Implementation of OGC's WPS standard: PyWPS

Jachym Cepicky

October 2, 2007

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.

PyPWS 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 0.4.0 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 work by it self. As working tool, GRASS GIS, GDAL, PROJ, R and other programs can be used.

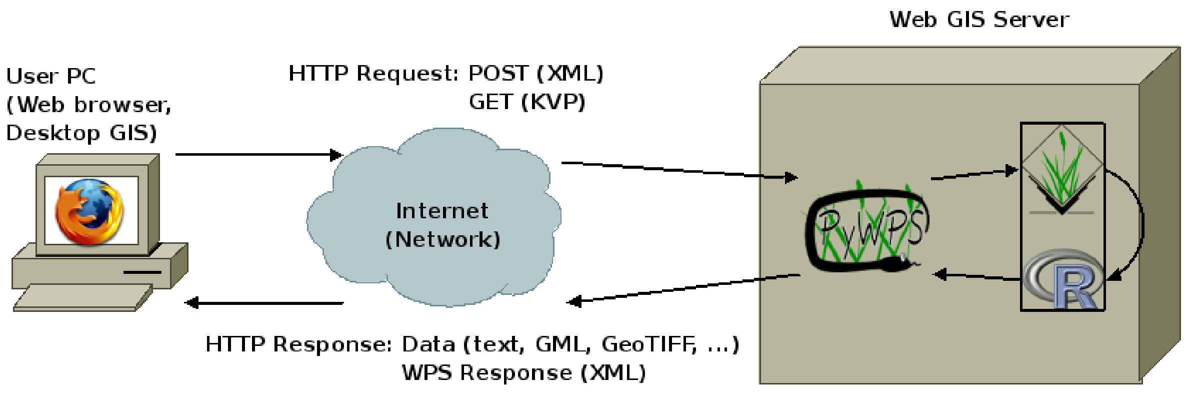

Figure 1: How does PyWPS work: GRASS GIS is in this case working tool

2 Quick install

1. Install PyWPS, see page 4 for details
2. NOTE: Copy 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__` structure.

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

- Sometimes, when there is e.g. SyntaxError in the process, teporary file `/tmp/pywps*` is not deleted, which ledts to `ServerBussy` exception and the files have to be removed by hand.

- If inputs are of type `LiteralValue` and it's type is string, it is not controlled properly. 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

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
```

Several binary packages for Linux distributios 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 05-007r4) version 0.4.0[10].

NOTE: Note, that the configuration option are CASE SENSITIVE

Pywps configuration takes places in two files. The files are actually python scripts, so it does not harm, if you have some experience in python programming language. But you should be able to setup the program without any python knowledge.

The first file is in `pywps/etc/settings.py` and (optional) the second file is `pywps/etc/grass.py` which has to be setuped if you do want to use GRASS GIS modules in your scripts. Some special "tuning" can be done in `pywps/processes/__init__.py` file. You can allways obtain original configuration files from `pywps/Wps/default_settings.py` and `pywps/Wps/default_grass.py`.

5.1 Main configuration file `pywps/etc/settings.py`

This file has got two sections: WPS and serverSettings

In the `WPS` section, the main configuration is set, which appears mostly in GetCapabilities request. The *mandatory* parameters, which should be set up are (with default/recommend values):

```
WPS = {
    'version': "0.4.0",
    'encoding': "utf-8",
    'ServiceIdentification': {
        'Title':"Jachym's WPS server",
```

[9] http://pywps.wald.intevation.org

[10] http://www.opengeospatial.org/standards/requests/28

```
        'ServiceType':"WPS",
        'ServiceTypeVersion':"0.1.0",
        'Abstract':'Abstract to this WPS',
    },
    'ServiceProvider': {
            'ProviderName' : "Your Company",
            'IndividualName':"Your Name",
            'PositionName':"Your Position",
            'Role':"your role",
            'DeliveryPoint': "Street",
            'City': "City",
            'PostalCode':"00000",
            'Country': "Your country",
            'ElectronicMailAddress':"your.email@address",
    },

    'OperationsMetadata': {
        'ServerAddress' : "http://localhost/cgi-bin/wps/wps.py",
    },
    'Keywords' : ['GRASS','GIS','WPS'],
}
```

In the ServerSettings section, the variables are set, which have impact on the whole server.

```
ServerSettings = {
        # NOTE: You have to create this directory manually and set rights, so
        #       the program is able to store data in there
        'outputPath': '/var/www/wpsoutputs',

        #
        # 'outputUrl' - URL of the directory, where the outputs will be stored
        'outputUrl':  'http://192.168.1.31/wpsoutputs',

        #
        # tempPath - path to directory, where temporary data will be stored.
        # NOTE: the pywps has to have rights, to create directories and files
        #       in this directory
        'tempPath': '/tmp',

        #
        # maxOperations - maximum number of operations, which is allowed to low
        # on this server at ones
        # default = 1
        'maxOperations':1,

        #
        # maxSize: maximum input file size in bytes
        # NOTE: maximum file size is 5MB, no care, if this number is higher
        'maxSize':5242880, # 5 MB

        #
        # maxInputParamLength: maximal length of input values
        # NOTE: maximum length of input parameters is 256, no matter, how height
        #       is this number
        'maxInputParamLength':256,
}
```

5.2 etc/grass.py

This file servers for configuration of GRASS GIS environment (if your processes need one). Everything is stored in `grassenv` structure.

```
grassenv = {
    # PATH in which your modules (processes) should be able the search.
    # Default value:
    'PATH': "/usr/local/grass-6.1.cvs/bin/:/usr/local/grass-6.1.cvs/scripts/:\
    /usr/bin/:/bin/:",

    # Add eventually some other path, in which should GRASS search for modules
    'GRASS_ADDON_PATH': "",

    # Version of GRASS, you are using
    'GRASS_VERSION': "6.1.cvs",

    # GRASS_PERL, where is your PERL bin installed
    'GRASS_PERL': "/usr/bin/perl",

    # GRASS_GUI should be always "text" unless you know, what you are doing
    'GRASS_GUI': "text",

    # GISBASE is place, where your GRASS installation is
    'GISBASE': "/usr/local/grass-6.1.cvs",

    # LD_LIBRARY_PATH
    'LD_LIBRARY_PATH':"/usr/local/grass-6.1.cvs/lib",

    # HOME has to be set
    'HOME':"/var/www",
}
```

5.3 Testing after installation

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

```
$ ./wps.py
Content-type: text/xml

<?xml version="1.0" ?>
<ExceptionReport version="1.0.0" xmlns="http://www.opengis.net/ows"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <Exception exceptionCode="MissingParameterValue" locator="request"/>
</ExceptionReport>
```

If you got some other message, like e.g.:

```
Traceback (most recent call last):
  File "trunk/index.py", line 53, in ?
    from Wps import wpsexceptions
  File "/home/jachym/prog/pywps/trunk/Wps/wpsexceptions.py", line 8, in ?
    from xml.dom.minidom import Document
ImportError: No module named xml.dom.minidom
```

Than something is wrong with your Python installation or with the program. This message means, that the xml.dom.minidom package is not installed in your system.

6 Add your own processes

NOTE: This section has changed from previous stable 1.0.0 version. The processes, you defined for 1.0.0 branch should work for 2.0.0 branch too.

All processes are stored in the `processes` directory. Put your file e.g. `myprocess.py` in there. Several example processes are distributed along with PyWPS source code.

Process is python class derived from prepared `WPSProcess` class in `pywps.Wps` package. In it `__init__` method, process metadata, inputs and outpus are defined and in `execute(self)` method, own computation is performed.

It is possible also to add as many your functions, as you wish.

6.1 Process initialization and configuration

First of all, you have to add name of your process to `pywps/processes/__init__.py` file. Then you can start to edit the process file it's self.

```
# importing necessary files
import pywps.Wps.WPSProcess

class Process(WPSProcess):
     def __init__(self):
         WPSProcess.__init__(self,
               Identifier="ogr2ogr",
               Title="ogr2ogr interface",
               Abstract="Convert vector file to another format",
               processVersion = "0.2",
               statusSupported="true",
               storeSupported="true")
```

We defined new process called `ogr2ogr`. 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`).

Eventually optional attributes can be found in the table 38 - "Parts of ExecuteResponse data structure" in the WPS reference document[11]. It is also possible to redefine some variable later, after initialization:

```

            self.Title="ogr2ogr interface"

```

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")
```

This code will produce in DescribeProcess responce document following element:

```
...
<ows:Metadata Identifier="point" type="point">
    Click in the map
</ows:Metadata>
...
```

6.1.1 Data Inputs

Data inputs are stored in `self.Inputs` array. To add inputs to your process, you should use methods defined in `WPSProcess` class.

Four 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
- ComplexValueReference Input – URL to location, where the process is supposed to get the input data.
- BoundingBox Input – Coordinates for lower-left and upper-right corner.

ComplexValue and ComplexValueReference defined on the same way – PyWPS is able to guess, if the input data are reference (link) to some map or raw data directly.

[11] http://www.opengeospatial.org/standards/requests/28

LiteralInput Basic type of data input is `LiteralInput` type. To define LiteralInput the easy way, you should use `AddLiteralInput` method:

```

        self.AddLiteralInput(Identifier="value",
                             Title="Value to be added",
                             type=type(0))
```

Above example will add new input with identifier `value` of type integer. Examples of other possibilities of LiteralInputs and resulting part of XML are folowing:

Example of any allowed input value (default)

```
self.AddLiteralInput(Identifier="someinput",
                     Title="Some Input",
                     allowedvalues='*')
```

```
...
  <Input>
    <ows:Identifier>someinput</ows:Identifier>
    <ows:Title>Some Input</ows:Title>
    <ows:Abstract/>
        <LiteralData>
            <SupportedUOMs defaultUOM="m">
                <ows:UOM>m</ows:UOM>
            </SupportedUOMs>
            <ows:AnyValue/>
        </LiteralData>
    <MinimumOccurs>1</MinimumOccurs>
  </Input>
...
```

Example of specified list (with range) of allowed inputs Following example will define input with specified list of values: Only values 20, 30, everything between 40-100 and 110 will be accepted:

```
self.AddLiteralInput(Identifier="someinput",
                     Title="Some Input",
                     allowedvalues=[20,30,[40,100],110])
```

```
...
  <Input>
    <ows:Identifier>someinput</ows:Identifier>
    <ows:Title>Some Input</ows:Title>
    <ows:Abstract/>
    <LiteralData>
     <SupportedUOMs defaultUOM="m">
        <ows:UOM>m</ows:UOM>
        </SupportedUOMs>
        <AllowedValues>
            <Value>20</Value>
            <Value>30</Value>
            <Range>
                <MinimumValue>40</MinimumValue>
                <MaximumValue>100</MaximumValue>
            </Range>
            <Value>110</Value>
        </AllowedValues>
    </LiteralData>
    <MinimumOccurs>1</MinimumOccurs>
  </Input>
...
```

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`[12] file distributed along with PyWPS source code.

ComplexInput If the request comes as HTTP GET, it is assumed, that the input is only reference to some map. If it comes as HTTP POST, PyWPS tryes to guess, if the client is sending URL to source of the data or if the input data are part of input XML request (e.g. as GML file). So, you, as a process coder do not have to take care on this:

```
self.AddComplexInput(Indentifier="inputmap",
      Title="Input map, which should be processed",
      Formats=["text/xml","image/tiff"])
...
    <Input>
        <ows:Identifier>input</ows:Identifier>
        <ows:Title>Input raster map</ows:Title>
        <ows:Abstract/>
        <ComplexData defaultFormat="image/tiff">
            <SupportedComplexData>
                <Format>image/tiff</Format>
                <Format>text/xml</Format>
            </SupportedComplexData>
        </ComplexData>
        <MinimumOccurs>1</MinimumOccurs>
    </Input>
```

BoundingBox Input With bounding box, you can define two coordinate pairs, if you have to.

```
self.AddBondingBoxInput(Identifier="bbox",
        Title="BBox input")
```

[12] Documentation to process.py module

6.1.2 Data Outputs

Again four types of output are defined:

- Literal Output
- ComplexValue Output
- ComplexValue Reference
- BoundingBox Output

Data outputs can be defined on similar way, using similar methods:

LiteralOutput

```
self.AddLiteralOutput(Identifier="litoutput",
                     Title="Resulting output value")

...
  <Output>
    <ows:Identifier>litoutput</ows:Identifier>
    <ows:Title>Resulting output value</ows:Title>
    <ows:Abstract/>
    <LiteralOutput>
      <SupportedUOMs defaultUOM="m">
        <ows:UOM>m</ows:UOM>
      </SupportedUOMs>
    </LiteralOutput>
  </Output>
...
```

ComplexValue and ComplexValueReference Output To the oposite of data Inputs, Outputs can distinguish between ComplexValue output and ComplexValueReference. ComplexValue is directly embed into the output XML document and ComplexValueReference is stored on the server and only URL pointing the the file is refering to it. In general, vector files in GML format can be easy embed to the output XML, TIFF raster files is better to leave on the server.

```
self.AddComplexReferenceOutput(Identifier="output",
                Title="Resulting output map",
                Formats=["image/tiff"])

...
  <Output>
    <ows:Identifier>output</ows:Identifier>
    <ows:Title>Resulting output map</ows:Title>
    <ows:Abstract/>
    <ComplexOutput defaultFormat="image/tiff">
```

```
      <SupportedComplexData>
        <Format>image/tiff</Format>
      </SupportedComplexData>
    </ComplexOutput>
  </Output>
...
```

BoundingBox Output Beside LiteralValue and ComplexValue, BoundingBoxValue is also defined. The coordinates are stored in array of four members:

```
self.GetInputValue("bboxinput")
[0,0,100,100]
```

So on our `ogr2ogr` process, we have to define three types of input: `ComplexValue` of input vector file and EPSG codes of target and source files:

```
    self.AddComplexInput(Identifier="inputmap",
        Title ="Input vector file",
        Abstract = "Input vector file to be converted",
        Formats=["text/xml"])

    self.AddLiteralInput(Identifier="sepsg",
        Title="Source EPSG",
        Abstract="Source EPSG code",
        value=4326)
```

And we also define two outputs: ComplexValueReference and ComplexValue type.

```
    self.AddComplexOutput(Identifier="outputmap",
        Title ="Input vector file",
        Abstract = "Input vector file to be converted",

    self.LiteralOutput(Identifier="sepsg",
        Title="Source EPSG",
        Abstract="Source EPSG code",
        value=4326)
```

6.2 Process Programming

The process must be defined in the `execute(self)` function. You can access the input values via `self.GetInputValue(Identifier)` method.

NOTE: Usage of the old method, accessing the values via `self.Inputs[index]['value']` or via `self.DataInputs` array is possible, but should not be used.

Also variable `self.grassenv` will be in your process at your service. This dictionary stores environment variables used by GRASS GIS, such se `LOCATION_NAME` or `MAPSET`.

Output values should be set using `self.SetOutputValue(Identifier, value)` method.

NOTE: Usage of the old methods of output values setting, directly to `self.Outputs[index]['value']` variable or to `self.DataOutputs['identifier']` dictionary, is possible, but should better not be used.

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.

```

    def execute(self):

        #
        # calculation
        #
        self.Cmd("""ogr2ogr -s_srs "+init=epsg:4326" -t_srs \
        "+init=epsg:2065" %s output.file""" % (self.GetInputValue("inputmap")))
       #
       # setting results
       #
       self.SetOutputValue("outputmap","output.file")
       self.SetOutputValue("sepsg","4326")

       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.

6.2.2 Using standard in- and output with external commands

The `self.Cmd()` accepts `input` parameter, wich is a text string, which is directred to standard input of the command:

```
result = self.Cmd(cmd="wc -c",
         input="calculate number of characters for this sentence")

# result[0].split()
```

`self.Cmd()` returns list of lines from the programms standard output to the process:

```
for line in self.Cmd("ls -l"):
    # do some operations of list of files
    pass
```

6.3 GRASS specific notes

Special class for GRASS GIS is defined too, which has functions and variables specific to this program. The process, in which should use GRASS modules should be defined as follows:

```
# importing necessary files
import pywps.Wps.GRASSWPSProcess

class Process(GRASSWPSProcess):
     def __init__(self):
         GRASSWPSProcess.__init__(self,
               Identifier="spearpath",
               Title="Spearfish path searching",
               Abstract="Find the shortest path on the roads map on Spearfish dataset",
               processVersion = "0.2",
               statusSupported="true",
               storeSupported="true",
               # grassLocation="/var/www/spearfish60/" # work on existing location)
```

By default, `self.grassLocation`[13] variable is set to `None`, which means, that temporary location will be created and after the calculation is done, it will be deleted again. You can set this while process initialization or later[14].

`WPSProcess` class provides special method `self.GCmd(command_string)`, which tryes to catch output from GRASS modules, especially progress information inidcated by percent done. Method `GCmd()` stores the output of GRASS modules to `self.status` variable, so if the process is running assynchronously, client application can track the progress of each module directly.

```
    def execute(self):
        """
        This function serves like simple GRASS - python script

        It returns None, if process succeed or String if process failed
        """
        self.GCmd("g.region -d")


        # v.net.path reads from standard input
        self.GCmd("v.net.path in=roads out=path","0 %s %s %s %s" % (self.GetInputValue('x1
                self.GetInputValue('y1'),
                self.GetInputValue('x2'),
                self.GetInputValue('y2')))

        self.GCmd("v.out.ogr format=GML input=path dsn=out.xml olayer=path.xml")

        if "out.xml" in os.listdir(os.curdir):
            shutil.copy("out.xml","out2.xml")
            self.SetOutputValue('outputReference',"out.xml")
            self.SetOutputValue('outputData',"out2.xml")
            return
```

[13] See e.g. GRASS manual for details

[14] `self.grassLocation="/path/to/location"`

```
        else:
            return "Ouput file not created!"
```

It is also possible to run GRASS modules using python's `os.system()` or `os.popen()` function. Before you do so, it is important to import the `os` python package (usually one of the first lines in the file). This approach might not be the best, but it is the simplest one. Feel free to use any other low-end functions.

Unfortunately, the GRASS modules are very verbose. Some messages are written to STDOUT, some to STDERR. The STDERR will be stored in the error file of your web server. But you have to "catch" the messages, sent to STDOUT. This can be done e.g. by using "1 > &2" statement (redirecting STDOUT to STDERR in shell):

```
os.system("""
    echo "Rekni jim drazi, tatko, za to nic nedas."  >&2
""")
```

You can avoid this problem using formentioned `self.GCmd()` method.

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=0.4.0&service=Wps&request=DescribeProcess&\
    Identifier=your_process"

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

- Execute request:

```
./wps.py "version=0.4.0&service=Wps&\
    request=Execute&Identifier=your_process&\
    datainputs=input1,value1,input2,value2"

wget -nv -q -O - "http://localhost/cgi-bin/wps.py?\
    version=0.4.0&service=Wps&\
    request=Execute&Identifier=your_process&\
    datainputs=input1,value1,input2,value2" \
```

8 Using PyWPS

8.1 Input

To get response from PyWPS you have to formulate appropriate query string first. You can use HTTP GET style or HTTP POST style.

HTTP GET style is standard URL, with all parameters in one line. You can not set any `ComplexValue` data in your process via HTTP GET. Example:

```
wget -nv -q -O - --post-data="version=0.4.0&service=Wps&\
        request=Execute&Identifier=your_process&\
        datainputs=input1,value1,input2,value2"\
        "http://localhost/cgi-bin/wps.py"
```

In HTTP POST style, you send one "request" parameter, which contains XML input. The XML file can contain also included ComplexValue data, e.g. GML file. Example:

```
wget --post-file=execute-post.txt \
        "http://localhost/pywps/wps.py" -O - -nv -q
```

The `execute-post.txt` file can look like follows:

```
<?xml version="1.0" encoding="utf-8"?>
<Execute service="WPS" version="0.4.0" store="false" status="false"
xmlns="http://www.opengeospatial.net/wps"
xmlns:ows="http://www.opengeospatial.net/ows"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengeospatial.net/wps/wpsDescribeProcess.xsd">
    <ows:Identifier>searchpath</ows:Identifier>
    <DataInputs>
        <Input>
            <ows:Identifier>streetmap</ows:Identifier>
            <ows:Title>The map</ows:Title>
            <ows:ComplexValue>
                <Value>
<ogr:FeatureCollection
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://ogr.maptools.org/ donut.xsd"
     xmlns:ogr="http://ogr.maptools.org/"
     xmlns:gml="http://www.opengis.net/gml">
  <gml:boundedBy>
    <gml:Box>
      <gml:coord><gml:X>4.263256414560601e-14</gml:X>
        <gml:Y>-70.71067811865474</gml:Y></gml:coord>
      <gml:coord><gml:X>141.4213562373095</gml:X>
      <gml:Y>70.71067811865474</gml:Y></gml:coord>
    </gml:Box>
  </gml:boundedBy>
  <gml:featureMember>
    <ogr:donut fid="F0">
      <ogr:geometryProperty><gml:LineString><gml:coordinates>
      70.710678118654755,70.710678118654741,0 141.42135623730951,0.0,
      0 70.710678118654741,-70.710678118654741,0 0.000000000000043,
      0.000000000000057,0 70.710678118654755,
      70.710678118654741,0</gml:coordinates>
      </gml:LineString></ogr:geometryProperty>
    </ogr:donut>
  </gml:featureMember>
  <gml:featureMember>
    <ogr:donut fid="F0">
      <ogr:geometryProperty><gml:LineString><gml:coordinates>50.000000000000014,
      0.000000000000021,0 71.213203435596427,-21.213203435596419,0
      92.426406871192853,0.0,0 71.213203435596427,21.213203435596423,0
      50.000000000000014,0.000000000000021,0</gml:coordinates>
      </gml:LineString></ogr:geometryProperty>
    </ogr:donut>
  </gml:featureMember>
</ogr:FeatureCollection>
                </Value>
            </ows:ComplexValue>
        </Input>

        <Input>
            <ows:Identifier>x1</ows:Identifier>
            <ows:LiteralValue>591679.31</ows:LiteralValue>
        </Input>
        <Input>
            <ows:Identifier>y1</ows:Identifier>
            <ows:LiteralValue>4927205.07</ows:LiteralValue>
        </Input>
        <Input>
            <ows:Identifier>x2</ows:Identifier>
            <ows:LiteralValue>608642.625</ows:LiteralValue>
        </Input>
        <Input>
            <ows:Identifier>y2</ows:Identifier>
            <ows:LiteralValue>4915876.31</ows:LiteralValue>
        </Input>
    </DataInputs>
</Execute>
```

You can see, that there are 4 inputs in this process:

1. ComplexValue GML File

2. x1 coordinate
3. y1 coordinate
4. x2 coordinate
5. y2 coordinate

8.2 Output

The output from PyWPS can be either XML file or results of processes directly. In default configuration, no files are stored on the server, resulting values (maps) are returned to the client. If you want to return XML file with outputs encoding, you have to enable it in you process configuration with option `storeSupported`:

```
self.storeSupported = "true"
```

And you have to call the PyWPS with "store=true" option:

```
version=0.4.0&service=Wps&request=Execute&Identifier=your_process&\
                datainputs=input1,value1,input2,value2&store=true
```

Or in XML input:

```
<?xml version="1.0" encoding="utf-8"?>
<Execute service="WPS" version="0.4.0" store="true" status="false"
xmlns="http://www.opengeospatial.net/wps"
xmlns:ows="http://www.opengeospatial.net/ows"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengeospatial.net/wps/wpsDescribeProcess.xsd">

...
```

This will cause PyWPS to look after `self.status` array in your process in form form

```
self.status = ["Message", Percent_Done]
```

and generate XML file in statusLocation with this embed message. E.g.

```
self.status = ["Generating raster map", 50]
```

or better

```
self.SetStatus("Generarting raster map", 50)
```

will become

```
...
    <Status>
            <ProcessStarted message="Generating raster map" percentCompleted="50"/>
    </Status>
...
```

A Example process: `addvalue`

This sample process describes how to made your own WPS processes. Purpose of this process is:

- Download input raster map from some server
- Convert it to integer values
- Add input value to each raster cell
- Convert raster to vector
- Export raster to TIFF and vector to GML. Vector file will be embed ot output XML file.

```
"""

pywps process example:

addvalue: Adds some value to raster map
"""
# Author: Jachym Cepicky
#         http://les-ejk.cz
# Lince:        GNU/GPL
#
# Copyright (C) 2006 Jachym Cepicky

import os,time,string,sys,shutil
from pywps.Wps.process import GRASSWPSProcess

class Process (GRASSWPSProcess):
    #
    # Initialization
    #
    def __init__(self):
        GRASSWPSProcess.__init__(self,
                            Identifier="Addvalue",
                    Title="Add some value to input raster map",
                            processVersion = "0.2",
                            statusSupported="true",
                            storeSupported="true",
                            grassLocation = None)


        #
        # Inputs
        self.AddComplexInput(Identifier="input",
                            Title="Input raster map",
                         Formats=["image/tiff"])
        self.AddLiteralInput(Identifier="value",
                         Title="Value to be added",
                         type=type(0))

        #
        # Outputs
        self.AddComplexReferenceOutput(Identifier="output",
                         Title="Resulting output map",
                         Formats=["image/tiff"])
    #
    # Execute part of the process
    #
    def execute(self):
        """
        This function
            1) Imports the raster map
            2) runs r.mapcalc out=in+value
            3) Exports the raster map
            4) returns the new file name or 'None' if something went wrong
        """

        # import of the data
        self.SetStatus("Importing data")
        if not self.GCmd("r.in.gdal -o in=%s out=input" %\
                (self.GetInputValue("input"))):
            return "Could not import raster file"
        self.SetStatus("Importing data",10)

        # compositing 3 bands to one raster file
        for gdalinfoln in os.popen("gdalinfo %s" %\
                (self.GetInputValue("input"))):
            if gdalinfoln.split()[0] == "Band" and gdalinfoln.split()[1] == "3":
                self.GCmd("""g.region rast=input.red """)
                self.GCmd("r.composite r=input.red b=input.blue g=input.green out=input")

        # region setting
        self.GCmd("""g.region rast=input""")

        # adding the value
        self.SetStatus("Adding new value to raster map",50)
        self.GCmd("r.mapcalc output='input+%f'" % float(self.GetInputValue('value')))

        # output
        self.SetStatus("Raster file export", 90)
        self.GCmd("r.out.gdal type=Int32 in=output out=%s" % "output.tif")
        # setting output values
        self.SetOutputValue("output","output.tif")
        if "output.tif" in os.listdir(os.curdir):
            return  # OK
        else:
            return "Output file not created!" # FAILED
        """
```

B KVP request encoding of `addvalue`

This process can be lunched with URL:
http://localhost/cgi-bin/wps.py?service=wps&version=0.4.0&identifier=addvalue&request=execute&\
datainputs=input,http://localhost/data/raster.tif,value,250&status=true&store=true

C XML request encoding `addvalue`

```
request=<?xml version='1.0' encoding='UTF-8' standalone='yes'?>
<Execute service='wps' version='0.4.0' store='true' status='false'
    xmlns="http://www.opengeospatial.net/wps"
    xmlns:ows="http://www.opengeospatial.net/ows">
<ows:Identifier>addvalue</ows:Identifier>
<DataInputs>
    <Input>
        <ows:Identifier>input</ows:Identifier>
        <ComplexValueReference reference='http://localhost/wps/data/soils.tif' />
    </Input>
    <Input>
        <ows:Identifier>value</ows:Identifier>
        <LiteralValue>250</LiteralValue>
    </Input>
    <!-- Input>
        <ows:Identifier>bbox</ows:Identifier>
        <BoundingBoxValue>
            <BoundingBox>
                <LowerCorner>-1 -1</LowerCorner>
                <UpperCorner>10 10</UpperCorner>
            </BoundingBox>
        </BoundingBoxValue>
    </Input -->
</DataInputs>
</Execute>
```