]
keywords=t,ZOO-Project, ZOO-Kernel,WPS,GIS
title=ZOO-Project demo instance
abstract= This is ZOO-Project, the Open WPS platform.
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://geolabs.fr
phoneFacsimile=False
```

```
addressElectronicMailAddress=gerald@geolabs.fr
addressCity=Lattes
individualName=Gerald FENOY
```

Main section

The main.cfg `[main]` section parameters are explained bellow.

- `version`: Supported WPS version.
- `encoding`: Default encoding of WPS Responses.
- `dataPath`: Path to the directory where data files are stored (used to store mapfiles and data when MapServer support is activated).
- `tmpPath`: Path to the directory where temporary files are stored (such as *ExecuteResponse* when *storeExecuteResponse* is set to true).
- `tmpUrl`: URL to access the temporary files directory (cf. `tmpPath`).
- `cacheDir`: Path to the directory where cached request files [1] are stored (optional).
- `serverAddress`: URL to the ZOO-Kernel instance.
- `mapservAddress`: URL to the MapServer instance (optional).
- `msOgcVersion`: Version of all supported OGC Web Services output [2] (optional).
- `lang`: Supported natural languages separated by a coma (the first is the default one),
- `cors`: Define if the ZOO-Kernel should support Cross-Origin Resource Sharing[16]. If this parameter is not defined, then the ZOO-Kernel won't support CORS.
- `servicePath`: Define a specific location to search for services rather than using the ZOO-Kernel directory. If this parameter is not defined, then the ZOO-Kernel will search for services using its directory.
- `libPath`: (Optional) Path to a directory where the ZOO-kernel should search for service providers, e.g., shared libraries with service implementations (the `serviceProvider` parameter in the service configuration (.zcfg) file).

Warning: The `libPath` parameter is currently only recognized by services implemented in C/C++ or PHP, and may be moved to another section in future versions.

In case you have activated the MapServer support, please refer to *this specific section*.

Identification and Provider

The `[identification]` and `[provider]` sections are not ZOO-Project specific. They provide OGC metadata [3] and should be set according to the XML Schema Document[17] which encodes the parts of ISO 19115 used by the common *ServiceIdentification* and *ServiceProvider* sections of the *GetCapabilities* operation response, known as the service metadata XML document.

[1] If GET requests are passed through `xlink:href` to the ZOO-Kernel , the latter will execute the request the first time and store the result on disk. The next time the same request is executed, the cached file will be used and this will make your process run much faster. If `cachedir` was not specified in the `main.cfg` then the `tmpPath` value will be used.

[2] Usefull when the *Optional MapServer support* is activated (available since ZOO-Project version 1.3.0).

[16] https://www.w3.org/TR/cors/

[3] ZOO-Kernel and MapServer are sharing the same metadata for OGC Web Services if the *Optional MapServer support* is activated.

[17] http://schemas.opengis.net/ows/1.1.0/ows19115subset.xsd

Details of the common OWS 1.1.0 *ServiceIdentification* section can be found in this XML Schema Document[18].

Details of the common OWS 1.1.0 *ServiceProvider* section can be found in this XML Schema Document[19].

3.2.2 Additional sections

All the additional sections discribed in the following section are optional.

Headers section

The `[headers]` section can be set in order to define a specific HTTP Response header, which will be used for every response. As an example, you can check http://zoo-project.org using *curl* command line tool and notice the specific header *X-Powered-By: Zoo-Project@Trac*.

In case you want to allow CORS support for POST requests coming from `myhost.net`, then you should define the following minimal parameters in this section:

```
Access-Control-Allow-Origin=myhost.net
Access-Control-Allow-Methods=POST
Access-Control-Allow-Headers=content-type
```

env section

The `[env]` section can be used to store specific environment variables to be set prior the loading of *Services Provider* and Service execution.

A typical example is when a Service requires the access to a X server running on *framebuffer*, which takes to set the DISPLAY environnement variable, as follow:

```
[env]
DISPLAY=:1
```

In case you have activated the OTB support, please refer to *this specific section*.

lenv section

The `lenv` section is used by the ZOO-Kernel to store runtime informations before the execution of a WPS service, it contains the following parameters:

- `sid` (r): The WPS Service unique identifier,
- `status` (rw): The current progress value (a value between 0 and 100 in percent (%)),
- `cwd` (r): The current working directory of ZOO-Kernel,
- `message` (rw): An error message used when `SERVICE_FAILED` is returned (optional),
- `cookie` (rw): The cookie to be returned to the client (for example for authentication purpose).
- `file.pid` (r): The file used by the ZOO-Kernel to store process identifier.
- `file.sid` (r): The file used by the ZOO-Kernel to store service identifier.
- `file.responseInit` (r): The file used by the ZOO-Kernel to store the initial (then final) WPS response.

[18] http://schemas.opengis.net/ows/1.1.0/owsServiceIdentification.xsd
[19] http://schemas.opengis.net/ows/1.1.0/owsServiceProvider.xsd

- `file.responseFinal` (r): The file used by the ZOO-Kernel to temporary store the final WPS response.

renv section

The `renv` section is automatically created by the ZOO-Kernel before the execution of a WPS service, it contains all the environment variables available at runtime (so including the header fields in case it is used through http, refer to [https://tools.ietf.org/html/rfc3875 rfc3875] for more details).

senv section

The `senv` section can be used to store sessions information on the server side. Such information can then be accessed automatically from the Service if the server is requested using a valid cookie (as defined in `lenv` section). ZOO-Kernel will store the values set in the `senv` maps on disk, load it and dynamically replace its content to the one in the `main.cfg`. The `senv` section must contain the following parameter at least:

- `XXX`: The session unique identifier where `XXX` is the name included in the cookie which is returned.

For instance, adding the following in the Service source code :

```
conf["lenv"]["cookie"]="XXX=XXX1000000; path=/"
conf["senv"]={"XXX": "XXX1000000","login": "demoUser"}
```

means that ZOO-Kernel will create a file named `sess_XXX1000000.cfg` in the `cacheDir` directory, and will return the specified cookie to the client. Each time the client will request ZOO-Kernel using this cookie, it will automatically load the value stored before the Service execution.

Security section

The `[security]` section can be used to define what headers, the ZOO-Kernel has initially received in the request, should be passed to other servers for accessing resources (such as WMS, WFS, WCS or any other file passed as a reference). This section contains two parameters:

- `attributes`: The header to pass to other servers (such as Authorization, Cookie, User-Agent ...),
- `hosts`: The host for wich the restriction apply (can be "*" to forward header to every server or a coma separated list of host names, domain, IP).

Both parameters are mandatory.

Suppose you need to share Authorization, Cookie and User-Agent to every server for accessing ressources, then yo ucan use the following section definition:

```
[security]
attributes=Authorization,Cookie,User-Agent
hosts=*
```

In case only local servers require such header forwarding, you may use the following definition:

```
[security]
attributes=Authorization,Cookie,User-Agent
hosts=localhost,127.0.0.1
```

Database section

The database section allows to configure the *ZOO-Kernel optional database support*.

```
[database]
dbname=zoo_project
port=5432
user=username
host=127.0.0.1
type=PG
schema=public
```

This will generate strings to be passed to GDAL to connect the database server:

```
<type>:host=<host> port=<port>  user=<user> dbname=<dbname>
```

With the previous database section, it will give the following:

```
PG:"dbname=zoo_project host=127.0.0.1 port=5432 user=username"
```

Please refer to this section to learn how to setup the database.

Include section

The `[include]` section (optional) lists explicitely a set of service configuration files the the ZOO-Kernel should parse, e.g.,

```
[include]
servicename1 = /my/service/repository/service1.zcfg
servicename2 = /my/service/repository/service2.zcfg
```

The `[include]` section may be used to control which services are exposed to particular user groups. While service configuration files (.zcfg) may be located in a common repository or in arbitrary folders, main.cfg files at different URLs may include different subsets of services.

When the ZOO-Kernel handles a request, it will first check if there is an `[include]` section in main.cfg and then search for other .zcfg files in the current working directory (CWD) and subdirectories. If an included service happens to be located in a CWD (sub)directory, it will be published by its name in the `[include]` section. For example, the service `/[CWD]/name/space/myService.zcfg` would normally be published as name.space.myService, but if it is listed in the `[include]` section it will be published simply as myService:

```
[include]
myService =  /[CWD]/name/space/myService.zcfg
```

On the other hand, with

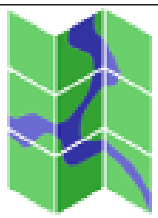
```
[include]
myService =  /some/other/dir/myService.zcfg
```

there would be two distinct services published as myService and name.space.myService, respectively, with two different zcfg files.

Note: As currently implemented, the ZOO-Kernel searches the CWD for the library files of included services if the `libPath` parameter is not set.

3.3 Optional MapServer support

Processing geospatial data using WPS Services is usefull. Publishing their results directly as WMS, WFS or WCS ressources is even more convenient. This is possible since ZOO-Project 1.3[20] using the **optional MapServer support**. The latter thus allows for automatic publication of WPS Service output as WMS/WFS or WCS using a *ZOO-Kernel* specific internal mechanism which is detailed in this section.



Note: MapServer[21] is an open source WMS/WFS/WCS server. Learn more by reading its documentation[22].

3.3.1 How does it work ?

If a request with `mimeType=image/png` is sent to *ZOO-Kernel*, the latter will detect that the *useMapServer* option is set to true an it will automatically:

- Execute the service using the <*Default*> block definition (these values must be understood by GDAL[23])
- Store the resulting output on disk (in the `[main]` > `dataPath` directory)
- Write a mapfile[24] (in the `[main]` > `dataPath` directory) using the MapServer[25] C-API (this sets up both WMS and WFS services).

Existing WPS Services source code doesn't need to be modified once the MapServer support is activated. It only takes to edit their respective *ZOO-Service configuration file* files accordingly.

Note: In case of a vector data source output, both WMS and WFS configuration are included by default in the resulting mapfile.

Note: In case of a raster data source output, both WMS and WCS configuration are included by default in the resulting mapfile.

Depending on the requests, ZOO-Kernel is able to return a location header and different request types:

- ResponseDocument=XXXX@asReference=true[26]

In this case, ZOO-Kernel will return the GetMap/GetFeature/GetCoverage request as KVP in the *href* of the result.

- ResponseDocument=XXXX@asReference=false[27]

In this case, ZOO-Kernel will return the result of the GetMap/GetFeature/GetCoverage request as KVP of the href used in the previous case.

[20] http://zoo-project.org
[21] http://mapserver.org
[22] http://mapserver.org/documentation.html
[23] http:/gdal.org
[24] http://mapserver.org/mapfile/index.html
[25] http://mapserver.org
[26] ResponseDocument=XXXX@asReference=true
[27] ResponseDocument=XXXX@asReference=false

- RawDataOutput=XXXX@asReference=true/false[28]

In this case, ZOO-Kernel will return the GetMap/GetFeature/GetCoverage request as KVP in a specific location header, which implies that the browser is supposed to request MapServer directly.

Whatever the default output *mimeType* returned by a WPS service is, it is used if the *useMapserver* option is found at runtime. As an example, if `<Default>` and `<Supported>` blocks are found in the ZOO Service configuration file as shown bellow, this means that the service returns GML 3.1.0 features by default.

```
<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>
```

3.3.2 Installation and configuration

Follow the step described bellow in order to activate the ZOO-Project optional MapServer support.

Prerequisites

- latest ZOO-Kernel[29] trunk version
- MapServer[30] version >= 6.0.1

First download the lastest zoo-kernel by checking out the svn. Use the following command from do the directory where your previously checked out (in this example we will use `<PREV_SVN_CO>` to design this directory).

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

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

```
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
```

Once done, compile ZOO-Kernel with MapServer support from the `<PREV_SVN_CO>` directory, using the following command:

```
cd zoo-kernel-ms
autoconf
./configure --with-python --with-mapserver=/tmp/zoo-ms-src/mapserver-6.0.1
make
```

[28] RawDataOutput=XXXX@asReference=true/false
[29] http://zoo-project.org/trac/browser/trunk/zoo-project/zoo-kernel
[30] http://mapserver/org

You can then copy the new ZOO-Kernel to `/usr/lib/cgi-bin` directory, as follow:

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

Main configuration file

Open and edit the `/usr/lib/cgi-bin/main.cfg` file, by adding the following content in the `[main]` section:

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

The `dataPath` directory is mandatory and must belong to the Apache user.

```
mkdir /var/www/temp/
chown -r apache:apache /var/www/temp/
```

A `symbols.sym` file is required in this directory. Create it and add the following content in it:

```
SYMBOLSET
SYMBOL
  NAME "circle"
  TYPE ellipse
  FILLED true
  POINTS
    1 1
  END
END
END
```

Note: Only one symbol definition is required (with any name) for the WMS service output.

The ZOO-Project optional MapServer support is activated at this step. Don't forget to add the `mapserverAddress` and `msOgcVersion` parameters to the `main.cfg` file in order to to specify the path to MapServer and the OGC WebService version used by the Services.

Warning: ZOO-kernel will segfault (checking `NULL` value should correct this behavior) if the `mapserverAddress` parameter is not found

Service configuration file

useMapserver

In order to activate the MapServer WMS/WFS/WCS output for a specific service, the `useMapserver` parameter must be added to the `<Default>` or `<Supported>` blocks of the Service *services-zcfg*. If `useMapserver=true`, this means that the output result of the Service is a GDAL compatible datasource and that you want it to be automatically published by MapServer as WMS,WFS or WCS.

When the useMapserver option is used in a `<Default>` or `<Supported>` block, then you have to know what are the corresponding mimeType:

- text/xml: Implies that the output data will be accessible through a WFS GetFeature request (default protocol version 1.1.0)

- image/tiff: Implies 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: Implies that the output data will be accessible through a WMS GetMap request (default protocol version 1.3.0). You can check the supported output mimeType by sending a GetCapabilities request to MapServer.

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).

Styling

The optional `msStyle` parameter can also be used to define a custom MapServer style block (used for vector datasource only), as follow:

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

If a WPS service outputs a one band raster file, then it is possible to add a `msClassify` parameter and set it to `true` in the output ComplexData `<Default>` or `<Supported>` nodes of its `zcfg` file. This allows ZOO-Kernel to use its own default style definitions in order to classify the raster using equivalent intervals.

```
msClassify = ....
```

Example

An example *ZOO-Service configuration file* file configured for the optional MapServer support is shown bellow:

```
<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 = application/vnd.google-earth.kmz
 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
 msClassify = ....
</Supported>
```

In this example, 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.

3.3.3 Test requests

The optional MapServer support can be tested using any service. The simple *HelloPy* Service is used in the following example requests.

Note: The following examples require a zip file containing a Shapefile (http://localhost/data/data.zip) and a tif file (http://localhost/data/demo.tif)

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&
```

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&
```

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&
```

Accessing a remote tiff as WCS GetMap Request:

```
http://localhost/cgi-bin/zoo_loader.cgi?request=Execute&service=WPS&version=1.0.0&Identifier=HelloPy&
```

3.4 Optional Orfeo Toolbox support

Orfeo Toolbox[31] provides simple to advanced algorithms for processing imagery available from remote sensors. The optional Orfeo Toolbox support is available since ZOO-Project 1.5[32]. It allows to execute the OTB Applications[33] directly as ZOO WPS Services thanks to a *ZOO-Kernel* specific internal mechanism which is detailed in this section.

Note:  Orfeo Toolbox[34] is an open source image processing library. Learn more by reading its documentation[35].

3.4.1 Installation and configuration

Follow the step described bellow in order to activate the ZOO-Project optional Orfeo Toolbox support.

Prerequisites

- latest ZOO-Kernel[36] trunk version
- Orfeo Toolbox (OTB 4.2.1[37])

[31] http://orfeo-toolbox.org/otb/
[32] http://zoo-project.org
[33] http://orfeo-toolbox.org/otb/otb-applications.html
[34] https://www.orfeo-toolbox.org
[35] https://www.orfeo-toolbox.org/documentation/
[36] http://zoo-project.org/trac/browser/trunk/zoo-project/zoo-kernel
[37] http://orfeo-toolbox.org/otb/

- Insight Segmentation and Registration Toolkit (ITK-4.7[38])

Installation steps

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

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[39] 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
```

[38] http://itk.org/ITK/resources/software.html/
[39] http://zoo-project.org/trac/browser/trunk/thirds/otb2zcfg

Test requests

Once done, OTB Applications should be listed as available WPS Services when runing 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=
```

Here is an example request executing the *OTB.BandMath* Application with the OTB Cookbook[40] sample data as input

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

Note: The usual ZOO GetStatus requests also work when using the OTB Applications as WPS Services.

3.5 Optional SAGA GIS support

SAGA GIS[41] provides a comprehensive set of geoscientific methods and spatial algorithms. The optional SAGA GIS support is available since ZOO-Project 1.5[42]. It allows to execute the SAGA Modules[43] directly as ZOO WPS Services thanks to a *ZOO-Kernel* specific internal mechanism which is detailed in this section.

Note:  SAGA GIS[44] is the System for Automated Geoscientific Analyses. Learn more on official website[45].

3.5.1 Installation and configuration

Follow the step described bellow in order to activate the ZOO-Project optional SAGA GIS support.

Prerequisites

- latest ZOO-Kernel[46] trunk version
- SAGA GIS (SAGA-GIS 2.1.4[47])
- libLAS-1.2 (LibLAS-1.2[48])

[40] https://www.orfeo-toolbox.org/CookBook/CookBook.html
[41] http://orfeo-toolbox.org/otb/
[42] http://zoo-project.org
[43] http://www.saga-gis.org/saga_module_doc/2.1.4/index.html
[44] https://www.orfeo-toolbox.org
[45] http://www.saga-gis.org/en/index.html
[46] http://zoo-project.org/trac/browser/trunk/zoo-project/zoo-kernel
[47] http://saga-gis.org
[48] https://github.com/libLAS/libLAS-1.2

Installation steps

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

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-saga=/usr/local/
make
```

And copy the newly created zoo_loader.cgi to `/usr/lib/cgi-bin` :

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

Configuration steps

Services configuration file

Building the **'saga2zcfg <http://zoo-project.org/trac/browser/trunk/thirds/otb2zcfg >'_** utility is required to activate the available SAGA-GIS Modules as WPS Services. This can be done using the following command:

```
cd thirds/saga2zcfg
make
```

The following commands will then generate all the needed zcfg files for the available SAGA-GIS Modules:

```
mkdir zcfgs
cd zcfgs
../saga2zcfg
mkdir /location/to/your/cgi-bin/SAGA
cp *zcfg /location/to/your/cgi-bin/SAGA
```

Test requests

The SAGA-GIS Modules should be listed as available WPS Services when runing a GetCapabilities request, as follow:

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

Each SAGA-GIS 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=SAGA.garden

And executed according to your needs. The following example executes *SAGA.garden_fractals.1* with no optional parameter:

http://localhost/cgi-bin/zoo_loader.cgi?request=Execute&service=WPS&version=1.0.0&Identifier=SAGA.garden_fractals

Note: The common ZOO GetStatus requests also work when using the SAGA-GIS Modules as WPS Services.

CHAPTER

FOUR

ZOO-SERVICES

This section will guide you for creating your own WPS Services using the ZOO-Project[1] platform. It also gives usefull information for taking advantage of the ready-to-use **ZOO-Services** which are available in the ZOO-Project svn.

4.1 What are ZOO-Services ?

ZOO-Services are WPS compliant Web Services working with *ZOO-Kernel*, the ZOO-Project[2] WPS server.

4.1.1 What is a ZOO-Service?

A ZOO Service is a couple composed of:

- Source code you want to create or reuse as WPS Service
- A *configuration file* (.zcfg) which describes this WPS Service

Learn how to *create your own* and how to configure ZOO-Services according to the *ZCFG Reference*.

4.1.2 Available ZOO-Services

ZOO-Project[3] includes ready-to-use WPS Services based on reliable open source libraries such as GDAL[4], GRASS GIS[5], OrfeoToolbox[6] and CGAL[7]. The so-called ZOO-Services aim at reusing existing geospatial algorithms through standard WPS, with no or minor modification of the involved software or library source codes.

Available ZOO-Services provide a number of significant examples to build your own.

4.2 ZOO-Service configuration file

The ZOO-Service configuration file (.zcfg) describes a WPS service. It provides metadata information on a particular WPS Service and it is parsed by ZOO-Kernel when *DescribeProcess* and *Execute* request are sent.

[1] http://zoo-project.org
[2] http://zoo-project.org
[3] http://zoo-project.org
[4] http://gdal.org
[5] http://grass.osgeoorg
[6] http://orfeo-toolbox.org
[7] http://gcal.org

The ZOO-Service configuration file is divided into three distinct sections :

- Main Metadata information
- List of Inputs metadata information (optional since rev. 469[8])
- List of Outputs metadata information

Warning: The ZOO-Service configuration file is case sensitive.

Note: There are many example ZCFG files in the `cgi-env` directory of the ZOO-Project svn[9].

Note: A ZCFG file can be converted to the YAML syntaxe by using the zcfg2yaml command line tool.

4.2.1 Main section

The fist part of the ZOO-Service configuration file is the `main` section, which contains general metadata information on the related WPS 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.

An example of the `main` section is given bellow as reference.

```
[Name of WPS Service]
Title = Title of the WPS Service
Abstract = Description of the WPS Service
processVersion = Version number of the WPS Service
storeSupported = true/false
statusSupported = true/false
serviceType = Pprogramming language used to implement the service (C|Fortran|Python|Java|PHP|Ruby|Jav
serviceProvider = Name of the Services provider (shared library|Python Module|Java Class|PHP Script|
<MetaData>
  title = Metadata title of the WPS Service
</MetaData>
```

Warning: 'Name of WPS Service' must be the exact same name as the function defined in the WPS Service source code.

Note: An `extend` parameter may be used for the Process profile registry.

4.2.2 List of Inputs

The second part of the ZOO-Service configuration file is the `<DataInputs>` section which lists the supported inputs. Each input is defined as :

- Name (between brackets as for the name of the service before)
- Various medata properties (`Title`, `Abstract`, `minOccurs`, `maxOccurs` and, in case of ComplexData, the optional `maximumMegabytes`)

[8] http://zoo-project.org/trac/changeset/469
[9] http://zoo-project.org/trac/browser/trunk/zoo-project/zoo-services

- *Type Of Data Node*

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

```
<DataInputs>
  [Name of the first input]
    Title = Title of the first input
    Abstract = Abstract describing the first input
    minOccurs = Minimum occurence of the first input
    maxOccurs = Maximum occurence of the first input
    <Type Of Data Node />
  [Name of the second input]
    Title = Title of the second input
    Abstract = Abstract describing the second input
    minOccurs = Minimum occurence of the second input
    maxOccurs = Maximum occurence of the second input
    <Type Of Data Node />
</DataInputs>
```

Note: A <MetaData> node can also be added, as in the main metadata information.

4.2.3 List of Outputs

The third part of the ZOO Service configuration file is the <DataOutputs> section, which lists the supported outputs and is is very similar to a list of inputs.

A typical list of outputs (<DataOutputs>) looks like the following:

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

4.2.4 Type Of Data Nodes

The *Type Of Data Nodes* describes data types for inputs and outputs. There are three different types which are described in this section.

- *LiteralData*
- *BoundingBoxData*
- *ComplexData*

Warning: Every *BoundingBoxData* and *ComplexData* must have at least one <Default> node (even empty like <Default />)

Warning: In WPS 2.0.0 version, it is possible to define nested inputs and outputs[a]. So, from revision 790[b], you are allowed to use a new input/output definition here.

[a] http://docs.opengeospatial.org/is/14-065/14-065.html#13
[b] http://www.zoo-project.org/trac/changeset/790

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 interger and so on (consult the complete list[10] 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]
```

[10] http://www.w3.org/TR/xmlschema-2/#built-in-datatypes

```
  <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 conbined 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
    rangeMin = 990
    rangeClosure = co
    rangeSpacing = 10
  </Supported>
</LiteralData>
```

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[11] and EPSG:3785[12]), looks like the following:

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

[11] http://www.epsg-registry.org/indicio/query?request=GetRepositoryItem&id=urn:ogc:def:crs:EPSG::4326
[12] http://www.epsg-registry.org/indicio/query?request=GetRepositoryItem&id=urn:ogc:def:crs:EPSG::3785

```
    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 optionaly `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:

```
<ComplexData>
  <Default>
    mimeType = application/json
    encoding = UTF-8
  </Default>
  <Supported>
    mimeType = text/xml
    encoding = base64
    schema = http://fooa/gml/3.1.0/polygon.xsd
  </Supported>
  <Supported>
    mimeType = text/xml
    encoding = UTF-8
    schema = http://fooa/gml/3.1.0/polygon.xsd
  </Supported>
</ComplexData>
```

4.3 Process profiles registry

WPS Services belonging to the same Services provider often share the same inputs and outputs. In such a case, every *ZCFG* file would contain the same metadata information and this may be a waste of time to write them all.

ZOO-Kernel is able to handle metadata inheritance from rev. 607[13], and this solves the issue of writing many ZCFG with same input and output. A registry can be loaded by the ZOO-Kernel (before any other ZCFG

[13] http://www.zoo-project.org/trac/changeset/607

files) and contain a set of Process Profiles organized in hierarchic levels according to the following rules:

- *Concept*: The higher level in the hierarchy. *Concepts* are basic text files containing an abstract description of a WPS Service (see the OGC definition[14] for more details).
- *Generic*: A *Generic* profile can make reference to *Concepts*. It defines inputs and outputs without data format or maximum size limitation (see the OGC definition[15] for more details).
- *Implementation*: An *Implementation* profile can inherit from a generic profile and make reference to concepts (see the OGC definition[16] for more details). It contains all the metadata information about a particular WPS Service (see *ZCFG reference* for more information).

Both *Generic* and *Implementation* process profiles are created from *ZCFG* files and stored in the registry sub-directories according to their level (*Concept*, *Generic* or *Implementation*).

To activate the registry, you have to add a `registry` key to the `[main]` section of your `main.cfg` file, and set its value to the directory path used to store the profile ZCFG files. Please see *Setup registry browser* for more details about the other services and parameters required.

Note: Even if the profile registry was first introduced in WPS 2.0.0, it can be also used in the same way for WPS 1.0.0 Services.

4.3.1 Generic Process Profile

A Generic Process Profile is a ZCFG file located in the `generic` sub-directory, it defines main metadata information, inputs and outputs name, basic metadata and multiplicity. It can make reference to a concept by defining a `concept` key in the main metadata information part.

You can find below the *GO.zcfg* file, a typical Generic Process Profile for Generic Geographic Operation, taking one InputPolygon input parameter and returning a result named Result, it make reference to the `GOC` concept:

```
[GO]
 Title = Geographic Operation
 Abstract = Geographic Operation on exactly one input, returning one output
 concept = GOC
 level = generic
 statusSupported = true
 storeSupported = true
 <DataInputs>
  [InputPolygon]
   Title = the geographic data
   Abstract = the geographic data to run geographipc operation
   minOccurs = 1
   maxOccurs = 1
 </DataInputs>
 <DataOutputs>
  [Result]
   Title = the resulting data
   Abstract = the resulting data after processing the operation
 </DataOutputs>
```

Note: if you need to reference more than one concept, you should separate their names with a comma (ie. concept = GO,GB),

[14] http://docs.opengeospatial.org/is/14-065/14-065.html#33
[15] http://docs.opengeospatial.org/is/14-065/14-065.html#34
[16] http://docs.opengeospatial.org/is/14-065/14-065.html#35

4.3.2 Process Implementation Profile

A Process Implementation Profile is similar to a ZCFG file located in the *implementation* sub-directory, it defines (or inherit from its parent) all the properties of a *Generic Process Profile* and specify Data Format for both inputs and outputs. It can make reference to a concept by defining a `concept` key in the main metadata information part.

You can find below the *VectorOperation.zcfg* file, a typical Process Implementation Profile for Vector Geographic Operation, it inherit from the *GP generic profile*:

```
[VectorOperation]
 Title = Vector Geographic Operation
 Abstract = Apply a Vector Geographic Operation on a features collection and return the resulting fea
 extend = GO
 level = profile
 <DataInputs>
  [InputPolygon]
   Title = the vector data
   Abstract = the vector data to run geographic operation
   <ComplexData>
    <Default>
     mimeType = text/xml
     encoding = UTF-8
     schema = http://fooa/gml/3.1.0/polygon.xsd
    </Default>
    <Supported>
     mimeType = application/json
     encoding = UTF-8
     extension = js
    </Supported>
 </DataInputs>
 <DataOutputs>
  [Result]
   Title = the resulting data
   Abstract = the resulting geographic data after processing the operation
   <ComplexData>
    <Default>
     mimeType = text/xml
     encoding = UTF-8
     schema = http://fooa/gml/3.1.0/polygon.xsd
    </Default>
    <Supported>
     mimeType = application/json
     encoding = UTF-8
     extension = js
    </Supported>
   </ComplexData>
 </DataOutputs>
```

4.3.3 ZCFG inheritance

For the ZCFG files at the service level, you can inherit the metadata from a Process Implementation Profile available in the registry. As before, you simply need to add a `extend` key refering the ZCFG you want to inherit from and a `level` key taking the *'implementation'* value to your main metadata informations.

So, for example, the original ConvexHull.zcfg[17] may be rewritten as:

```
[ConvexHull]
 Title = Compute convex hull.
 Abstract = Return a feature collection that represents the convex hull of each geometry from the inp
 serviceProvider = ogr_service.zo
 serviceType = C
 extend = VectorOperation
 level = implementation
```

Now, suppose that your service is able to return the result in KML format, then you may write the following:

```
[ConvexHull]
 Title = Compute convex hull.
 Abstract = Return a feature collection that represents the convex hull of each geometry from the inp
 serviceProvider = ogr_service.zo
 serviceType = C
 extend = VectorOperation
 level = implementation
 <DataOutputs>
  [Result]
     <Supported>
      mimeType = application/vnd.google-earth.kml+xml
      encoding = utf-8
     </Supported>
 </DataOutputs>
```

4.3.4 Setup registry browser

In the `zoo-project/zoo-services/utils/registry` you can find the source code and the `Makefile` required to build the Registry Browser Services Provider. To build and install this service, use the following comands:

```
cd zoo-project/zoo-services/utils/registry
make
cp cgi-env/* /usr/lib/cgi-bin
```

To have valid `href` in the metadata children of a `wps:Process`, you have to define the `registryUrl` to point to the path to browse the registry. For this you have two different options, the first one is to install the `GetFromRegistry` ZOO-Service and to use a WPS 1.0.0 Execute request as `registryUrl` to dynamically generate Process Concept[18], Generic Process Profile[19] and Process Implementation Profile[20]. You also have to add a `registryUrl` to the `[main]` section to inform the ZOO-Kernel that it should use the Registry Browser to create the href attribute of Metadata nodes. So by adding the following line:

```
registryUrl = http://localhost/cgi-bin/zoo_loader.cgi?request=Execute&service=WPS&version=1.0.0&Ident
```

The second option is to pre-generate each level of the hierarchy by running shell commands then set `registryUrl` to the URL to browse the generated files. In such a case, you will also have to define the `registryExt` and set it to the file extension you used to generate your registry cache.

To generate the cache in `/opt/zoo/registry/`, use the following command:

[17] http://www.zoo-project.org/trac/browser/trunk/zoo-project/zoo-services/ogr/base-vect-ops/cgi-env/ConvexHull.zcfg?rev=491
[18] http://docs.opengeospatial.org/is/14-065/14-065.html#33
[19] http://docs.opengeospatial.org/is/14-065/14-065.html#34
[20] http://docs.opengeospatial.org/is/14-065/14-065.html#35

```
cd /usr/lib/cgi-bin
mkdir /opt/zoo/regcache/{concept,generic,implementation}
for i in $(find /opt/zoo/registry/ -name "*.*") ;
do
    j=$(echo $i | sed "s:../registry//::g;s:.zcfg::g;s:.txt::g") ;
   if [ -z "$(echo $j | grep concept)" ];
   then
       ext="xml" ;
   else
       ext="txt";
   fi
    ./zoo_loader.cgi "request=Execute&service=wps&version=1.0.0&Identifier=GetFromRegistry&RawDataOut
done
```

4.4 Create your own ZOO-Services

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, C# and JavaScript with links to their corresponding `.zcfg` files.

Contents

4.4.1 General information

The function of the process for each programming language take three arguments: the main configuration, inputs and outputs.

Note: The service must return **3** if the process run successfully

Note: The service must return **4** if the process ended with an error

4.4.2 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

```
{
'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',
             'positionName': 'Developer',
             'providerName': 'Name of provider',
             'addressAdministrativeArea': 'False',
             'phoneVoice': 'False',
             'addressCity': 'City',
             'providerSite': 'http://www.your.site',
             'addressPostalCode': '38122',
             'role': 'Developer',
             'addressDeliveryPoint': 'False',
             'phoneFacsimile': 'False',
             'addressElectronicMailAddress': 'your@email.com',
             'individualName': 'Your Name'
            }
}
```

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):
   outputs["Result"]["value"]="Hello "+inputs["a"]["value"]+" from Python World !"
   return zoo.SERVICE_SUCCEEDED
```

4.4.3 PHP

ZOO-API

The ZOO-API for the PHP language is automatically available from your service code. Tthe 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[21])

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 (ie. 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

```
<?
function HelloPHP(&$main_conf,&$inputs,&$outputs){
   $tmp="Hello ".$inputs[S][value]." from PHP world !";
   $outputs["Result"]["value"]=zoo_Translate($tmp);
   return zoo_SERVICE_SUCCEEDED();
```

[21] http://www.gnu.org/software/libc/manual/html_node/Locating-gettext-catalog.html#index-textdomain

```
}
?>
```

4.4.4 Java

Specifically for the Java support, you may add the following three sections to your `main.cfg` file:

[java] This section is used to pass -D* parameters to the JVM created by the ZOO-Kernel to handle your ZOO-Service (see ref. 1[22] or ref. 2[23] for sample available). For each map `a = b` available in the `[java]` section, the option `-Da=b` will be passed to the JVM.

[javax] The section is used to pass -X* options to the JVM (see ref.[24]). For each map `a = b` available in the `[javax]` section, the option `-Xab` will be passed to the JVM (ie. set `mx=2G` to pass `-Xmx2G`).

[javaxx] This section is used to pass -XX:* parameters to the JVM created by the ZOO-Kernel to handle your ZOO-Service (see ref. 1[25] or ref. 2[26] for sample available). For each map `a = b` available in the `[javaxx]` section, the option `-XX:a=b` will be passed to the JVM. In case of a map `a = minus` (respectively `a=plus`) then the option `-XX:-a` (respectivelly `-XX:+a`) will be passed.

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
```

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

You should copy the `libZOO.so` in a place Java can find it. In case you have defined the `java.library.path` key as `/usr/lib/cgi-bin` (in the `[java]` section), then you should copy it there.

```
cp libZOO.so /usr/lib/cgi-bin
```

The ZOO-API provides the following functions:

String translate(String s) This function call the internal ZOO-Kernel function responsible for searching a translation of `s` in the zoo-services dictionary.

void updateStatus(Hashmap conf,String pourcent,String message) This function call the updateStatus ZOO-Kernel function responsible for updating the status of the running service (only usefull when the service has been called asynchronously).

[22] http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html#BehavioralOptions
[23] http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html#PerformanceTuning
[24] http://docs.oracle.com/cd/E22289_01/html/821-1274/configuring-the-default-jvm-and-java-arguments.html
[25] http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html#BehavioralOptions
[26] http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html#PerformanceTuning

Java ZCFG requirements

Note: For each Service provided by your ZOO Java Services Provider (your corresponding Java class), the ZCFG File should have the name of the Java public method corresponding to the service (case-sensitive).

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 `HelloJava.class` then you should use `HelloJava`.

Java Data Structure used

The three parameters are passed to the Java function as java.util.HashMap[27].

Sample ZOO Java Services Provider

```
import java.util.*;
public class HelloJava {
  public static int HelloWorldJava(HashMap conf,HashMap inputs, HashMap outputs) {
     HashMap hm1 = new HashMap();
     hm1.put("dataType","string");
     HashMap tmp=(HashMap)(inputs.get("S"));
     java.lang.String v=tmp.get("value").toString();
     hm1.put("value","Hello "+v+" from JAVA WOrld !");
     outputs.put("Result",hm1);
     System.err.println("Hello from JAVA WOrld !");
     return ZOO.SERVICE_SUCCEEDED;
  }
}
```

4.4.5 C#

Specifically for the C# support, you should add the following section to your `main.cfg` file.

[mono] This section is used to define both `libPath` and `etcPath` required by the Mono .NET Framework.

ZOO-API

Before you build your first ZOO-Service implemented in Mono, you should first build the `ZMaps.dll` containing the Mono ZOO-API.

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

```
cd zoo-api/mono
make
```

[27] http://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html

Then you should copy the `ZMaps.dll` in your `servicePath` or in the directory where your `zoo_loader.cgi` file is stored.

The ZOO-API is available from a C# class named ZOO_API and provides the following static variables:

int SERVICE_SUCCEEDED Value to return in case your service end successfully.

int SERVICE_FAILED Value to retrun in case of failure.

The ZOO-API provides the following static functions:

string Translate(String s) This function call the internal ZOO-Kernel function responsible for searching a translation of `s` in the zoo-services dictionary.

void UpdateStatus(ZMaps conf,String pourcent,String message) This function call the updateStatus ZOO-Kernel function responsible for updating the status of the running service (only usefull when the service has been called asynchronously).

C# ZCFG requirements

Note: For each Service provided by your ZOO Mono Services Provider (your corresponding Mono class), the ZCFG File should have the name of the Mono public static function corresponding to the service (case-sensitive).

The ZCFG file should contain the following :

serviceType Mono

serviceProvider The full name of the C# dll containing the ZOO-Service Provider (including `.dll`).

serviceNameSpace The namespace of the C# class containing the ZOO-Service Provider.

serviceClass The name of the C# class containing the ZOO-Service Provider definition.

C# Data Structure used

The three parameters of the function are passed to the Mono static function as `ZMaps` which are basically `Dictionary<String,_ZMaps>`.

Sample ZOO C# Services Provider

```
using System;
using ZooGenerics;
using System.Threading;

namespace Default
{
    public class Service{
        public static int HelloMono(ZMaps conf,ZMaps inputs,ZMaps outputs){
            _ZMaps test;
            if(inputs.TryGetValue("a", out test)){
                ZMap content=test.getContent();
                String test1;
                if(content.TryGetValue("value", out test1)){
                    outputs.setMapsInMaps("Result","value",ZOO_API.Translate("Hello ")+test1+" from t
                }
                return ZOO_API.SERVICE_SUCCEEDED;
```

```
            }else{
                conf.setMapsInMaps("lenv","message","Unable to run the service");
                return ZOO_API.SERVICE_FAILED;
            }
        }
        public static int longProcessMono(ZMaps conf,ZMaps inputs,ZMaps outputs){
            _ZMaps test;
            int i=1;
            while(i<10){
                ZOO_API.UpdateStatus(conf,"Step "+i,(i*10));
                Thread.Sleep(1000);
                i+=1;
            }
            if(inputs.TryGetValue("a", out test)){
                ZMap content=test.getContent();
                String test1;
                if(content.TryGetValue("value", out test1)){
                    outputs.setMapsInMaps("Result","value",ZOO_API.Translate("Hello ")+test1+" from t
                }
                return ZOO_API.SERVICE_SUCCEEDED;
            }else{
                conf.setMapsInMaps("lenv","message","Unable to run the service");
                return ZOO_API.SERVICE_FAILED;
            }
        }
    };
}
```

4.4.6 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

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

4.5 Translation Support

ZOO-Kernel support translating internal messages it emits but it can also translate both the metadata informations stored in the ZCFG file and the messages emitted by the ZOO-Service itself. This document show how to create the files required to handle such a translation process for the ZOO-Services.

4.5.1 ZCFG translation

First of all, use the following commands from your Services Provider directory in order to extract all the messages to translate from the ZCFG files :

```
    #!/bin/bash
    mkdir -p locale/{po,.cache}
    for j in cgi-env/*zcfg ;
      do
        for i in Title Abstract;
         do
          grep $i $j | sed "s:$i = :_ss(\":g;s:$:\"):g" ;
         done;
     done > locale/.cache/my_service_string_to_translate.c
```

Then generate the `messages.po` file based on the Services Provider source code (located in `service.c` in this example) using the following command:

```
    #!/bin/bash
    xgettext service.c locale/.cache/my_service_string_to_translate.c -o message.po -p locale/po/ -k
```

Once `messages.po` is created, use the following command to create the `.po` file for the targeted language to translate into. We will use the French language here as an example:

```
    #!/bin/bash
    cd locale/po/
    msginit -i messages.po -o zoo_fr_FR.po -l fr
```

Edit the `zoo_fr_FR.po` file with your favorite text editor or using one of the following tools:

- poedit[28]
- virtaal[29]
- transifex[30]

Once the `zoo_fr_FR.po` file is completed, you can generate and install the corresponding `.mo` file using the following command:

[28] http://www.poedit.net/
[29] http://translate.sourceforge.net/wiki/virtaal/index
[30] https://www.transifex.net/

```
#!/bin/bash
msgfmt locale/po/zoo_fr_FR.po -o /usr/share/locale/fr/LC_MESSAGES/zoo-services.mo
```

In order to test the Services Provider ZCFG and internal messages translation, please add the language argument to you request. As an example, such a request:

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

would become the following:

http://youserver/cgi-bin/zoo_loader.cgi?request=GetCapabilities&service=WPS&language=fr-FR

The following command may also be useful in order to pull all the translations already available for a specific language.

```
#!sh
msgcat -o compilation.po $(find ../../ -name fr_FR.utf8.po)
msgfmt compilation.po -o /usr/share/locale/fr/LC_MESSAGES/zoo-services.mo
```

4.6 ZOO Status Service

The ZOO-Status Service is a ZOO-Project[31] utility allowing to get the status of a running WPS Service.

4.6.1 Description

It returns the stage of completion of the ongoing Service in percentage (%). The ZOO-Status Service is usefull to monitor *ZOO-Services*. It can also be used to animate WPS progress bars from client-side applications.

4.6.2 Installation

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):

```
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` :

```
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:

```
cp /path/to/zoo/source/zoo-services/utils/status/cgi-env/*{xsl} /var/www/data
```

[31] http://zoo-project.org

4.7 Debugging ZOO Services

Several methods can be used in order to debug *ZOO-Services*. The most common solutions are web or command line.

4.7.1 Web

Any problem can be checked in the Apache server log file when using http WPS requests.

On Unix, the log files is usually located in `/var/log/apache2` and the relevant one is named *error_log*. A simple way to read this file is to use the `tail` command, as it allows to see the file updates for each request

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

If the log is not clear enough, you still have the possibility to add more debug information to your source code, writing to standard errors.

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

Using JavaScript, you can use `alert` to print a string to standard error, for example:

```
// add this line when you want to see 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]`

4.7.2 Command line

ZOO-Kernel (*zoo_loader.cgi*) can also be used from command line. This is really useful for debugging services in a deeper way, for example:.

```
# 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.

In case you should simulate POST requests, you can use the following command to tell the ZOO-Kernel to use the file `/tmp/req.xml` as the input XML request:

```
# Define required environment settings
export REQUEST_METHOD=POST
export CONTENT_TYPE=text/xml
# Run the request stored in a file
./zoo_loader.cgi < /tmp/req.xml
```

GDB

From command line you can use also the command line tool GDB[32] 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&responsedocu
```

Note: You can use the same parameter used before to simulate POST requests when running from gdb.

If nothing helped, you can ask help at the ZOO mailing list[33] copying the result of the command.

Python

For Python, you can use `pdb`, more info at 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

4.8 Available ZOO-Services

ZOO-Project[34] includes some ready-to-use WPS Services based on reliable open source libraries such as such as GDAL[35], CGAL[36], GRASS GIS[37], OrfeoToolbox[38] and SAGA GIS[39].

ZOO-Services are either developed in C/Python (with minor modifications with respect to the orginal software source code) and stored in the `zoo-services` svn[40] directory or automatically generated using some of *ZOO-Kernel* configuration options.

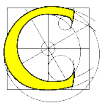
[32] http://www.gnu.org/software/gdb/
[33] http://lists.osgeo.org/cgi-bin/mailman/listinfo/zoo-discuss
[34] http://zoo-project.org
[35] http://gdal.org
[36] http://gcal.org
[37] http://grass.osgeo.org
[38] http://orfeo-toolbox.org
[39] https://www.orfeo-toolbox.org
[40] http://zoo-project.org/trac/browser/trunk/zoo-project/zoo-services

4.8.1 Based on GDAL

Note:  GDAL[41] is the Geospatial Data Abstraction Library. Learn more on official website[42].

Name	Description	Language
Gdal_Contour[43]	Builds vector contour lines from a raster elevation model	C
Gdal_Grid[44]	Creates regular raster grid from the scattered data read from an OGR datasource	C
Gdal_Dem[45]	Provides tools to analyze raster elevation model	C
Gdal_Ndvi[46]	Computes Normalized Difference Vegetation Index on a raster file	Python
Gdal_Profile[47]	Fetches XYZ values of a raster DEM along a linestring	C
Gdal_Translate[48]	Converts raster data between different formats	C
Gdal_Warp[49]	Mosaic/Reproject/Warp a raster image	C
Ogr2Ogr[50]	Converts vector data from one format to another	C
Base-vect-ops[51]	Provides tools for single and multiple geometries vector-based spatial analysis	C
Base-vect-ops[52]	Provides tools for single and multiple geometries vector-based spatial analysis	Python

4.8.2 Based on CGAL

Note:  CGAL[53] is the Computational Geometry Algorithms Library. Learn more on official website[54].

[41] http://gdal.org
[42] http://gdal.org
[43] http://zoo-project.org/trac/browser/trunk/zoo-project/zoo-services/gdal/contour
[44] http://zoo-project.org/trac/browser/trunk/zoo-project/zoo-services/gdal/grid
[45] http://zoo-project.org/trac/browser/trunk/zoo-project/zoo-services/gdal/dem
[46] http://zoo-project.org/trac/browser/trunk/zoo-project/zoo-services/gdal/ndvi
[47] http://zoo-project.org/trac/browser/trunk/zoo-project/zoo-services/gdal/profile
[48] http://zoo-project.org/trac/browser/trunk/zoo-project/zoo-services/gdal/translate
[49] http://zoo-project.org/trac/browser/trunk/zoo-project/zoo-services/gdal/translate
[50] http://zoo-project.org/trac/browser/trunk/zoo-project/zoo-services/ogr/ogr2ogr/
[51] http://zoo-project.org/trac/browser/trunk/zoo-project/zoo-services/ogr/base-vect-ops
[52] http://zoo-project.org/trac/browser/trunk/zoo-project/zoo-services/ogr/base-vect-ops-py
[53] http://gdal.org
[54] http://cgal.org

Name	Description	Language
Cgal_Delaunay[55]	Computes the edges of Delaunay triangulation for a set of data points	C
Ggal_Voronoi[56]	Computes the edges of Voronoi diagram for a set of data points	C

4.8.3 Based on GRASS GIS

Note:  GRASS GIS[57] is the Geographic Resources Analysis Support System. Learn more on official website[58].

Name	Description	Language
Raster modules (r.*)[59]	Most of the GRASS7 vector modules are supported	C
Vector modules (v.*)[60]	Most of the GRASS7 vector modules are supported	C
Imagery modules (i.*)[61]	Most of the GRASS7 iamgery modules are supported	C

GRASS GIS 7[62] modules can be used as *ZOO-Services* without any modification using the wps-grass-bridge[63] library. The latter includes useful tools such as GrassXMLtoZCFG.py[64] and ZOOGrassModuleStarter.py[65] for using the supported GRASS modules directly as ZOO-Services. A step-by-step installation guide suited for ZOO-Project is available in the wps-grass-bridge[66] documentation.

4.8.4 Based on Orfeo Toolbox

Note:  Orfeo Toolbox[67] is an open source image processing library. Learn more on official website[68].

Orfeo Toolbox[69] Applications[70] can be used as *ZOO-Services* without any modification using the *Optional Orfeo Toolbox support*.

[55] http://zoo-project.org/trac/browser/trunk/zoo-project/zoo-services/cgal/delaunay.c
[56] http://zoo-project.org/trac/browser/trunk/zoo-project/zoo-services/cgal/voronoi.c
[57] http://grass.osgeo.org
[58] http://grass.osgeo.org
[59] http://grass.osgeo.org/grass70/manuals/raster.html
[60] http://grass.osgeo.org/grass70/manuals/vector.html
[61] http://grass.osgeo.org/grass70/manuals/imagery.html
[62] http://grass.osgeo.org
[63] https://code.google.com/p/wps-grass-bridge/
[64] https://code.google.com/p/wps-grass-bridge/source/browse/trunk/GrassXMLtoZCFG.py
[65] https://code.google.com/p/wps-grass-bridge/source/browse/trunk/ZOOGrassModuleStarter.py
[66] https://code.google.com/p/wps-grass-bridge/wiki/ZOO_WPS_Integration
[67] https://www.orfeo-toolbox.org
[68] https://www.orfeo-toolbox.org
[69] https://www.orfeo-toolbox.org
[70] http://otbcb.readthedocs.org/en/latest/Applications.html

4.8.5 Based on SAGA GIS

Note:  SAGA GIS[71] is the System for Automated Geoscientific Analyses. Learn more on official website[72].

[71] https://www.orfeo-toolbox.org
[72] http://www.saga-gis.org/en/index.html

CHAPTER

FIVE

ZOO-API

This section provides information on **ZOO-API**, the ZOO-Project[1] server-side JavaScript API.

5.1 What is ZOO-API ?

ZOO-API is a server-side Javascript library for creating and chaining *ZOO-Services*. It lets you script on the server-side to execute WPS *Processes*, and thus to use common JavaScript controls and logic for WPS chaining.

5.1.1 Server-side JavaScript WPS

ZOO-API *JavaScript Support (Optional)* works on the server-side using the Mozilla foundation JavaScript[2] engine, SpiderMonkey[3]. It uses a Proj4js[4] adaptation for server-side reprojection. It also allows to easily convert vector formats (such as GML[5], KML[6], GeoJSON[7], etc).

5.2 Using ZOO-API

This section will help you to get started using *ZOO-API*.

5.2.1 Prerequisites

ZOO-API relies on the following software:

- *ZOO-Kernel* (http://zoo-project.org), the ZOO-Project WPS Server.
- SpiderMonkey (https://developer.mozilla.org/en/SpiderMonkey), the Mozilla JavaScript[8] engine.

Warning: The ZOO-Kernel *optional JavaScript support* is required for using ZOO-API

[1] http://zoo-project.org
[2] https://developer.mozilla.org/en/JavaScript
[3] https://developer.mozilla.org/en/SpiderMonkey
[4] http://proj4js.org/
[5] http://www.opengeospatial.org/standards/gml
[6] http://www.opengeospatial.org/standards/kml
[7] http://geojson.org/geojson-spec.html
[8] https://developer.mozilla.org/en/JavaScript

5.2.2 Download

- zoo-api.js[9]
- zoo-proj4js.js[10]

If you did not *download* the ZOO-Project source code already, please proceed to a svn checkout with the following command:

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

5.3 ZOO-API Classes

The following classes are available in the ZOO API:

5.3.1 ZOO

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

Constants

NAME	DESCRIPTION
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

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

Constants

SERVICE_ACCEPTED `{Integer}` used for

SERVICE_STARTED `{Integer}` used for

SERVICE_PAUSED `{Integer}` used for

[9] http://zoo-project.org/trac/export/1/trunk/zoo-api/js/ZOO-api.js
[10] http://zoo-project.org/trac/export/1/trunk/zoo-api/js/ZOO-proj4js.js

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

5.3.2 ZOO.Format.WPS

Read/Write WPS.

Inherits from

- ZOO.Format

Functions and Properties

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

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

namespaces `{Object}` 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.

5.3.3 ZOO.Process

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

Properties and Functions

NAME	DESCRIPTION
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
Execute	Query the OGC's Web PRocessing Servcie to Execute the process.
buildInput	Object containing methods to build WPS inputs.
buildInput.complex	Given an E4XElement representing the WPS complex data input.
buildInput.reference	Given an E4XElement representing the WPS reference input.
buildInput.literal	Given an E4XElement representing the WPS literal data input.
buildDataInput-sNode	Method to build the WPS DataInputs element.

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 Servcie 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 indetifier
`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 indetifier
`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 indetifier
`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.

5.3.4 ZOO.Request

Contains convenience methods for working with ZOORequest which replace XMLHttpRequest.

Functions

NAME	DESCRIPTION
GET	Send an HTTP GET request.
POST	Send an HTTP POST request.

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.

5.4 Examples

This section gathers sample scripts using *ZOO-API*, the ZOO-Project[11] server-side JavaScript API.

ZOO-API contains many classes and functions. You can find the description list *here*.

5.4.1 ZOO.Process example

```
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_SUCEEDED`, `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`).

[11] http://zoo-project.org

5.4.2 ZOO.UpdateStatus example

```
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 helpfull when the ZOO Kernel will run your JavaScript Service in background mode (if the user set to `true` the `storeExecuteResponse` parameter in his request).

CHAPTER

SIX

ZOO-CLIENT

This section provides information on **ZOO-Client**, the ZOO-Project[1] WPS JavaScript client.

6.1 What is ZOO-Client ?

ZOO-Client is a client-side JavaScript API which provides simple methods for interacting with WPS[2] server from web applications. It is helpful for sending requests to any WPS compliant server (such as *ZOO-Kernel*) and to parse the output responses using simple JavaScript.

6.1.1 JavaScript

ZOO-Client relies on modern JavaScript libraries and can be seamlessly integrated in new or existing web platforms or applications. ZOO-Client works by expanding the tags available in WPS specific templates using values provided by a JavaScript hash or object. It allows to build valid WPS requests and to send them to a WPS server. It also provides functions to easily parse and reuse the output XML responses. Read the *next section* to get started.

Please, refer to the ZOO-Client API documentation[3] for accessing the up-to-date documentation.

6.1.2 Templates

ZOO-Client uses logic-less Mustache[4] templates for creating well-formed WPS requests. Please, refer to the ZOO-Client API documentation[5] for more details about the functions using the templates.

GetCapabilities

GetCapabilities requests are created using the following template:

```
<wps:GetCapabilities xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:wps="http://www.opengis.net/wps
    <wps:AcceptVersions>
        <ows:Version>1.0.0</ows:Version>
    </wps:AcceptVersions>
</wps:GetCapabilities>
```

[1] http://zoo-project.org
[2] http://www.opengeospatial.org/standards/wps/
[3] http://www.zoo-project.org/jsDoc/index.html
[4] http://mustache.github.io/
[5] http://www.zoo-project.org/jsDoc//module-wpsPayload.html

DescribeProcess

DescribeProcess requests are created using the following template:

```
<DescribeProcess xmlns="http://www.opengis.net/wps/1.0.0" xmlns:ows="http://www.opengis.net/ows/1.1"
{{#identifiers}}
  <ows:Identifier>{{.}}</ows:Identifier>
{{/identifiers}}
</DescribeProcess>
```

Execute

Execute requests are created using a more complex template, as shown bellow:

```
<wps:Execute service="WPS" version="1.0.0" xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:ows="ht
../wpsExecute_request.xsd" language="{{language}}">
  <!-- template-version: 0.21 -->
  <ows:Identifier>{{Identifier}}</ows:Identifier>
  <wps:DataInputs>
{{#DataInputs}}
{{#is_literal}}
    <wps:Input>
      <ows:Identifier>{{identifier}}</ows:Identifier>
      <wps:Data>
        <wps:LiteralData{{#dataType}} dataType="{{dataType}}"{{/dataType}}>{{value}}</wps:LiteralData
      </wps:Data>
    </wps:Input>
{{/is_literal}}
{{#is_bbox}}
    <wps:Input>
      <ows:Identifier>{{identifier}}</ows:Identifier>
      <wps:Data>
        <wps:BoundingBoxData ows:crs="{{crs}}" ows:dimensions="{{dimension}}">
          <ows:LowerCorner>{{lowerCorner}}</ows:LowerCorner>
          <ows:UpperCorner>{{upperCorner}}</ows:UpperCorner>
        </wps:BoundingBoxData>
      </wps:Data>
    </wps:Input>
{{/is_bbox}}
{{#is_complex}}
{{#is_reference}}
{{#is_get}}
    <wps:Input>
      <ows:Identifier>{{identifier}}</ows:Identifier>
      <wps:Reference xlink:href="{{href}}"{{#schema}} schema="{{shema}}"{{/schema}}{{#mimeType}} mime
    </wps:Input>
{{/is_get}}
{{#is_post}}
    <wps:Input>
      <ows:Identifier>{{identifier}}</ows:Identifier>
      <wps:Reference xlink:href="{{href}}" method="{{method}}" {{#schema}} schema="{{shema}}"{{/scher
{{#headers}}
        <wps:Header key="{{key}}" value="{{value}}" />
{{/headers}}
        <wps:Body>{{{value}}}</wps:Body>
      </wps:Reference>
    </wps:Input>
{{/is_post}}
```

```
{{/is_reference}}
{{^is_reference}}
    <wps:Input>
      <ows:Identifier>{{identifier}}</ows:Identifier>
      <wps:Data>
        <wps:ComplexData{{#schema}} schema="{{shema}}"{{/schema}}{{#mimeType}} mimeType="{{mimeType}}
      </wps:Data>
    </wps:Input>
{{/is_reference}}
{{/is_complex}}
{{/DataInputs}}
  </wps:DataInputs>
  <wps:ResponseForm>
{{#RawDataOutput}}
{{#DataOutputs}}
    <wps:RawDataOutput {{#mimeType}}mimeType="{{mimeType}}"{{/mimeType}}>
      <ows:Identifier>{{identifier}}</ows:Identifier>
    </wps:RawDataOutput>
{{/DataOutputs}}
{{/RawDataOutput}}
{{^RawDataOutput}}
    <wps:ResponseDocument{{#storeExecuteResponse}} storeExecuteResponse="{{storeExecuteResponse}}"{{/
{{#DataOutputs}}
{{#is_literal}}
      <wps:Output{{#dataType}} dataType="{{dataType}}"{{/dataType}}{{#uom}} uom="{{uom}}"{{/uom}}>
        <ows:Identifier>{{identifier}}</ows:Identifier>
      </wps:Output>
{{/is_literal}}
{{^is_literal}}
      <wps:Output{{#asReference}} asReference="{{asReference}}"{{/asReference}}{{#schema}} schema="{
        <ows:Identifier>{{identifier}}</ows:Identifier>
      </wps:Output>
{{/is_literal}}
{{/DataOutputs}}
    </wps:ResponseDocument>
{{/RawDataOutput}}
  </wps:ResponseForm>
</wps:Execute>
```

6.2 Using ZOO-Client

This section will help you to get started using *ZOO-Client*.

6.2.1 Prerequisites

ZOO-Client is based on the following Javascript libraries

- jQuery (http://www.jquery.com)
- x2js (https://code.google.com/p/x2js[6])
- Require.js (http://requirejs.org[7])

[6] https://code.google.com/p/x2js/
[7] http://requirejs.org/

- Hogan.js (http://twitter.github.io/hogan.js)
- query-string (https://github.com/sindresorhus/query-string[8])

Warning: Node.js[a] is also required on your system for compiling ZOO-Client templates.

[a] http://nodejs.org/

6.2.2 Download

If you did not *download* the ZOO-Project source code already, please proceed to a svn checkout with the following command:

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

Warning: You do not necessarily need to *install* the ZOO-Project server for using ZOO-Client. The corresponding svn directory[a] is needed only.

[a] http://zoo-project.org/trac/browser/trunk/zoo-project/zoo-client

6.2.3 Compiling ZOO-Client templates

In order to work with ZOO-Client, you will first need to compile the provided Mustache[9] templates using Node.js[10]. The ZOO-Client templates are located in the `/zoo-project/zoo-client/lib/tpl` directory downloaded from svn.

- Install Node.js (see related documentation[11].)
- Install Hogan, the JavaScript templating engine, using the following command:

```
sudo npm install hogan
```

- Use Hulk (Hogan's command line utility) for compiling the tempaltes using the following command:

```
hulk zoo-client/lib/tpl/*mustache > \ zoo-client/lib/js/wps-client/payloads.js
```

Warning: Using different versions of Hogan to compile and to use in a web application may lead to compatibility issue.

Everything is now ready to work with *ZOO-Client*. Read the *next section* for an example JavaScript application.

6.2.4 Building ZOO-Client API documentation

You may also build the ZOO-Client API documentation using jsDoc[12], with the following command:

```
npm install jsdoc
~/node_modules/.bin/jsdoc zoo-client/lib/js/wps-client/* -p
```

[8] https://github.com/sindresorhus/query-string/
[9] http://mustache.github.io/
[10] http://nodejs.org/
[11] https://github.com/joyent/node/wiki/Installing-Node.js-via-package-manager
[12] http://usejsdoc.org/

This will build HTML documentation in a new directory named `/out` in your working directory.

Note: Building the ZOO-Client API documentation is optional, please refer to the up-to-date ZOO-Client API Documentation[13] for the current API version.

6.3 Example application

This section gives a detailed example of ZOO-Client based JavaScript appliclation.

Note: For this example application, first setup a `/zoo-client-demo` directory accessible from your web server at *http://localhost/zoo-client-demo*.

The following subdirectories must be created in the `/zoo-client-demo` directory:

```
assets
assets/js
assets/js/lib
assets/js/lib/hogan
assets/js/lib/jquery
assets/js/lib/query-string
assets/js/lib/xml2json
assets/js/lib/zoo
assets/tpl
```

You will need to copy your node_modules javascript files copied in the *hogan* and *query-string* directories. First, you wil 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.

6.3.1 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:

```
<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.

```
1 requirejs.config({
2     baseUrl: 'assets/js',
3     paths: {
4         jquery: 'lib/jquery/jquery-1.11.0.min',
5         hogan: 'lib/hogan/hogan-3.0.2',
6         xml2json: 'lib/xml2json/xml2json.min',
```

[13] http://www.zoo-project.org/jsDoc/index.html

```
7           queryString: 'lib/query-string/query-string',
8           wpsPayloads: 'lib/zoo/payloads',
9           wpsPayload: 'lib/zoo/wps-payload',
10          utils: 'lib/zoo/utils',
11          zoo: 'lib/zoo/zoo',
12          domReady: 'lib/domReady',
13          app: 'first-app',
14      },
15      shim: {
16          wpsPayloads: {
17              deps: ['hogan'],
18          },
19          wpsPayload: {
20              deps: ['wpsPayloads'],
21              exports: 'wpsPayload',
22          },
23          hogan: {
24              exports: 'Hogan',
25          },
26          xml2json: {
27            exports: "X2JS",
28          },
29          queryString: {
30              exports: 'queryString',
31          },
32      },
33  });
34
35  requirejs.config({
36      config: {
37          app: {
38              url: '/cgi-bin/zoo_loader.cgi',
39              delay: 2000,
40          }
41      }
42  });
43
44  require(['domReady', 'app'], function(domReady, app) {
45      domReady(function() {
46          app.initialize();
47      });
48  });
```

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[14] 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.

```
1   define([
2       'module','zoo','wpsPayload'
3   ], function(module, ZooProcess, wpsPayload) {
4
5       var myZooObject = new ZooProcess({
6           url: module.config().url,
7           delay: module.config().delay,
```

[14] http://github.com/requirejs/domReady

```
8       });
9
10      var initialize = function() {
11          self = this;
12          myZooObject.getCapabilities({
13             type: 'POST',
14               success: function(data){
15                       console.log(data);
16              }
17          });
18
19          myZooObject.describeProcess({
20              type: 'POST',
21              identifier: "all",
22              success: function(data){
23                  console.log(data);
24              }
25          });
26
27         myZooObject.execute({
28             identifier: "Buffer",
29             dataInputs: [{"identifier":"InputPolygon","href":"XXX","mimeType":"text/xml"}],
30             dataOutputs: [{"identifier":"Result","mimeType":"application/json","type":"raw"}],
31             type: 'POST',
32             success: function(data) {
33                  console.log(data);
34             },
35             error: function(data){
36                  console.log(data);
37             }
38          });
39      }
40
41      // Return public methods
42      return {
43          initialize: initialize
44      };
45
46  });
```

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.

CHAPTER

SEVEN

CONTRIBUTOR GUIDE

This is the ZOO-Project[1] **Contributor Guide**. This document provides information and guidelines to anyone willing to contribute to the ZOO-Project[2] open source software project and help making it better.

7.1 How to contribute ?

Please consider the following simple rules if you would like to contribute to the ZOO-Project[3] open source software.

7.1.1 Community

Anybody is welcome to share and contribute ideas, code, documentation or any relevant resource. This should be done according to the directives stated in the ZOO-Project *Contributor Guide*.

Governance

ZOO-Project[4] activities are directed by the Project Steering Committee (PSC) and the software itself is being developed, maintained and documented by an international community of users and developers (aka ZOO-Tribe[5]).

Contributions are moderated and integrated in trunk at the discretion of the ZOO-Project[6] PSC. Commit access are usually granted to active contributors by vote from the PSC members.

Licensing

ZOO-Project[7] source code is open source and made available under the MIT/X-11[8] license[9]. You must agree to the terms of that same license when creating, submiting and releasing new source code.

[1] http://zoo-project.org
[2] http://zoo-project.org
[3] http://zoo-project.org
[4] http://zoo-project.org
[5] http://zoo-project.org/new/ZOO-Project/ZOO%20Tribe
[6] http://zoo-project.org
[7] http://zoo-project.org
[8] http://opensource.org/licenses/MITlicense
[9] http://zoo-project.org/trac/browser/trunk/zoo-project/LICENSE

ZOO-Project[10] documentation is open source and made available under the Creative Commons Attribution-ShareAlike 4.0 International Public License[11] . You must agree to the terms of that same license when creating, submiting and releasing a new documentation file.

7.1.2 Available media

Discussions and contributions to ZOO-Project[12] are encouraged using the following public media.

Mailing lists

Feel free to post any question, feedback, comment or idea to the general public mailing list. For project managment or governance topics, please use the PSC list.

Name	Description
zoo-discuss[13]	General mailing list for ZOO-Project users and developers
zoo-psc[14]	Project Steering Committe mailing list

IRC

Join the **#zoo-project** channel on irc.freenode.net to discuss with the ZOO-Tribe at any time.

Tracker

Bug reports and code patches should be shared using the ZOO-Project[15] bug tracking system, as specified in the contribute-code section.

Wiki

Wiki pages can also be created by registered users. They can be used in order to describe any concept, contribution and or action that benefits or is related to the project.

7.2 Contribute code

Anybody can take part to the ZOO-Project[16] developement and is welcome to:

- Share new source code or correction
- Create tickets to report bugs
- Write a new feature request.

[10] http://zoo-project.org
[11] https://creativecommons.org/licenses/by-sa/4.0/legalcode
[12] http://zoo-project.org
[13] http://lists.osgeo.org/cgi-bin/mailman/listinfo/zoo-discuss
[14] http://lists.osgeo.org/cgi-bin/mailman/listinfo/zoo-psc
[15] http://zoo-project.org
[16] http://zoo-project.org

7.2.1 Submit new code

For new comers

New source code or existing source code corrections (patches) should be submitted using the ZOO-Project bug tracking system (ZOO-Trac[17]).

Create a new ticket[18] in order to describe your code or patch and attach it to the ticket (attach all the files required to use your code or patch). It will then be checked and discussed with the developers, and can potentially be integrated and merged with the trunk.

For registered developers

ZOO-Project registered developers have direct svn access and can:

- Commit fixes, enhancement and new source directly to trunk
- Create and commit to a new branch of the svn

ZOO-Project registered developers must accept and respect the *Committer guidelines* when contributing code.

7.2.2 Bug tracking

General information

Bug reports and wishes can be submitted using the ZOO-Trac[19] . This requires you to setup a user account (userid) using this section.

The following trackers are available:

- *defects* to report bugs and 'bad' features
- *enhancement* to describe feature wishes
- *task* to describe any different but relevant topic.

The following components are available:

- *Developemnt platform* to report bugs and 'bad' features
- *ZOO-Kernel* to report a bug or problem with the ZOO-Project WPS server
- *ZOO-Services* to report a bug or problem with the ZOO-Project WPS services
- *ZOO-API* to report a bug or problem with the ZOO-Project API
- *ZOO-Client* to report a bug or problem with the ZOO-Project Client
- *Documentation* to report a problem or suggest an enhancement to the documentation

[17] http://zoo-project.org/trac
[18] http://zoo-project.org/trac/newticket
[19] http://zoo-project.org/trac

Best practices

Please consider the following when submitting bugs or feature requests:

- Check if the bug is still persistent in svn trunk before reporting. If you use an older version, please consider upgrading.
- Before reporting a bug, please search if it is yet unknown in the bug tracking system.
- Give an appropriate, starightforward and understandable title to your ticket using the *Summary* field
- Make sure the developers get all the needed information to recreate the bug using the *Description* field (e.g. tell about your configuration and explain every step to reproduce the bug).
- Select at least a *Type* of tracker and a *Component* for your new ticket.
- Report only one single bug by ticket.

7.3 Contribute documentation

ZOO Documentation is a collaborative process managed by the ZOO developers. Anybody is welcome to contribute to the ZOO-Project documentation. Please consider the following instructions before doing so.

7.3.1 General information

Heading syntaxe

Tere are various title heading used in the documentation, when you create a new document, you're invited to follow the following heading underline syntaxe:

- for Heading 1, use =,
- for Heading 2, use -,
- for Heading 3, use .,
- for Heading 4, use *,
- for Heading 5, use #.

For new comers

New users are encouraged to contribute documentation using the following ways:

- Download the ZOO-Project svn, edit the documentation files located /docs directory and share the modifications through a new ticket set to 'Documentation' tracker
- Create a wiki page containg new or corrected documentation text, and create a new ticket to report its creation.

The ZOO developers responsible for the documentation will then review the contributions to add them into the official docs.

For registered developers

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[20] documentation generator is used to convert the reStructuredText files to html, and the live website is then updated on an hourly basis.

7.3.2 Installing and using Sphinx

On Linux

- Make sure you have the Python dev and setuptools packages installed. For example on Ubuntu:

```
sudo apt-get install python-dev
sudo apt-get install python-setuptools
```

- Install sphinx using easy_install:

```
sudo easy_install Sphinx==1.3.1
```

> **Note:** Make sure you install Sphinx 1.3.1 or more recent.

- Checkout the /docs directory from SVN, such as:

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

- 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.

Note: If there are more than one translation, the above commands will automatically build all translations.

On Mac OS X ™

- Install sphinx using easy_install:

```
sudo easy_install-2.7 Sphinx==1.3.1
```

> **Note:** Make sure you install Sphinx 1.3.1 or more recent.

- Install MacTex[21] if you want to build pdfs
- Checkout the /docs directory from SVN, such as:

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

- To process the docs, from the ZOO /docs directory, run:

[20] http://sphinx.pocoo.org/
[21] http://www.tug.org/mactex/2009/

```
make html
```

or

```
make latex
```

The HTML output will be written to the build/html sub-directory.

On Windows ™

- Install Python 2.X[22]
- Download setuptools[23]
- Make sure that the `C:/Python2X/Scripts` directory is your path
- Execute the following at commandline:

```
easy_install Sphinx==1.3.1
```

...you should see message: "Finished processing dependencies for Sphinx"

Note: Make sure you install Sphinx 1.3.1 or more recent. See note above.

- Install MiKTeX[24] if you want to build pdfs
- Checkout the /docs directory from SVN, such as:

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

- Inside the /docs directory, execute:

```
make html
```

or

```
make latex
```

The HTML output will be written to the _build/html sub-directory.

reStructuredText Reference Guides

The following resources are considered as useful for editing and creating new ZOO-Project documentation files.

- Docutils Quick reStructuredText[25]
- Docutils reStructuredText Directives[26]
- Sphinx's reStructuredText Primer[27]
- search Sphinx's mailing list[28]

[22] http://www.python.org/
[23] http://pypi.python.org/pypi/setuptools#windows
[24] http://miktex.org
[25] http://docutils.sourceforge.net/docs/user/rst/quickref.html
[26] http://docutils.sourceforge.net/docs/ref/rst/directives.html
[27] http://sphinx.pocoo.org/rest.html
[28] http://groups.google.com/group/sphinx-dev

7.4 Committer guidelines

This section gathers information to the registered ZOO-Project developers.

7.4.1 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 Commitee*. 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 commiters are responsible for having read, and understood this document.

7.4.2 Committer Tracking

A list of all project committers will be kept in the main zoo-project directory (called COMMITTERS[29]) listing for each SVN committer:

- Userid: the id that will appear in the SVN logs for this person.
- Full name: the users 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.

7.4.3 SVN Administrator

One member of the Project Steering Committee will be designed 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.

7.4.4 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 mentionned ticket.

[29] http://zoo-project.org/trac/browser/trunk/zoo-project/COMMITTERS

- After commiting 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.

7.4.5 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 commiter themselves understanding about the license is hopefully clear. For other contributors, the commiter should verify the understanding unless the commiter 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.

7.5 Release Procedure

The ZOO-Project release procedure is commonly defined by the following rules:

- Any of the *ZOO-Project Commiters* can ask for a release by asking the *ZOO-Project Project Steering Commitee* and pointing a release manager. This last will then vote for accepting both the manager and the release procedure to happen.
- If not already created, create a wiki page (like this one[30] using this scheme: Release/M.m.r/Notes), summarizing changes from the previous release (extracted from the revision log[31]).
- That file should include new features, changed features, and deprecated features if any. Changes to the official documentation should be specifically noted along with other items that will cause breaking changes during upgrades.
- Read the documentation and remove outdated parts.
- Create release candidate as .zip and .tar.bz2 then add them on this page[32] (by editing this wiki page[33])
- Cut a release candidate once you think that everything is in order. Announce the release candidate for review for at least 1 week. In this period of time, it is also appropriate for you to deploy in production since you are asserting that it is stable and (significant) bug free. Publish a specific revision with this.
- If significant bugs are reported, fix and cut a new release candidate. If no major bugs, then announce that the release candidate has officially been promoted to the official release (if you want, you can do this with a motion and support of the PSC).
- Ensure that release exactly matches something in SVN. Tag and branch appropriately.
- Update documentation as needed.
- Announce on various email list and other locations (news_item@osgeo.org[34], SlashGeo, etc)

[30] http://zoo-project.org/trac/wiki/Release/1.3.0/Notes
[31] http://zoo-project.org/trac/browser/trunk/zoo-project/HISTORY.txt
[32] http://zoo-project.org/new/Code/Download
[33] http://zoo-project.org/trac/wiki/ZooWebSite/2015/Code/Download
[34] news_item@osgeo.org

7.5.1 Creating an Official Release

Release versions lead to an update in documentation and standard tarballs. This is to help future administrators repeatably create releases.

- Double check that the pages from the ZOO-Project.org web site[35] match the current version.
- Double check that the latest build file matches the current revisions number.
- If this is a new major release create a branch and a tag.

```
cd zoo-project-svn/
svn cp trunk branches/branch-1.6
svn cp trunk tags/rel-1.6.0
```

- If this is a major or minor relase, create a tag.

```
svn cp branches/branch-1.6 tags/rel-1.6.1
```

- Commit the tags or branches with the version numbers.

```
svn commit -m 'Created branch/tags for the X.Y.Z release'
```

- Create version archives

```
export VERSION=2.6.0
cd zoo-propject-svn
cp -r trunk zoo-project-$VERSION
cd zoo-project-$VERSION
rm -rf $(find ./ -name ".svn")
cd zoo-project/zoo-kernel
autoconf
# In case you did not build ZOO-Kernel
cd ../../..
# In case you built ZOO-Kernel, then remove the generated file from the archive
make clean
rm -f  {Makefile,ZOOMakefile.opts}
cd ../../..
# In case you built one or more ZOO-Services, then remove the generated file from the archive
rm $(find ./zoo-project-$VERSION/zoo-project/zoo-services -name "*zo")
# Remove documentation from the archive
rm -rf ./zoo-project-$VERSION/{docs,workshop}
tar -cvjf ./zoo-project-$VERSION.tar.bz2 ./zoo-project-$VERSION
zip -r ./zoo-project-$VERSION.zip ./zoo-project-$VERSION
scp -P 1046 ./zoo-project-$VERSION.{zip,tar.bz2} zoo-project.org:/var/www/localhost/htdocs/dl/
```

- Update the Downloads page[36] to add the latest release (by editing this wiki page[37]).

7.6 Contribute translation

7.6.1 Introduction

This chapter aims to explain how to help translating the internal ZOO-Kernel messages into another language than English. Some languages are already enabled for translating and some are not. Please see *Request a new language* if needed.

[35] http://zoo-project.org/
[36] http://zoo-project.org/new/Code/Download
[37] http://zoo-project.org/trac/wiki/ZooWebSite/2015/Code/Download

This chapter describes how to add a new language, how to join a language group, and how to translate the internal ZOO-Kernel messages.

7.6.2 What is Transifex for?

Transifex[38] is a croudsourcing translation web application. It allows projects to let people easily translate documentation, websites, and applications.

It offers a graphic interface for those of you who don't like command line but also some features for those who love them.

Transifex helps the translator to improve their translation with suggestions, glossary and proofreading process.

7.6.3 Starting with Transifex

Subscribe to Transifex

You need to subscribe to Transifex before translating. Go to Transifex subscribing page[39] and create a new account.

Access the project

The project can be found here: ZOO-Project Dashboard[40]

Each project has a news page to send some information to you, so please, take some time to read them. You can also send some messages to others to discuss on a topic.

Ask to join the Translator group

Click on the link to access the dashboard of the ZOO-Project project in Transifex (link above), find the button *Join team*. You should now wait for the manager accept you in the group (given them time to read your request and act in consequence).

7.6.4 Request a new language

Go to the ZOO-Project dashboard on Transifex (see link above) and click on 'Request language'.

Creating a new language by the manager should not be too long, actually as long as to accept you in the team.

7.6.5 Translate messages

Once you are a member of the Translator team, go on the ZOO-Project Dashboard you can translate and click on the language you want to translate, then press the button 'Translate' and you can start translating strings by using the online editor. Please refer to this documentation[41] if you need details on how to use this editor.

[38] https://www.transifex.com/signup/contributor/
[39] https://www.transifex.com/signup/contributor/
[40] https://www.transifex.com/organization/zoo-project/dashboard
[41] http://docs.transifex.com/tutorials/txeditor/

7.7 List of contributors

7.7.1 ZOO-Project founders

The ZOO-Project concept, architecture and source code implementation was initiated in 2008 by the following individuals:

- Gérald FENOY (aka djay)
- Nicolas BOZON (aka nbozon)
- Venkatesh RAGHAVAN (aka venka)

7.7.2 ZOO-Project Project Steering Commitee

The ZOO Project Steering Comitee is responsible to manage the project which is maintained, improved, and supported by a small but growing developer community. The PSC is composed of the following people (by alphabetical order):

- Nicolas BOZON (GeoLabs SARL[42]), FR
- Maria Antonia BROVELLI (Politecnico di Milano[43]), IT
- Massimiliano CANNATA (SUPSI[44]), CH
- Gérald FENOY (GeoLabs[45]), FR **(Chair)**
- Hirofumi HAYASHI (AppTech[46]), JP
- Daniel KASTL (Georepublic[47]), DE
- Jeff McKENNA (Gateway Geomatics[48]), CA
- Markus NETELER (Fondazione Edmund Mach[49]), IT
- Venkatesh RAGHAVAN (Osaka City University[50]), JP
- Angelos TZOTSOS (National Technical University of Athens[51]), GR

7.7.3 ZOO-Project Commiters

The following individuals will be considered authorized ZOO-Project committers as long as they each review the commiter guidelines, and agree to adhere to them. The ZOO-Project commiters are listed here by alphabetical order.

- Nicolas BOZON (aka nbozon)
- Trevor CLARKE (aka tclarke)
- Luca DELUCCHI (aka lucadelu)

[42] http://geolabs.fr
[43] http://www.polimi.it
[44] http://www.ist.supsi.ch/
[45] http://www.geolabs.fr/
[46] http://www.apptec.co.jp/
[47] http://georepublic.de/en/
[48] http://www.gatewaygeomatics.com/
[49] http://gis.fem-environment.eu/
[50] http://www.osaka-cu.ac.jp/index-e.html
[51] http://users.ntua.gr/tzotsos/

- René-Luc D'HONT (aka reluc)
- Gérald FENOY (aka djay) **Admin**
- Knut LANDMARK (aka knut)
- Jeff MCKENNA (aka jmckenna)
- Markus NETELER (aka neteler)
- Marco NEGRETTI (aka nmarco)
- David SAGGIORATO (aka david)
- Angelos TZOTSOS (aka kalxas)

7.7.4 Other contributors

The following individuals have also contributed to the ZOO-Project source code or documentation.

- Thomas GRATIER
- Guillaume SUEUR
- Daisuke YOSHIDA