Implementation of OGC WPS standard: PyWPS

Jachym Cepicky

September 12, 2008

Copyright ©2006-2009 PyWPS Development Team Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

In this file, you can found the description of installation and configuration of PyWPS script. At the and, you can learn, how to add your own process. This document describes most recent version of PyWPS (*2.0.0*), available in subversion respository.

PyWPS project has been started on April 2006 with support of DBU – Deutsche Bundesstiftung Umwelt[1] and with help of GDF-Hannover[2] and Help Service Remote Sensing[3] companies. Initial author is Jachym Cepicky[4].

Contents

[1] http://dbu.de
[2] http://gdf-hannover.de
[3] http://www.bnhelp.cz
[4] http://les-ejk.cz

1 Introduction

PyWPS (Python Web Processing Service) is implementation of Web Processing Service 1.0.x standard from Open Geospatial Consortium[5].

It has been started on Mai 2006 as project supported by DBU. It offers environment for programming own process (geofunctions or models) which can be accessed from the public. The main advantage of PyWPS is, that it has been written with native support for GRASS GIS[6]. Access GRASS modules via web interace should be as easy as possible. However, not only GRASS GIS is supported. Usage of other programs, like R package or GDAL or PROJ tools is possible as well.

PyWPS is written in Python programming language, your processes must use this language too.

PyWPS Homepage can be found at http://pywps.wald.intevation.org. PyWPS Wiki is hosted on http://pywps.ominiverdi.org/wiki.

1.1 How it works

PyWPS is an translator application between client (Web Browser, Desktop GIS, command line tool, ...) and working tool installed on the server. PyWPS does no process the data by it self. As working tool, GRASS GIS, GDAL, PROJ, R and other programs can be used.

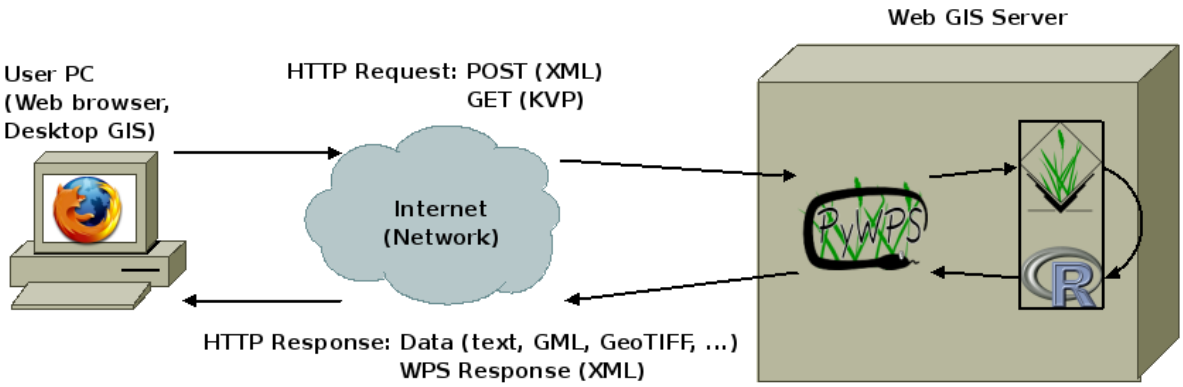

Figure 1: How does PyWPS work: GRASS GIS is in this case working tool

2 Quick install

1. Install PyWPS, see page 4 for details
2. NOTE: Rename original files (process examples, configuration files) with `.py-dist` suffix to `.py`, when you see them.
3. Edit configuration files in `pywps/etc/` directory. See page 5 for details.

[5] http://www.opengeospatial.org/standards/requests/28
[6] http://grass.itc.it

4. Create or edit `__init__.py` file in `pywps/processes` directory. Add available process names to `__all__` array.
5. Add your processes to `pywps/processes` directory. See page 8 for details.
6. Run PyWPS with `./wps.py` command, see page 7 for details.

3 Know issues

Known bugs and limitations to

- Translations do not work for GetCapabilities. They only work for DescribeProcess request types.
- If inputs are of type `LiteralValue` and it's type is string, it could be security problem. Take care on your inputs and do not use it directly in scripts to avoid your server to be hacked.

Please report all problems or unexpected handeling either via pywps mailing list[7] or using PyWPS bugtracker[8].

4 Installation

Required packages:

- python
- python-xml
- python-htmltmpl

Recommended packages:

- Web Server (e.g. Apache) – `http://httpd.apache.org` - You will need an web server, to be able to execute processes from remote computers.
- GIS GRASS – `http://grass.itc.it` - Geographical Resources Analysis Support System (GRASS) is Open Source GIS, which provides more then 350 modules for raster and vector (2D, 3D) data analysis. PyWPS is written with native support for GRASS and it's functions.
- PROJ.4 – `http://proj.maptools.org` - Cartographic Projections library used in various Open Source projects, such as GRASS, UMN MapServer, QGIS and others. It can be used e.g. for data transformation.
- GDAL/OGR – `http://gdal.org` - translator library for raster geospatial data formats, is used in various projects for importing, exporting and transformation between various raster and vector data formats.
- R – `http://www.r-project.org` - is a language and environment for statistical computing and graphics.

[7] PyWPS - development list
[8] PyWPS Bug tracker

4.1 Installation the quick 'n' dirty way

For installing pywps to your server simply unzip the archive to the directory, where cgi programs are allowed to run. You can also use current repository version.

```
$ cd /usr/lib/cgi-bin/
$ tar xvzf /tmp/pywps-VERSION.tar.gz
$ pywps/wps.py
```

4.2 Installation the 'clean' way

Unzip the package

```
$ tar -xzf pywps-VERSION.tar.gz
```

and run

```
$ python setup.py install
```

adjust the configuration file

```
$ vim /etc/pywps.cfg
```

permint write access to templates directory

```
# chmod -R 777 /usr/lib/python2.5/site-packages/pywps/Templates
```

Several binary packages for Linux distributios (`RPM`,`DEB`) are also avaliable on PyWPS homepage[9].

5 Configuration

Before you start to tune your PyWPS installation, you should get your copy of OpenGIS(R) Web Processing Service document (OGC 05-007r7) version 1.0.0[10].

NOTE: Note, that the configuration option are CASE SENSITIVE

Pywps configuration takes place in `pywps.cfg` file located in `/etc/pywps.cfg` or `pywps/etc/pywps.cfg`.

Default configuration file is located in `pywps/default.cfg`, you can always make a copy of this file and start the configuration from scratch.

Several sections are in the file.

- Section [**wps**] contains general WPS settings, which are:
  - encoding – Language encoding (utf-8, iso-8859-2, windows-1250, ...)
  - title – Server title
  - version – WPS version (1.0.0)
  - abstract – Server anstract
  - fees – Possible fees

[9] http://pywps.wald.intevation.org
[10] http://www.opengeospatial.org/standards/wps

  - constraints – Possible constraints
  - serveraddress – WPS script address: `http://foo/bar/wps.py`
  - keywords – Comma-separated list of kyewords
  - lang – Default langue (eng)

- Section [**provider**] contains informations about you
  - providerName – Name of your company
  - individualName – Your name
  - positionName
  - role
  - deliveryPoint – Street
  - city
  - postalCode
  - country
  - electronicMailAddress – foo@bar
  - providerSite – `http://foo.bar`
  - phoneVoice
  - phoneFacsimile
  - administrativeArea

- Section [**server**] contains server settings
  - maxoperations – Maximal number of parallel running processes. If set to 0, then there is no limit.
  - maxinputparamlength – Maximal length of string input parameter.
  - maxfilesize – Maximal input file size (raster or vector). The size can be determined as follows: 1GB, 5MB, 3kB, 1000b.
  - tempPath – Direcotory for temporary files (mostly temporary GRASS locations).
  - outputUrl – Url where the outputs are stored.
  - outputPath – Path. where output files are stored.
  - debug – true/false
  - processPath – path to your processes. Default is `pywps/processes`.
    NOTE: You can set also `PYWPS_PROCESSES` environment variable with same result.

- Section [**grass**] – GRASS GIS settings
  - path – $PATH variable, e.g. `/usr/lib/grass/bin`
  - addonPath – $GRASS_ADDONS variable
  - version – GRASS version
  - gui – Should be "text"
  - gisbase – Path to GRASS GIS_BASE directory (`/usr/lib/grass`)

- ldLibraryPath – Path of GRASS Libs (`/usr/lib/grass/lib`)
- gisdbase – Full path to location directory (`/home/foo/grassdata`)

File example follows:

```
[wps]
encoding=utf-8
title=PyWPS Server
version=1.0.0
abstract=See http://pywps.wald.intevation.org and http://www.opengeospatial.org/standards/
fees=None
constraints=none
serveraddress=http://localhost/cgi-bin/wps
keywords=GRASS,GIS,WPS
lang=eng

[provider]
providerName=Your Company Name
individualName=Your Name
positionName=Your Position
role=Your role
deliveryPoint=Street
city=City
postalCode=000 00
country=eu
electronicMailAddress=login@server.org
providerSite=http://foo.bar
phoneVoice=False
phoneFacsimile=False
administrativeArea=False

[server]
maxoperations=3
maxinputparamlength=1024
maxfilesize=3mb
tempPath=/tmp
outputUrl=http://localhost/wps/wpsoutputs
outputPath=/var/www/wps/wpsoutputs
debug=true

[grass]
path=/usr/lib/grass/bin/:/usr/lib/grass/scripts/
addonPath=
version=6.2.1
gui=text
gisbase=/usr/lib/grass/
ldLibraryPath=/usr/lib/grass/lib
```

```
gisbase=/home/foo/datagrass
```

subsectionTesting after installation For test, just run `wps.py` in your command line:

```
$ ./wps.py "service=wps&request=getcapabilities"

INIT DONE
LOADING PRECOMPILED
TEMPLATE: UPTODATE
PRECOMPILED: UPTODATE
Content-type: text/xml

<?xml version="1.0" encoding="utf-8"?>
<wps:Capabilities service="WPS" version="1.0.0" xml:lang="eng,ger"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:wps="http://www.opengis.net/wps/1.0.0"
    xmlns:ows="http://www.opengis.net/ows/1.1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
    xsi:schemaLocation="http://www.opengis.net/wps/1.0.0
    http://schemas.opengis.net/wps/1.0.0/wpsGetCapabilities_response.xsd"
    updateSequence="1">
<ows:ServiceIdentification>
<ows:Title>PyWPS Development Server</ows:Title>
    ...
</wps:Capabilities>
```

If you got something like this, (Capabilities response), everything looks fine.
If you got some other message, like e.g.:

```
Traceback (most recent call last):
  File "/usr/bin/wps.py", line 221, in <module>
    wps = WPS()
  File "/usr/bin/wps.py", line 140, in __init__
    self.performRequest()
  File "/usr/bin/wps.py", line 188, in performRequest
    from pywps.WPS.GetCapabilities import GetCapabilities
  File "/usr/lib/python2.5/site-packages/pywps/WPS/GetCapabilities.py", line 26, in <modul
    from Response import Response
  File "/usr/lib/python2.5/site-packages/pywps/WPS/Response.py", line 28, in <module>
    from htmltmpl import TemplateManager, TemplateProcessor
ImportError: No module named htmltmpl
```

Than something is wrong with your Python installation or with the program. This message means, that the python-htmltmpl package is not installed in your system.

6 Write your own processes

All processes are stored in the `pywps/processes` directory. You can create custom directory anywhere in your system and set `$PYTHON_PROCESS` environment variabl (how to do this for

the web server, refer to your Server documentation). Following example will describe buffering process. Several example processes are distributed along with PyWPS source code.

Create file `exampleBufferProcess.py` in PYWPS_PROCESSES directory.

Each process is stand-alone python script with one class `Process`, which has two methods: `__init__`, `execute`. It is possible also to add as many your functions/methods, as you wish.

6.1 Process initialization and configuration

```
 1 from pywps.Process.Process import WPSProcess
 2 class Process(WPSProcess):
 3     """Main process class"""
 4     def __init__(self):
 5         """Process initialization"""
 7         # init process
 8         WPSProcess.__init__(self,
 9             identifier = "exampleBufferProcess",
10             title="Buffer",
11             version = "0.2",
12             storeSupported = "true",
13             statusSupported = "true",
14             abstract="Create a buffer around an input vector file",
15             grassLocation = True)
```

We defined new process called `exampleBufferProcess`. The process is allowed to store it's output data on the server (`storeSupported`) and it is also possible to run it in asynchronous mode (`statusSupported`). The process will run within GRASS GIS environment (`grassLocation = True`).

Metadata defition is stored in array `self.Metadata` in `__init__` method. You can add new Medatada using `self.AddMetadata()` method:

```
self.AddMetadata(identifier="point",type="point",
                 textContent="Click in the map")
```

6.1.1 Data Inputs

Three types of data inputs are defined:

- Literal Input – Basic literal input – single number or text value
- ComplexValue Input – Mostly vector file embded in input XML request or reference (URL) to such file.
- BoundingBox Input – Coordinates for lower-left and upper-right corner.

ComplexInput Complex input can be raster or vector file, to be processed.

```
18         self.dataIn = self.addComplexInput(identifier="data",
19                                 title = "Input data")
20
```

LiteralInput With literal input, you can obtain any type of character string.

```
21          self.widthIn = self.addLiteralInput(identifier = "width",
22                                  title = "Width")
23
```

For further documentation, refere example processes distributed with the source code as well as `pydoc pywps/Wps/Process.py`. This help is also available in `process.html`[11] file distributed along with PyWPS source code.

6.1.2 Data Outputs

Data outputs can be defined in similar way.

- Literal Output
- ComplexValue Outout
- BoundingBox Output

ComplexValue Output The complex value can be raster or vector file (or any other binary or text file).

```
24          self.bufferOut = self.addComplexOutput(identifier="buffer",
25                                   title="Output buffer file")
26
```

Literal Output If you want to output any text string.

```
27           self.textOut = self.addLiteralOutput(identifier="text",
28                                  title="just some text")
29
```

6.2 Process Programming

The process must be defined in the `execute(self)` method. Basicly, you want to get input values and set output values. For this purpose, you can use `getValue(input_identifier)` and `setValue(output_identifier,value)` methods of the input/output objects (see lower).

If you need to execute some shell command, you should use `self.cmd(command,["string for standard input"])` instead of e.g. `os.system()` or `os.popen()` functions.

Calculation progress can be set using `self.status(string message, number percent)` method.

Example follows:

```
33      def execute(self):
34          """Execute process.
35
36          Each command will be executed and output values will be set
```

[11] Documentation to Process.py module

```
37          """
38
39          # run some command from the command line
40          self.cmd("g.region -d")
41
42          # set status value
43          self.status.set("Importing data",20)
44          self.cmd("v.in.ogr dsn=%s output=data" %\
45                  (self.getInputValue('data')))
46
47          self.status.set("Buffering",50)
48          self.cmd("v.buffer input=data output=data_buff buffer=%s scale=1.0 tolerance=0.
49                  (self.getInputValue('width')))
50
51          self.status.set("Exporting data",90)
52
53          self.cmd("v.out.ogr type=area format=GML input=data_buff dsn=out.xml olayer=pat
54
55          self.bufferOut.setValue("out.xml")
56          self.textOut.setValue("ahoj, svete")
57          return
```

6.2.1 Error handling

At the end of the `execute` function, `None` value should be returned. Any other value means, that the calculation will be stopped and error report will be returned back to the client, example:

```
def execute(self):
    ...
    return "Ups, something failed!"
```

6.3 Using GRASS GIS

Configuration is done using standard pywps configuration file (see page5).

If you want to use GRASS GIS commands in your process, and there is no GRASS Location to be used, you have to set `grassLocation=True` in process definition:

```
WPSProcess.__init__(self,
    identifier = "exampleBufferProcess",
    ....
    grassLocation = True)
```

In this case, temporary GRASS Location will be created and after the process is done, it will be deleted again. By default, no GRASS Location is created.

You can also work in existing GRASS Location, then just set only the location name.

```
WPSProcess.__init__(self,
```

```
        identifier = "exampleBufferProcess",
        ....
        grassLocation = "spearfish60")
```

7 Testing your new process

To test your PyWPS installation, you run it either as Webserver cgi-application or in the command line directly. It is always good to start with the command line test, so do not have to check `error.log` of the web server.

- GetCapabilities request (webserver)

```
./wps.py "service=wps&request=getcapabilities"

wget -nv -q -O - "http://localhost/cgi-bin/wps.py?\
    service=Wps&request=getcapabilities"
```

- DescribeProcess request:

```
./wps.py "version=1.0.0&service=Wps&request=DescribeProcess&\
    Identifier=bufferExampleProcess"

wget -nv -q -O - "http://localhost/cgi-bin/wps.py?\
    version=0.4.0&service=Wps&request=DescribeProcess&\
    Identifier=exampleBufferProcess"
```

- Execute request: For data inputs encoding, using HTTP Get method, see OGC 05-007r7[12], page 38 „Execute HTTP GET request KVP encoding“

```
./wps.py "version=1.0.0&service=Wps&\
    request=Execute&Identifier=exampleBufferProcess&\
    datainputs=data=http://foo/bar/roads.gml;width=0.5"
```

Some examples of XML request econding are available in doc/examples directory.

Before testing WPS via HTTP POST, you have to set `REQUEST_METHOD` environment variable, then you can redirect input XML into `wps.py` script via standard input:

```
$ export REQUEST_METHOD=POST
$ cat doc/wps_execute_request-responsedocument.xml|./wps.py
```

[12] http://opengeospatial.org/standards/wps/