PyWPS Documentation

Release 3.2.5

Jachym Cepicky

June 06, 2016

Contents

Documentation for PyWPS. PyWPS is implementation of OGC Web Processing Service (WPS), version 1.0.0.

PyWPS is written in Python. It is possible to run it as CGI, Mod_python environemnt, as well as Java servlet via jython.

PyWPS should be between server-side application and WPS clients. The application can be written with Python, Java or executabed from the command line. PyWPS was written with direct support for GRASS GIS.

Contents:

CHAPTER 1

Installation of PyWPS

1.1 Prerequirements

- python – currently 2.5, 2.6 are supported, 2.4 is deprecated
- python-xml

1.2 Recommended packages

Web Server (e.g. Apache) - http://httpd.apache.org - You will need a web server to be able to execute processes from remote clients over the Internet. PyWPS was tested with Apache 1.1 and 2.x versions.

GIS GRASS http://grass.osgeo.it - Geographical Resources Analysis Support System (GRASS) is an 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. GRASS also has Python bindings, so you can run the modules directly.

PROJ.4 http://proj.maptools.org - Cartographic Projections library used in various Open Source projects, such as GRASS, MapServer, QGIS and others. It can be used e.g. for coordinate transformation. Proj4 is required if you want to integrate MapServer as well, using the *python-pyproj* package.

GDAL/OGR http://gdal.org - translator library for raster geospatial data formats. GDAL/OGR is used in various projects for importing, exporting and transformation between various raster and vector data formats. GDAL and OGR are also required if you want to integrate MapServer, using the *python-gdal* package.

MapServer http://mapserver.org - If you want to access ComplexValue outputs using OGC OWS (WMS, WFS, WCS) services, MapServer is required. PyWPS will generate a MapServer mapfile for you automatically. The *python-mapscript* package is required.

R http://www.r-project.org - is a language and environment for statistical computing and graphics.

1.3 Installation the quick 'n' dirty way

For installing PyWPS to your server quickly, simply unpack the archive to some directory. You can also use current repository version.:

1 - cd to target directory:

```
$ cd /usr/local/
```

2 - unpack pywps:

```
$ tar xvzf /tmp/pywps-VERSION.tar.gz
```

1.3.1 Post-installation steps

You have to change the write access of `pywps/Templates/*WPS_VERSION*/` directory, so the web server can compile the templates:

```
chmod 777 /usr/local/pywps-VERSION/pywps/Templates/1_0_0
```

1.4 Installation the 'clean' way

Unpack the package

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

and run

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

1.5 Installation using prebuild distribution packages

PyWPS provides packages for Debian and RPM based Linux Distributions.

Note: The packages are not maintained properly and until we don't find packagers, we recommend to use any other approach, described earlier in this section.

1.6 Testing the installation

If PyWPS has been successfully installed, you can test this with running it without any input or further configuration.

First you need to find the *cgiwps.py* script, which is in the root of pywps installation directory (if you used the Quick and dirty way), or it should be located in `/usr/bin` directory, if you used the clean way of installation.

Run *cgiwps.py* on the command line:

```
$ ./cgiwps.py
```

And you should get result like this (which is a mixture of standard output and standard error):

```
PyWPS NoApplicableCode: Locator: None; Value: No query string found.
Content-type: text/xml

<?xml version="1.0" encoding="utf-8"?>
<ExceptionReport version="1.0.0" xmlns="http://www.opengis.net/ows" xmlns:xsi="http://www.w3.org/200
        <Exception exceptionCode="NoApplicableCode">
                <ExceptionText>
                        No query string found.
                </ExceptionText>
```

```
        </Exception>
</ExceptionReport>
```

In this case, you have installed PyWPS correctly and you are ready to proceed to configuration.

CHAPTER 2

Configuration

Note: Before you start to tune your PyWPS installation, you can download OpenGIS(R) Web Processing Service document (OGC 05-007r7) version 1.0.0 http://www.opengeospatial.org/standards/wps or later, for reference.

2.1 Setting up the PyWPS instance

PyWPS can be installed once on your server, but it be configured for many WPS servers (instances). Each WPS server needs a set of processes (stored in one directory) and a configuration file. Processes are stored together as python programs in one directory as follows:

1 - create `processes` directory – directory, where you store all processes for particular PyWPS instance:

```
$ mkdir -p /usr/local/wps/processes
```

2 - copy template of the configuration file to some location, and configure your PyWPS installation (see below):

```
$ cp pywps-VERSION/pywps/default.cfg /usr/local/wps/pywps.cfg
$ $EDITOR /usr/locap/wps/pywps.cfg
```

3 - create any process(es) in the `processes` directory. You can start with the example processes, stored in *pywps-VERSION/examples/processes* directory. See how-to-write-custom-process for how to write custom processes.:

```
$ cp pywps-VERSION/examples/ultimatequestionprocess.py /usr/local/wps/processes/
```

6 - Each process in the `processes` directory must be registered in the *__init__.py* file. The file has to contain at least:

```
__all__ = ["ultimatequestionprocess"]
```

Where `__all__` represents list of processes (file names) within the `processes` directory.

2.1.1 Accepted environment variables

The following environment variables are accepted by a PyWPS instance:

PYWPS_CFG Configuration file location

PYWPS_PROCESSES Directory, where the processes are stored

PYWPS_TEMPLATES Templates directory (structure should be similar to file:*pywps/Templates*)

2.1.2 Setting up the Web Server

PyWPS can run as CGI application or in mod_python mode. CGI is easier to setup, where mod_python is less demanding on server resources, since after the first run, PyWPS and Python itself are loaded into memory.

PyWPS as CGI

CGI configuration is a simple appraoch, without any additional server configuration.

To configure PyWPS via CGI, copy the PyWPS CGI wrapper script to your `cgi-bin` directory and edit the variables:

```
$ cp pywps/resources/pywps.cgi /usr/lib/cgi-bin
$ $EDITOR /usr/lib/cgi-bin/pywps.cgi
```

Note: Windows users must create a either a .bat file or Python wrapper. This example is written as UNIX shell script.

Note: This script is to be used only via HTTP (with e.g. Apache). If you want to run PyWPS from the command line, use *wps.py* directly.

Below is a sample wrapper:

```
#!/bin/sh

# Author: Jachym Cepicky
# Purpose: CGI script for wrapping PyWPS script
# Licence: GNU/GPL
# Usage: Put this script to your web server cgi-bin directory, e.g.
# /usr/lib/cgi-bin/ and make it executable (chmod 755 pywps.cgi)

# NOTE: tested on linux/apache

export PYWPS_CFG=/usr/local/wps/pywps.cfg
export PYWPS_PROCESSES=/usr/local/wps/processes/

/usr/local/pywps-VERSION/cgiwps.py
```

You can also configure HTTP environment variables using standard Apache server configuration file (see mod_python) for example.

PyWPS in mod_python

Overall, PyWPS has better performance via mod_python. All necessary libraries are pre-loaded into memory and response times should be faster in some cases.

1 - Install necessary packages, on debian, it is *libapache2-mod-python* 2 - Configure Apache HTTP server (see Mod Python documentation).

1 - Create python directory (preferably outside `htdocs` directory):

```
$ mkdir /var/www/wps/
```

2 - Add this to your HTTP configuration file:

```
<Directory /var/www/wps>
    SetEnv PYWPS_PROCESSES /usr/local/wps/processes
    SetEnv PYWPS_CFG /usr/local/wps/pywps.cfg
    SetHandler python-program
    PythonHandler pywps
    PythonDebug On
    PythonPath "sys.path+['/usr/local/pywps-VERSION/']"
    PythonAutoReload On
</Directory>
```

or you can copy `resources/.htaccess` to */var/www/wps* – depending on what level of access you are provided by your system administrator.

3 - Copy `resources/pywps.py` to `/var/www/wps`

2.2 PyWPS configuration files

Configuration file for PyWPS can be located in several places. There are global and local PyWPS configuration files. Local configurations override global configurations.

2.2.1 Global PyWPS configuration files

1. File `/etc/pywps.cfg` (on Linux/Unix)
2. File `/usr/local/pywps-VERSION/etc/pywps.cfg`, which means the file `pywps.cfg` in directory `etc`, located in PyWPS install location.

And one special file:

File `/usr/local/pywps-VERSION/pywps/default.cfg`, which means the file `default.cfg` in directory `pywps`, located in PyWPS install location. This is the default configuration file.

Note: Never rewrite or remove this file. Use it only as template for your custom configuration files.

2.2.2 Local PyWPS configuration file

The local configuration file is used for the particular PyWPS instance only. It is the file, stored in `PYWPS_CFG` environment variable. This can be set either via web server configuration or with any wrapper script (see `resources/pywps.cgi` for example).

Make a copy of `pywps/default.cfg` to `/usr/local/wps/pywps.cfg` and customize the file as per below.

2.3 Configuration of PyWPS instance

Several sections are in the configuration file. The sections contain *key value* pairs of configuration options (see the example at the end of this section). If you do not set these options, they will be taken from the default configuration file.

2.3.1 WPS

The [wps] section contains general WPS instance settings, which are:

encoding Language encoding (utf-8, iso-8859-2, windows-1250, dots)

title Server title

version WPS version (1.0.0)

abstract Server abstract

fees Possible fees

constraints Possible constraints

serveraddress WPS script address: http://foo/bar/pywps.py or http://foo/bar/cgi-bin/pywps.cgi

keywords Comma-separated list of keywords realted to this server instance

lang Comma-separated list of supported server languages. Default is 'eng'.

2.3.2 Provider

The [provider] section contains information about you, your organization and so on:

providerName Name of your company

individualName Your name

positionName At which position you are working

role What your role is

deliveryPoint Street

city City

postalCode Postal code or Zip code

country Country name

electronicMailAddress E-mail address

providerSite Web site of your organization

phoneVoice Telephone number

phoneFacsimile Fax number

administrativeArea State, province, territory or administrative area

hoursofservice Hours of service to contact the provider

contactinstructions Instructions on how to contact the provider

2.3.3 Server

The [server] section contains server settings, constraints, safety configuration and so on:

maxoperations Maximum number of parallel running processes. If set to 0, then there is no limit.

maxinputparamlength Maximum length of string input parameter (number of characters).

maxfilesize Maximum input file size (raster or vector). The size can be determined as follows: 1GB, 5MB, 3kB, 1000b.

tempPath Directory for temporary files (e.g. `/tmp/pywps`). PyWPS will create temporary directories in this directory, and after the calculation is performed, they *should* be deleted again.

outputPath Path where output files are stored on the server. This should point to the *outputUrl* parameter (described below). For example http://foo/bar/wpsputputs. If outputPath starts with ftp:// it's assumed that FTP support shall be used.

outputUrl Url where the outputs are stored for client access. On Debian, it would be for example `/var/www/wpsoutputs`

ftplogin FTP user login, if empty, anonymous login is used.

Note: FTP support is activated by ftp:// in outputPath

---

ftppasswd FTP user password

ftpport Default FTP port 21 is used if variable not defined.

debug true/false - makes the logs for verbose

Note: This option is not used so wildly, as it should maybe be.

Note: Deprecated since 3.2. Use logLevel instead

---

processesPath path to your processes. Default is pywps/processes.

Note: You can also set the `PYWPS_PROCESSES` environment variable with the same result, as described earlier on this page.

---

logFile (since 3.0.1) File where all PyWPS logs go to. If not set, default error.log from Web Server configuration is used. Sometimes, this can cause problem for the asynchronous calls.

logLevel (since 3.2) one of DEBUG, INFO, WARNING, ERROR and CRITICAL, default is INFO

2.3.4 GRASS

The [grass] section is specifically for GRASS GIS settings (optional):

path `PATH` environment variable, e.g. `/usr/lib/grass/bin:/usr/lib/grass/scripts`

addonPath `GRASS_ADDONS` environment 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 GRASS database directory, where *Locations* are stored (`/home/foo/grassdata`)

2.4 Configuration file example

```
[wps]
encoding=utf-8
title=PyWPS Server
version=1.0.0
abstract=See http://pywps.wald.intevation.org and http://www.opengeospatial.org/standards/wps
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
processesPath=
outputUrl=http://localhost/wps/wpsoutputs
outputPath=/var/www/wps/wpsoutputs
debug=true
logFile=/var/log/pywps.log

[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
gisdbase=/home/foo/datagrass
```

2.5 Notes for Windows users

Windows users do have to adjust their paths to what is standard on this platform. E.g. instead of using ":" as delemiter ";" is supposed to be used. Also usage of slash "/" and backslash "\" can be tricky.

Generally speaking, it's good to start by installing GRASS (if needed) and all the required geospatial packages using OSGeo4W tool.

Having GRASS and PyWPS is possible and was successfuly tested. You have to adjust especially PATH variable. Example of relevant configuration parts follows:

```
[server]
maxoperations=30
maxinputparamlength=1024
maxfilesize=10mb
tempPath=c:\\\\tmp
processesPath=
outputUrl=http://localhost/tmp/wpsoutputs
outputPath=c:\OSGeo4W\apache\htdocs\tmp\wpsoutputs\
debug=true # deprecated since 3.2, use logLevel instead
logFile=
logLevel=INFO

[grass]
path=c:\\\\osgeo4w/apps/grass/grass-7.0.0/lib;c:\\\\osgeo4w/apps/grass/grass-7.0.0/bin;c:\\\\c/Users/
addonPath=
version=7.0.0
gui=text
gisbase=c:\\\\OSGeo4W\\\\apps\\\\grass\\\\grass-7.0.0
ldLibraryPath=c:\OSGeo4W\apps\grass\grass-7.0.0\lib
gisdbase=c:\Users\jachym\src\vugtk\grassdata\
home=c:\Users\jachym
```

FOr the configuration of Apache web server, you can directly use *wps.py* binary from the root of PyWPS source code and use it. Example of relevant httpd.conf file follows (it can of course be used on Unix as well):

```
# wps.py was copied from pywps-source/wps.py
Alias /wps C:\OSGeo4W/bin/wps.py
<Location /wps
        SetHandler cgi-script
        Options ExecCGI
        SetEnv PYWPS_CFG C:\path/to/your/configuration/pywps.cfg
        SetEnv PYWPS_PROCESSES C:\path/to/your/processes
</Location>
```

CHAPTER 3

Testing PyWPS

Testing PyWPS can be done on the command line – it is the easier way, how to get both – standard error and standard output – at once. Testing in the web server environment can be done later.

Before we start to test, be aware that we assume the following:

1 - PyWPS is installed properly, see installation 2 - Configuration file is stored in `/usr/local/wps/pywps.cfg`,

see *Configuration*

3 - At least one process is stored in the `/usr/local/wps/processes` directory.

4 - There is a `/usr/local/wps/processes/__init__.py` file, with at least:

```
__all__ = ['yourProcess']
```

text in it. For testing purposes, we assume that *yourProcess* is *ultimatequestionprocess*. For further reading about how to setup custom processes, see custom-processes.

For testing, we are using HTTP GET KVP encoding of OGC WPS request parameters. If you require clarification of WPS request parameters, please consult the OGC WPS 1.0.0 standard.

Note: Be aware that this document describes PyWPS, which is a *server* implementation of OGC WPS. There is some graphical user interface to the server (WPS Clients), but for testing purposes, they are not suitable. That is the reason, why following section will use command line tools and direct XML outputs.

3.1 Testing PyWPS installation

Find the location of `cgiwps.py` and run it without any further configuration.:

```
$ ./cgiwps.py
```

And you should get result like this (which is a mixture of standard output and standard error):

```
Content-type: text/xml

PyWPS NoApplicableCode: Locator: None; Value: No query string found.
<?xml version="1.0" encoding="utf-8"?>
<ExceptionReport version="1.0.0" xmlns="http://www.opengis.net/ows" xmlns:xsi="http://www.w3.org/200
        <Exception exceptionCode="NoApplicableCode">
                <ExceptionText>
```

```
                        No query string found.
                </ExceptionText>
        </Exception>
</ExceptionReport>
```

3.2 Testing PyWPS configuration

Now we have to export two environment variables: location of the configuration file and location of processes directory:

```
$ export PYWPS_CFG=/usr/local/wps/pywps.cfg
$ export PYWPS_PROCESSES=/usr/local/wps/processes
```

Afterwards, you can run the PyWPS CGI script. We will use HTTP GET requests, because they are easy to follow and faster to construct.

3.2.1 GetCapabilities

On the command line:

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

You should obtain a Capabilities response:

```
Content-Type: text/xml

<?xml version="1.0" encoding="utf-8"?>
<wps:Capabilities service="WPS" version="1.0.0" xml:lang="eng" xmlns:xlink="http://www.w3.org/1999/xl
        <ows:ServiceIdentification>
        [...]
```

Note: Have a more detailed look at the *<wps:ProcessOfferings>...</wps:ProcessOfferings> part of the output XML. There should be at least 'Process*

3.2.2 DescribeProcess

On the command line:

```
$ ./cgiwps.py "service=wps&version=1.0.0&request=describeprocess&identifier=Process"
```

You should obtain a ProcessDescriptions response:

```
<?xml version="1.0" encoding="utf-8"?>
<wps:ProcessDescriptions xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:ows="http://www.opengis.n
    <ProcessDescription wps:processVersion="0.2" storeSupported="True" statusSupported="True">
        <ows:Identifier>ultimatequestionprocess</ows:Identifier>
        <ows:Title>The numerical answer to Life, Universe and Everything</ows:Title>
        [...]
```

3.2.3 Execute

On the command line:

```
$ ./cgiwps.py "service=wps&version=1.0.0&request=execute&identifier=ultimatequestionprocess"
```

You should obtain an ExecuteResponse response (this may take some time):

```
<?xml version="1.0" encoding="utf-8"?>
<wps:ExecuteResponse xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:ows="http://www.opengis.net/o
    <wps:Process wps:processVersion="2.0">
        <ows:Identifier>ultimatequestionprocess</ows:Identifier>
        <ows:Title>Answer to Life, the Universe and Everything</ows:Title>
        <ows:Abstract>Numerical solution that is the answer to Life, Universe and Everything. The pro
    </wps:Process>
    <wps:Status creationTime="Tue Jan 26 12:37:18 2010">
        <wps:ProcessSucceeded>PyWPS Process ultimatequestionprocess successfully calculated</wps:Proc
    </wps:Status>
    <wps:ProcessOutputs>
        <wps:Output>
            <ows:Identifier>answer</ows:Identifier>
            <ows:Title>The numerical answer to Life, Universe and Everything</ows:Title>
            <wps:Data>
                <wps:LiteralData dataType="integer">42</wps:LiteralData>
            </wps:Data>
        </wps:Output>
    </wps:ProcessOutputs>
</wps:ExecuteResponse>
```

3.3 Issues

Note: A list of known problems follows. If you have seen something different, please let us know via the mailing list.

Note: Every error you get, should have standard error and standard output part, but they are mixed together. We describe here the most important part, the general error description.

Could not store file in compiled form: [Errno 13] Permission denied: 'pywps/Templates/1_0_0/GetCapabilities.tmplc'
PyWPS tries to store precompiled templates to Templates directory and does not have rights for it (or the user, under which PyWPS is running, does not have the rights, e.g. www-data). Change permissions of the directory, so that other users can write in it as well.

Process executed. Failed to build final response for output [los]: [Errno 13] Permission denied: '/var/tmp/pywps/los-6165.tif' Process executed. Failed to build final response for output [los]: [Errno 2] No such file or directory: '/var/tmp/pywpsx/los-6217.tif'

PyWPS probably successfully calculated the process, but when it tried to store result file to output directory, it failed. Try to set read-write access to directory with output files or create the output directory.

[Errno 2] No such file or directory: '/tmp/' [Errno 13] Permission denied: '/tmp/'

PyWPS did not find some directory or file, configured in the configuration file, or the appropriate permissions are not set.

No process in ProcessOfferings listed The `PYWPS_PROCESSES` is not set properly or there is no:

```
    __all__ = ['ultimatequestionprocess']
```

in the `__init__.py` in the `PYWPS_PROCESSES` directory.

CHAPTER 4

PyWPS Process

4.1 Processes directory

A PyWPS process can be thought of as a Python module. All PyWPS processes are stored in one directory Python Package. The `__init__.py` file must contain the list of processes in *__all__* array.

4.1.1 Default location for processes

The default location of PyWPS processes is located in the `pywps/processes` directory, in the installation location of PyWPS.

4.1.2 Configuration via `PYWPS_PROCESSES` environment variable

Usually, you will overwrite this with the `PYWPS_PROCESSES` environment variable in the *Configuration*.

4.1.3 Configuration via configuration file

Alternatively you can set the *processesPath* variable in the configuration file.

4.2 Logging

PyWPS uses Python module `logging` for logging purposes. If there is something you need to log (activity, variable content for debug or anything else), just import the module and use accordingly:

```
import logging
LOGGER = logging.getLogger(__name__)
...

LOGGER.info("Just information message")
LOGGER.debug("Variable content: %s" % variable)
LOGGER.error("Error occured: %s" % e)
```

The logs are printed to standard error, or to a file set in the configuration file *Configuration*. Log level is set with *logLevel* option, also in the configuration file.

4.2.1 Process structure

In the file containing the process, there must be one class with the name `Process`, which is instance of `pywps.Process.WPSProcess` class. Each process must contain at least two methods: `pywps.Process.WPSProcess.__init__()` and `pywps.Process.WPSProcess.execute()`.

Process initialization: __init__ method

This method is the constructor for the actual process. It has to invoke the `pywps.Process.WPSProcess.__init__()` method of the superior *WPSProcess* class with process configuration options, described in `pywps.Process.WPSProcess` in more detail.

The process can then define several `pywps.Process.InAndOutputs.Input` and `pywps.Process.InAndOutputs.Output` instances. Several methods can be used for this, namely `pywps.Process.WPSProcess.addLiteralInput()`, `pywps.Process.WPSProcess.addComplexInput()`, `pywps.Process.WPSProcess.addBBoxInput()` for inputs and `pywps.Process.WPSProcess.addLiteralOutput()`, `pywps.Process.WPSProcess.addComplexOu` `pywps.Process.WPSProcess.addBBoxOutput()` for outputs.

Process execution: execute method

The `pywps.Process.WPSProcess.execute()` method, which is originally empty, is called by PyWPS for process execution. The actual calculation is to be done here. When the process returns any text, it is handled as an error message. When a process is successfully calculated, this method returns None.

Example of PyWPS process

After adding *"returner"* string to *__all__* array, in the `__init__.py` file in the PyWPS Processes directory, we can try GetCapabilities, DescribeProcess and Execute requests.

4.2.2 Process Inputs and Outputs

Process inputs and outputs are of three types:

ComplexValue Usually used for raster or vector data

LiteralValue Used for simple text strings

BoundingBoxValue Two coordinate pairs of lower-left and upper-right corners in defined coordinate sytem.

Inputs and outputs should usually be defined in the *__init__* method of the process.

ComplexValue input and Output

ComplexValue inputs and outputs are used in WPS to send larger sets of data (usually raster or vector data) into the process or from the process back to the user. The `pywps.Process.WPSProcess.addComplexInput()` method returns an instance of `pywps.Process.InAndOutputs.ComplexInput` for inputs. For outputs, they are called with `pywps.Process.WPSProcess.addComplexOutput()`, which returns `pywps.Process.InAndOutputs.ComplexOutput`.

The `pywps.Process.InAndOutputs.ComplexInput.value` and `pywps.Process.InAndOutputs.ComplexOutput` attributes contain the *file name* of the raster or vector file.

For inputs, consider using the `pywps.Process.InAndOutputs.ComplexInput.getValue()` method, for getting the value of the input, which can be returned as file object or file name.

For outputs, you should definitely use the `pywps.Process.InAndOutputs.ComplexOutput.setValue()` method for setting the results file name. The method accepts a file object as well as a file name.

Sometimes, users are sending the data *as reference* to some URL (e.g. OGC WFS or WCS service). PyWPS downloads the data for you and stores them to a local file. If the client requires reference to the output data, PyWPS will create this for you. PyWPS is able to setup a MapServer instance for you, and return OGC WFS or WCS URLs back to the client. For more on this topic, see using-mapserver.

Even you can (and should) define support data mimetypes (`pywps.Process.InAndOutputs.ComplexInput.formats`), mimetype only is checked. PyWPS does not care about valid schemas or anything else. This should be handled by Your process.

Vector data values

Vectors are usually handled as GML files. You can send any other file format as well, such as GeoJSON, KML or any other vector data. Only condition is: the file should be in text form (so it can fit into XML correctly), if you want to append it as part of the input XML request and everything should be stored in *one* file.

Vectors are the default `pywps.Process.InAndOutputs.ComplexInput.format` of ComplexValue input or output – *text/xml* (GML) is expected.

Note: Some users do want to send ESRI Shapfiles. This is in general not to advisable. Shapefiles are a binary format, which is hard to be used with XML, and it consists out of at least three files shp, shx and dbf.

If you still want to handle shapefiles, you have either to zip everything in one file or define three separate complex inputs.

Example of simple input vector data:

```
self.inputVector = self.addComplexOutput(identifier="in",title="Input file")
```

Example of more complex input vector data:

```
self.gmlOrSimilarIn = self.addComplexInput(identifier="input",
                        title="Input file",
                        abstract="Input vector file, usually in GML format",
                        formats = [
                                    # gml
                                    {mimeType: 'text/xml',
                                    encoding:'utf-8',
                                    schema:'http://schemas.opengis.net/gml/3.2.1/gml.xsd'},
                                    # json
                                    {mimeType: 'text/plain',
                                    encoding: 'iso-8859-2',
                                    schema: None
                                    },

                                    # kml
                                    {mimeType: 'text/xml',
                                    encoding: 'windows-1250',
                                    schema: 'http://schemas.opengis.net/kml/2.2.0/ogckml22.xsd'}
                                    ],
                        # we need at least TWO input files, maximal 5
                        minOccurs: 2,
```

```
                        maxOccurs: 5,
                        metadata: {'foo':'bar','spam':'eggs'}
                    )
```

Raster data values

Sometimes, you need to work with raster data. You have to set the proper `pywps.Process.InAndOutputs.ComplexInput.formats` attribute of supported raster file format. Since rasters are usually in *binary* format, you would usually have to send the data always *as reference*. Fortunately, this is not the case. PyWPS can handle the input data, encoded in Base64 format and once PyWPS needs to send raster data out as part of Execute response XML, they are encoded with Base64 as well.

Example of simple output raster data:

```
self.dataIn = self.addComplexOutput(identifier="raster",
                    title="Raster out",
                    formats=[{"mimeType":"image/tiff"}])
```

LiteralValue input and Output

With literal input, you can obtain or send any type of character string. You will obtain an instance of `pywps.Process.InAndOutputs.LiteralInput` or `pywps.Process.InAndOutputs.LiteralOutput`.

Literal value Inputs can be more complex. You can define a list of allowed values, type of the literal input, spacing and so on.

Note: Spacing is not supported, so you can not currently define the step in allowed values row.

Type

For type settings, you can either use the `types` module, or the Python `type()` function. The default type is *type(0)* – Integer. PyWPS will check if the input value type matches allowed type.

Note: If you need the String type of literal input, PyWPS will always remove everything behind "#", ";", "!", "&" and similar characters. Try to avoid usage of LiteralValue input directly as input for e.g. SQL database or command line programs. This could cause a serious system compromise.

Allowed Values

PyWPS lets you define a list of allowed input values. These can be string, integer or float types. Default values are defined in the list. Ranges are defined as two-items filed in form of *(minimum,maximum)*. For example, we would like to allow values 1,2,3, 5 to 7, and 'spam', the `pywps.Process.InAndOutputs.LiteralInput.values` value would look like:

```
[1,2,3,[5,7],'spam']
```

Default is "*", which means *all values*.

Simple example of LiteralValue output:

```
self.widthOut = self.addLiteralOutput(identifier = "width",
                    title = "Width")
```

Complex example of LiteralValue input:

```
self.litIn = self.addLiteralInput(identifier = "eggs",
                title = "Eggs",
                abstract = "Eggs with spam and sausages",
                minOccurs = 0,
                maxOccurs = 1,
                uoms = "m",
                dataType=type(0.0),
                default=1.1,
                values=[(0.0,10.1)])
```

BoundingBoxValue input and Output

BoundingBox are two pairs of coordinates, defined in some coordinate system (2D or 3D). In PyWPS, they are defined in `pywps.Process.InAndOutputs.BoundingBoxInput` and `pywps.Process.InAndOutputs.BoundingBoxOutput`. For getting them, use `pywps.Process.WPSProcess.addBBoxInput()` and `pywps.Process.WPSProcess.addBBoxOutput()` respectively.

The value is a list of four coordinates in *(minx, miny, maxx, maxy)* format.

Example of BoundingBoxValue input:

```
self.bbox = self.addBBoxOutput(identifier = "bbox",
                        title = "BBox")
```

4.2.3 Execution of the process

Each process has to define a `pywps.Process.WPSProcess.execute()` method, which makes the calculation and sets the output values. This method is called via a WPS Execute request.

In principle, you can do what ever you need within this method. It is advised, to use python bindings to popular GIS packages, like GRASS GIS, GDAL/OGR, PROJ4, or any other Python-GIS package.

In the special chapter, we give you a quick intro to some of these packages. Some examples are also distributed along with the PyWPS source.

If you need to run some shell programs, you should use the `pywps.Process.WPSProcess.cmd()` method.

Below is an example of a simple execute method which transforms a raster file from one coordinate system to another, using Python bindings to GDAL.:

```
from osgeo import gdal
from osgeo.gdalconst import *

...

def execute(self):
    """Convert input raster file to PNG using GDAL"""
```

```
    # create gdal input dataset
    indataset = gdal.Open( self.inputRaster.getValue())

    # get output driver for PNG format
    pngDriver = gdal.GetDriverByName("png")

    # define output gdal dataset
    outfile = "output.png"
    outdataset = pngDriver.CreateCopy(outfile, indataset)

    self.outputRaster.setValue(outfile)
    self.outputRaster.format = {'mimeType':"image/png"}

    return
```

CHAPTER 5

Special PyWPS Topics

How to use PyWPS with other packages and projects

5.1 PyWPS and GRASS GIS

PyWPS was originally written with support for GRASS GIS. The processes can be executed within a temporary created GRASS Location or within an existing GRASS Location, within temporary created Mapset. If you are not familiar with this concepts, please review the GRASS documentation.

5.1.1 Configuring PyWPS

First you have to configure PyWPS configuration file, as described in *Configuration*.

5.1.2 Allowing Process to be executed in the GRASS environment

When you are initializing a new process (see process-initialization), you can add a `pywps.Process.WPSProcess.grassLocation` attribute to it.

The attribute can have the following values:

None GRASS Location is not created, GRASS environment is not started (default):

```
WPSProcess.__init__(self, identifier = "foo)
```

True Temporary GRASS Location is created in XY coordinate system. .. note:: In the future, GRASS Location will probably have a

coordinate system token from the input raster or vector file.:

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

String Name of the GRASS Location within the configured grassdbase. If the name starts with "/", the full path to the location is taken, without any other configuration.:

```
WPSProcess.__init__(self, identifier = "foo",
    ...
    grassLocation = "spearfish60")
```

or:

```
        WPSProcess.__init__(self,
            identifier = "foo",
            ...
            grassLocation = "/foo/bar/grassdata/spearfish60")
```

5.1.3 Running GRASS modules from PyWPS

You have two options: either run GRASS modules as you would do in shell script (running the modules directly) or access the GRASS-python interface.

Running GRASS command line modules

Once the `pywps.Process.WPSProcess.execute()` method is executed, you can use the `pywps.Process.WPSProcess.cmd()` method for calling GRASS modules.

Using GRASS-Python interface

Since GRASS 6.4, Python bindings are supported. There are both a ctypes interface and GRASS Modules-Python interface. They are both described in the GRASS Wiki . There are `grass.run_command()`, `grass.mapcalc()` and other useful methods.

GRASS-Python interface example

```
from pywps.Process import WPSProcess

process =  WPSProcess(identifier="grassprocess",
                      title="GRASS Process")

def execute():
    from grass.script import core as grass

    ret = grass.run_command("d.his", h_map = "drap_map",
                                     i_map = "relief_map",
                                     brighten = 0)
    return

process.execute = execute
```

5.2 PyWPS and UMN MapServer

PyWPS can integrate MapServer to return results of ComplexData back to the client.

The idea is as follows: if the client requires `pywps.Process.InAndOutputs.ComplexOutput` to be returned, *as reference*, usually, a direct link to the produced file is returned. But with MapServer, a WFS, WMS or WCS URL could be returned.

The client can later parse the URL of the resulting *ComplexValue* file and e.g. instead of having a GeoTIFF file (result of the calculation), obtained from the WCS, it can request a PNG file via WMS.

5.2.1 Requirements

To support MapServer for ComplexOutputs in your PyWPS installation, the following packages have to be installed:

- python-mapscript
- python-gdal
- python-pyproj

5.2.2 Usage

When you are initializing a new process (see process-initialization), you can set the `pywps.Process.InAndOutputs.ComplexOutput.useMapscript` attribute to *True* to get it run. Have a look at the `pywps.Process.InAndOutputs.ComplexOutput` documentation, also for other attributes, like projection or bbox (can be set automatically from georeferenced file). Required format (`pywps.Process.InAndOutputs.ComplexOutput.format` decides on the output service type (WCS for GeoTIFF and similar, WFS for xml or text, WMS for PNG, JPEG, GIF).:

```
self.outputMap = self.addComplexOutput(identifier = "map",
                title = "Resulting output map",
                formats = [
                        {"mimeType":"image/tiff"},
                        {"mimeType":"image/png"}
                ],
                useMapscript=True)
```

5.3 PyWPS and Mod Python

5.4 PyWPS and WSGI

For more detailed information about WSGI, please visit their *website <http://wsgi.org/wsgi/>_*. In general WSGI is preferred over mod_python mode.

- Install *mod_wsgi* for Apache server (if you are using it)
- Locate *webservices/wsgi/wsgiwps.py* which provides the WSGI interface
- Configure Apache server to something similar as:

```
     SetEnv PYTHONPATH /usr/local/src/pywps/ # is not installed the 'clean' way
     SetEnv PYWPS_CFG /usr/local/src/pywpsworkdir/pywps.cfg
     SetEnv PYWPS_PROCESSES /usr/local/src/pywpsworkdir/processes
    <Directory /usr/local/src/pywps/>
        Order allow,deny
        Allow from all
    </Directory>
    WSGIScriptAlias /wps /usr/local/src/pywps/webservices/wsgi/wpswsgi.py
```

CHAPTER 6

WPS Clients

In this chapter, several (Py)WPS clients will be described. Some of them are part of the PyWPS distribution, others can be found on the Internet.

6.1 PyWPS JavaScript client

Part of the PyWPS distribution includes a generic WPS client based on OpenLayers. The client *does not show any results in a map*, however it enables you to program the client easily. The client is located in `pywps-source/webclient/WPS.js`. In addition, OpenLayers must be included in the web page.

6.1.1 Initialization and GetCapabilities request

To initialize the WPS object, the service URL is required. This example can be found in `wpsclient/01-init.html`.

```
// set the proxy
OpenLayers.ProxyHost = "/cgi-bin/proxy.cgi?url=";

// set the url
var url = "http://foo/bar/wps.py";

// init the client
wps = new OpenLayers.WPS(url);

// run get capabilities
wps.getCapabilities(url);
```

6.1.2 Parsing GetCapabilities response

You must define a function to handle the GetCapabilities response.

```
wps = new OpenLayers.WPS(url, {onGotCapabilities: onGetCapabilities});

/**
 * This function is called, when GetCapabilities response
 * arrived and was parsed
 **/
function onGetCapabilities() {
```

```
    var capabilities = "<h3>"+wps.title+"</h3>";
    capabilities += "<h3>Abstract</h3>"+wps.abstract;
    capabilities += "<h3>Processes</h3><dl>";

    // for each process, get identifier, title and abstract
    for (var i = 0; i < wps.processes.length; i++) {
        var process = wps.processes[i];

        capabilities += "<dt>"+process.identifier+"</dt>";
        capabilities += "<dd>"+"<strong>"+process.title+"</strong><br />"+
                        process.abstract+"</dd>";
    }

    capabilities += "</dl>";

    document.getElementById("wps-result").innerHTML = capabilities;
};
```

6.1.3 Parsing DescribeProcess response

The DescribeProcess request requires the identifier of the process. You can obtain available processes from the GetCapabilities response (described previously). The `onDescribedProcess()` must be defined. This example can be found in `wpsclient/02-describe.html`.

```
wps = new OpenLayers.WPS(url, {onDescribedProcess: onDescribeProcess});

// run get capabilities
wps.describeProcess("dummyprocess");

/**
 * This function is called, when DescribeProcess response
 * arrived and was parsed
 **/
function onDescribeProcess(process) {

    var description = "<h3>"+process.title+"</h3>";
    description += "<h3>Abstract</h3>"+process.abstract;
    description += "<h3>Inputs</h3><dl>";

    // for each input
    for (var i = 0; i < process.inputs.length; i++) {
        var input = process.inputs[i];
        description += "<dt>"+input.identifier+"</dt>";
        description += "<dd>"+"<strong>"+input.title+"</strong><br />"+
                        input.abstract+"</dd>";
    }
    description += "</dl>";
    description += "<h3>Outputs</h3><dl>";

    // for each input
    for (var i = 0; i < process.outputs.length; i++) {
        var output = process.outputs[i];
        description += "<dt>"+output.identifier+"</dt>";
        description += "<dd>"+"<strong>"+output.title+"</strong><br />"+
                        output.abstract+"</dd>";
    }
    description += "</dl>";
```

```
    document.getElementById("wps-result").innerHTML = description;
};
```

6.1.4 Calling Execute request

The Execute request requires the identifier, inputs and outputs parameters. You can obtain available processes and their inputs and outputs from the GetCapabilities and DescribeProcessj response (described previously). The `onSucceeded()` must be defined.

Defining Inputs and Outputs for the process 'by hand'

In this example, we will define Inputs and Outputs of the process "by hand", instead of gathering this information automatically via GetCapabilities and DescribeProcess.

The 'by hand' process initialization consists of three steps:

1. Definition of process Inputs and Outputs
2. Definition of the Process itself
3. Adding a process to the WPS instance

This example can be found in `wpsclient/03-execute.html`.

```
// WPS object
wps = new OpenLayers.WPS(url,{onSucceeded: onExecuted});

// define inputs of the 'dummyprocess'
var input1 = new OpenLayers.WPS.LiteralPut({identifier:"input1",value:1});
var input2 = new OpenLayers.WPS.LiteralPut({identifier:"input2",value:2});

// define outputs of the 'dummyprocess'
var output1 = new OpenLayers.WPS.LiteralPut({identifier:"output1"});
var output2 = new OpenLayers.WPS.LiteralPut({identifier:"output2"});

// define the process and append it to OpenLayers.WPS instance
var dummyprocess =  new
OpenLayers.WPS.Process({identifier:"dummyprocess",
                        inputs: [input1, input2],
                        outputs: [output1, output2]});

wps.addProcess(dummyprocess);

// run Execute
wps.execute("dummyprocess");
```

Of course, func:*onExecuted* has to be defined:

```
/**
 * This function is called, when DescribeProcess response
 * arrived and was parsed
 **/
function onExecuted(process) {
    var executed = "<h3>"+process.title+"</h3>";
    executed += "<h3>Abstract</h3>"+process.abstract;

    executed += "<h3>Outputs</h3><dl>";
```

```
    // for each output
    for (var i = 0; i < process.outputs.length; i++) {
        var output = process.outputs[i];
        executed += "<dt>"+output.identifier+"</dt>";
        executed += "<dd>Title: <strong>"+output.title+"</strong><br />"+
                        "Abstract: "+output.abstract+"</dd>";
        executed += "<dd>"+"<strong>Value:</strong> "+
                        output.getValue()+"</dd>";
    }
    executed += "</dl>";
    document.getElementById("wps-result").innerHTML = executed;

};
```

Defining Inputs and Outputs for the process automatically

In this example, we will define Inputs and Outputs of the process automatically, using the GetCapabilities and DescribeProcess requests.

This example can be found in `wpsclient/04-execute-automatic.html`.

Call DescribeProcess first:

```
// init the client
wps = new OpenLayers.WPS(url,{
            onDescribedProcess: onDescribeProcess,
            onSucceeded: onExecuted
        });

// run Execute
wps.describeProcess("dummyprocess");

/**
 * DescribeProcess and call the Execute response
 **/
function onDescribeProcess(process) {
    process.inputs[0].setValue(1);
    process.inputs[1].setValue(2);

    wps.execute("dummyprocess");
};
```

The rest was already defined before.

CHAPTER 7

PyWPS API

The *pywps* package consists of several sub-packages and classes:

7.1 Module Exceptions

7.2 Module GRASS

7.3 Module Parser

Parser classes used by parsing of OGC WPS Requests

Particular request packages:

7.3.1 Module GetCapabilities

Class Post

Class Get

7.3.2 Module DescribeProcess

Class Post

Class Get

7.3.3 Module Execute

Class Post

Class Get

7.3.4 Module Parser

7.3.5 Module Get

7.3.6 Module Post

7.4 Module Process

7.4.1 Module Lang

Class Lang

7.4.2 Module InAndOutputs

Class Input

Class LiteralInput

Class ComplexInput

Class BoundingBoxInput

Class Output

Class LiteralOutput

Class ComplexOutput

Class BoundingBoxOutput

7.4.3 Class Status

7.4.4 Class WPSProcess

7.5 Module Soap

7.6 Module Wps

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

E

G

P