



Project No. 249024

NETMAR

Open service network for marine environmental data

Instrument: <i>Please tick</i>	CA	<input checked="" type="checkbox"/> STREP	IP	NOE
--	----	--	----	-----

ICT - Information and Communication Technologies Theme

D5.3.1 WPS Server – Basic WPS package

Reference: D5.3.1_WPS_Server_r1_20110429.doc

Due date of deliverable (as in Annex 1): M0 + 15

Actual submission date: 29 April 2011

Start date of project: 1 February 2010

Duration: 3 years

Plymouth Marine Laboratory

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	



 	<p>NETMAR Open service network for marine environmental data Project Reference: 249024 Contract Type: Collaborative Project Start/End Date: 01/03/2010 - 31/01/2013 Duration: 36 months</p>
	<p>Coordinator: Prof. Stein Sandven Nansen Environmental and Remote Sensing Center Thormøhlensgate 47, Bergen, Norway Tel.: +47 55 20 58 00 Fax. +47 55 20 58 01 E-mail: stein.sandven@nersc.no</p>

Acknowledgements

The work described in this report has been partially funded by the European Commission under the Seventh Framework Programme, Theme ICT 2009.6.4 ICT for environmental services and climate change adaptation.

Consortium

The NETMAR Consortium is comprised of:

- Nansen Environmental and Remote Sensing Center (NERSC), Norway (coordinator).
Project Coordinator: Prof. Stein Sandven (stein.sandven@nersc.no)
Deputy Coordinator: Dr. Torill Hamre (torill.hamre@nersc.no)
Quality Control Manager: Mr. Lasse H. Pettersson (lasse.pettersson@nersc.no)
- British Oceanographic Data Centre (BODC), National Environment Research Council, United Kingdom
Contact: Dr. Roy Lowry (rkl@bodc.ac.uk)
- Centre de documentation de recherche et d'expérimentations sur les pollutions accidentelles des eaux (Cedre), France.
Contact: Mr. François Parthiot (Francois.Parthiot@cedre.fr)
- Coastal and Marine Resources Centre (CMRC), University College Cork, National University of Ireland, Cork, Ireland.
Contact: Mr. Declan Dunne (d.dunne@ucc.ie)
- Plymouth Marine Laboratory (PML), United Kingdom.
Contact: Mr. Steve Groom (sbg@pml.ac.uk)
- Institut français de recherche pour l'exploitation de la mer (Ifremer), France.
Contact: Mr. Mickael Treguer (mickael.treguer@ifremer.fr)
- Norwegian Meteorological Institute (METNO), Norway.
Contact: Mr. Jan Ivar Pladsen (janip@met.no)

Author(s)

- Jorge Mendes de Jesus, PML, (jmdj@pml.ac.uk)
- Peter Walker, PML, (petwa@pml.ac.uk)
- Mike Grant, PML (mggr@pml.ac.uk)

Document approval

- Document status: Version 1
- WP leader approval: 2011-04-13
- Quality Manager approval: 2011-04-29
- Coordinator approval: 2011-04-29

Revision History

Issue	Date	Change records	Author(s)
Draft 1	2011-04-11	First draft of the report.	Jorge Mendes de Jesus
Draft 2	2011-04-12	Second draft of the report	Jorge Mendes de Jesus, Peter Walker
Draft 3	2011-04-13	Minor formatting changes	Peter Walker
Draft 4	2011-04-28	Incorporated reviewer feedback and added additional overview text	Jorge Mendes de Jesus, Mike Grant
1	2011-04-29	Final release approved	Torill Hamre

Executive Summary

The NETMAR project aims to develop a pilot European Marine Information System (EUMIS) for searching, downloading and integrating satellite, in situ and model data from ocean and coastal areas. EUMIS will use a semantic framework coupled with ontologies for identifying and accessing distributed data, such as near-real time, forecast and historical data. NETMAR will develop a set of data delivery services using standard web-GIS protocols. Processing services and adaptive service chaining services will also be developed, to enable users to generate new products suited to their needs. Both data retrieved from existing systems as well as the products generated dynamically can be accessed and visualised in the EUMIS portal.

This deliverable provides the recommended WPS server implementation that will support the basic processing services in the NETMAR project, required to support initial demonstration of the use cases in the EUMIS portal. The aim of this report is to briefly document the development work carried out by Plymouth Marine Laboratory (PML) and to provide sufficient information for NETMAR partners to install PyWPS (Python API supporting WPS) and set up their own processes.

PyWPS is a WPS (Web Processing Service) implementation written in the Python language. The current stable version 3.1.0 (<http://pywps.wald.intevation.org>) offers WPS 1.0.0 support. PyWPS provides a framework where programmers can deploy their geospatial algorithms. PyWPS's approach is not to offer processes but the means to create them by facilitating access to GRASS (Geographic Resources Analysis Support System) GIS and allowing any Python code to be run and served as WPS.

Since September 2010, PML has been working on an enhanced WPS implementation based on PyWPS. The developed software will provide easier web service integration and functionality inside the NETMAR EUMIS platform and will be fed back to the PyWPS community as a contribution by the NETMAR project.

Newly developed functionality such as WSDL (Web Service Description Language) and SOAP (Simple Object Access Protocol) support enables WPS services to be integrated into orchestration platforms like Taverna or to be used in web service execution languages like BPEL (Business Procedure Execution Language).

The enhanced PyWPS source code can be downloaded from <https://svn.wald.intevation.org/svn/pywps/branches/pywps-3.2-soap/>.

The PyWPS wiki where new development is documented and explained is found at http://wiki.rsg.pml.ac.uk/pywps/Main_Page.

Contents

1	INTRODUCTION.....	4
1.1	RELATIONSHIP OF THIS DELIVERABLE TO THE PROJECT OBJECTIVES.....	4
1.2	BACKGROUND.....	4
1.3	OBJECTIVE OF THIS REPORT.....	4
1.4	TERMINOLOGY.....	4
2	PYWPS OVERVIEW.....	6
3	INSTALLATION.....	7
3.1	PYWPS REQUIRED COMPONENTS.....	7
3.2	PYWPS RECOMMENDED PACKAGES.....	7
3.3	PYWPS INSTALLATION.....	7
3.4	PYWPS PROCESS CREATION.....	8
	3.4.1 Configuration file.....	9
	3.4.2 Apache Installation.....	10
	3.4.3 Testing WPS services.....	10
4	DEVELOPMENT SUMMARY.....	13
4.1	WSDL/SOAP DEVELOPMENT.....	13
4.2	TEST DRIVEN DEVELOPMENT (TDD).....	13
4.3	WPS-GRASS-BRIDGE.....	14
5	WIKI.....	14
6	TAVERNA WORKBENCH.....	15
7	REFERENCES.....	17

1 Introduction

1.1 Relationship of this deliverable to the project objectives

This deliverable provides the recommended WPS server implementation that will support the basic processing services in the NETMAR project, complementing the basic data services already delivered for the initial user demonstrations.

Service providers will use this package to implement basic processing services to meet the requirements of the use cases and to provide demonstration services in the EUMIS portal. A specific goal is to easily support service chaining, which the WSDL work described below does, and promote rapid development of processing services. These services may be made available via international registries, such as GEOSS, and can be used with other standards-based projects.

In the next stages of the project, the semantic framework will be incorporated to allow development of semantically-enabled services (e.g. a temperature-oriented processing service that is intelligent enough to ensure it has relevant, compatible and correct input data representing temperatures). The purpose of an initial non-semantically-enabled release is to allow rapid prototyping and earlier feedback from users.

1.2 Background

Since September 2010, Plymouth Marine Laboratory has been working on a new WPS (Web Processing Service) implementation based on PyWPS (Python API supporting WPS) [GRASS06]. The results will allow easier web service integration and functionalities inside the EUMIS platform, and will be fed back to the PyWPS community from the NETMAR project. The code being developed is located in PyWPS project SVN (Subversion) as a svn-branch and referred to as PyWPS-3.2-SOAP. As of April 2011 the code will migrate to a svn-tag and be an official release.

Newly developed functionalities like WSDL (Web Service Description Language) [WSDL01] and SOAP (Simple Object Access Protocol) [SOAP00] allow WPS services to be integrated into orchestration platforms such as Taverna or to be used in web service execution languages like BPEL (Business Procedure Execution Language).

1.3 Objective of this report

The aim of this report is to briefly document the development work carried out by PML and to provide sufficient information for NETMAR partners to install PyWPS and set up their own processes.

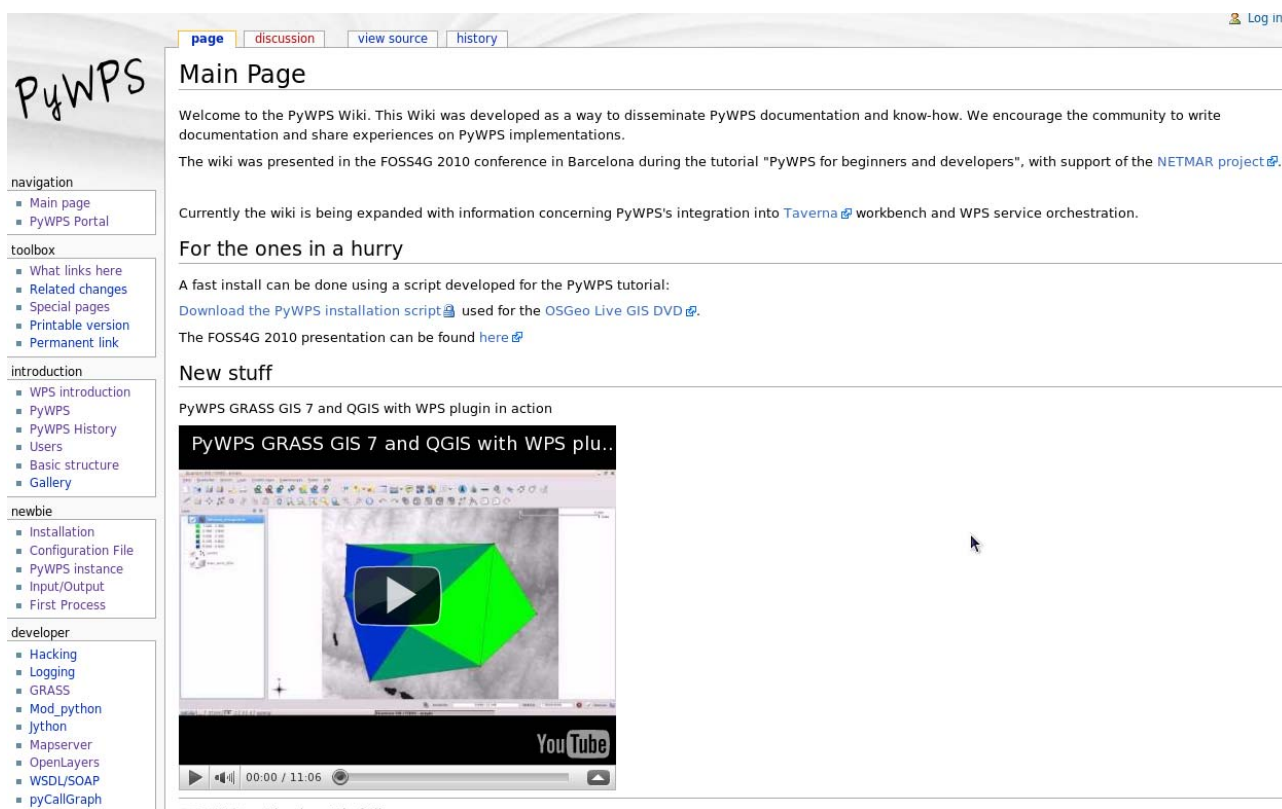
1.4 Terminology

API	Application Programming Interface
BBOX	Bounding Box
BPEL	Business Process Execution Language
CGI	Common Gateway Interface
EUMIS	European Marine Information System
GIS	Geographical Information System
GRASS	Geographic Resources Analysis Support System
HTTP	Hypertext Transfer Protocol
KVP	Key-Value-Pair
OGC	Open Geospatial Consortium
OWS	OGC Web Services
PyWPS-3.2-SOAP	The enhanced PyWPS developed by PML
REST	Representational State Transfer
SCUFL	Simple Conceptual Unified Flow Language

SOAP	Simple Object Access Protocol
SVN	Subversion (version control system)
TDD	Test Driven Development
URL	Uniform Resource Locator
WCS	Web Coverage Service
WFS	Web Feature Service
WMS	Web Map Service
WPS	Web Processing Service
WSDL	Web Services Description Language
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Template

2 PyWPS overview

PyWPS is a WPS implementation written in the Python language. The current stable version 3.1.0 (<http://pywps.wald.intevation.org/>) offers WPS 1.0.0 support. PyWPS provides a framework where programmers can deploy their geospatial algorithms. PyWPS's approach is not to offer processes but the means to create them by facilitating access to GRASS GIS and allowing any Python code to be run and served as WPS. The framework structure is organized into packages and classes that work using a "factory strategy", starting by parsing the inputs submitted to the WPS, offering them to the process source code, retrieving the outputs and finally generating the WPS XML (or raw) response. The factory structure approach and the 5000 lines that implement the framework make it a suitable candidate for testing and developing new WPS features like encryption, XML dynamic transformation, HTTP server support, etc.



The screenshot shows the PyWPS Wiki main page. The page title is "Main Page". The content includes a welcome message: "Welcome to the PyWPS Wiki. This Wiki was developed as a way to disseminate PyWPS documentation and know-how. We encourage the community to write documentation and share experiences on PyWPS implementations." It also mentions the wiki was presented at the FOSS4G 2010 conference in Barcelona. A section titled "For the ones in a hurry" provides a fast install method using a script. A "New stuff" section features a video player with the title "PyWPS GRASS GIS 7 and QGIS with WPS plu..". The video player shows a screenshot of a GIS application with a play button overlay. The page has a sidebar with navigation and toolbox links.

Figure 2-1 PyWPS wiki (<http://wiki.rsg.pml.ac.uk/pywps>) supported by NETMAR project and hosted on PML's servers

3 Installation

3.1 *PyWPS required components*

PyWPS-3.2-Soap requires the following packages

- **Python 2.6**
- **python-lxml 2.2.6-1**

3.2 *PyWPS recommended packages*

- **Apache 2.x web server.** Web server necessary to run PyWPS as a CGI (Common Gateway Interface).
- **GIS GRASS 6.5.** Geographical Resources Analysis Support System (GRASS) is Open Source GIS. PyWPS is written with native support for GRASS and its functions.
- **Mapserver/Mapscript(Python).** Necessary to generate ComplexValue outputs using OGC OWS (WMS, WFS, WCS) services. The **python-mapscript** package provided by several Linux distributions is sufficient.

3.3 *PyWPS installation*

The following installation procedures are based on a Linux operating system, and it is assumed that the user has minimal familiarity with the bash shell.

The PyWPS code is located on the following URL:

<https://svn.wald.intevation.org/svn/pywps/branches/pywps-3.2-soap/>

Basic SVN install

1. The code can be fetched using a svn command from the bash:
\$> svn checkout
https://svn.wald.intevation.org/svn/pywps/branches/pywps-3.2-soap/
2. cd to the directory with source code:
\$> cd ./pywps-3.2-soap/
3. run the Python default installation script:
\$> python setup.py install
4. In some systems the WPS template installation may raise an exception, caused by incorrect path detection and preventing the templates being compiled. In this case, please use a dry run installation:
\$> python setup.py install --dry-run

This install will copy the PyWPS start script to /usr/bin or /usr/local/bin. PyWPS can be run as a normal bash command

```
$> wps.py
```

```
PyWPS NoApplicableCode: Locator: None; Value: No query string found.
```

```
Content-Type: application/xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<ows:ExceptionReport version="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/ows/1.1
http://schemas.opengis.net/ows/1.1.0/owsExceptionReport.xsd">
<ows:Exception exceptionCode="NoApplicableCode">
  <ows:ExceptionText>
    'No query string found.'
  </ows:ExceptionText>
</ows:Exception>
```

When run as a command line program, PyWPS will accept Key-Value-Pair (KVP) inputs, normally used in HTTP-GET requests or an XML request provided on the standard input.

KVP example:

```
$> wps.py "request=GetCapabilities&service=WPS"
```

File request Example:

```
$> export REQUEST_METHOD=POST; cat wps_getcapabilities_request.xml | wps.py
```

Note: PyWPS uses KVP (HTTP-GET) by default and will run a KVP parse on anything sent to the script. The "export REQUEST_METHOD=POST" will activate the PyWPS XML parser.

The wps_getcapabilities_request.xml file is located within the source code tree: pywps-3.2-soap/tests/requests/wps_getcapabilities_request.xml

3.4 PyWPS process creation

PyWPS uses an instance strategy to run WPS services. After the initial installation, each WPS service needs its processes defined (stored in one directory) and a local/specific configuration file for the service. Processes are stored together as Python programs in one directory, following a module structure.

1. Create processes directory to store all process definitions for a particular PyWPS instance:

```
$> mkdir -p /usr/local/pywps/processes
```
2. Copy configuration file-template to some location, and configure your PyWPS installation:

```
$ cp pywps-3.2-soap/pywps/default.cfg /etc/pywps.cfg
$ edit /etc/pywps.cfg
```

Some process examples can be found in the pywps-3.2-soap/examples/processes directory.

Every process in the processes directory, needs to be registered in the `__init__.py` file. The file has to contain at least:

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

Where `all` represents a list of processes (file names) within the processes directory. The following will generate the file and content:

```
$> cd /usr/local/wps/processes/
$> echo "__all__=['ultimatequestionprocess']" > __init__.py
```

Each PyWPS instance is defined by its environment variables:

PYWPS_CFG	Configuration file location
------------------	-----------------------------

PYWPS_PROCESSES Directory, where the processes are stored

The environment variables can be set up in a CGI wrapper script. This will generate a specific WPS instance based on the configuration file and path to processes.

This is a basic example of a CGI wrapper script:

```
#!/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=/etc/pywps.cfg
export PYWPS_PROCESSES=/usr/local/pywps/processes
#assuming that wps.py is in a executable directory (whereis)
wps.py $1
```

3.4.1 Configuration file

The configuration file (in this example set in `/etc/pywps.cfg`) is a section/key-value file and defines WPS properties like process limits (max operations, file size limits, contact point, server address). These parameters are necessary to run a proper WPS instance.

For example:

```
[wps]
encoding=utf-8
title=PyWPS Server
version=1.0.0
abstract=PML WPS server
fees=None
constraints=none
serveraddress=http://rsg.pml.ac.uk/wps/wps.cgi
keywords=PML,NETMAR,RSG,Vector,XML
lang=en-CA

[provider]
providerName=Plymouth Marine Laboratory
individualName=Jorge Samuel Mendes de Jesus
positionName=Scientific Programmer
role=Administrator
deliveryPoint=Prospect Place,The Hoe
city=Plymouth
postalCode=PL1 3DH
country=uk
electronicMailAddress=rsgweb@pml.ac.uk
providerSite=http://rsg.pml.ac.uk

[server]
maxoperations=30
maxinputparamlength=2048
maxfilesize=300mb
tempPath=/tmp
outputUrl=http://rsg.pml.ac.uk/wps/wpsoutputs
outputPath=/home/data/pywps/wps/wpsoutputs
```

The configuration file is well structured and the most important parameters that require configuration are:

serveraddress, e.g.: <http://rsg.pml.ac.uk/wps/wps.cgi>

outputUrl, e.g.: <http://rsg.pml.ac.uk/wps/wpsoutputs>

outputPath, e.g.: </home/data/pywps/wps/wpsoutputs>

For a complete parameter description please refer to the PyWPS wiki:

http://wiki.rsg.pml.ac.uk/pywps/Configuration_File

3.4.2 Apache Installation

PyWPS may be run as a CGI program on a normal web server. We have used Apache in our testing but nothing prevents the use of other web servers such as LightHTTPD.

To run PyWPS under Apache it is necessary that the folder that contains the WPS wrapper script has execute rights. For example:

```
<Directory "/usr/lib/cgi-bin">
    AllowOverride None
    Options +ExecCGI -MultiViews +FollowSymLinks
    Order allow,deny
    Allow from all
</Directory>
```

Please refer to Apache installation tutorials <http://httpd.apache.org/docs/current/howto/cgi.html> for more information on configuring Apache.

3.4.3 Testing WPS services

Current WPS developments in NETMAR are publicly available on a test server, located at the following URL (Figure 3-1): <http://rsg.pml.ac.uk/wps/index.html>.

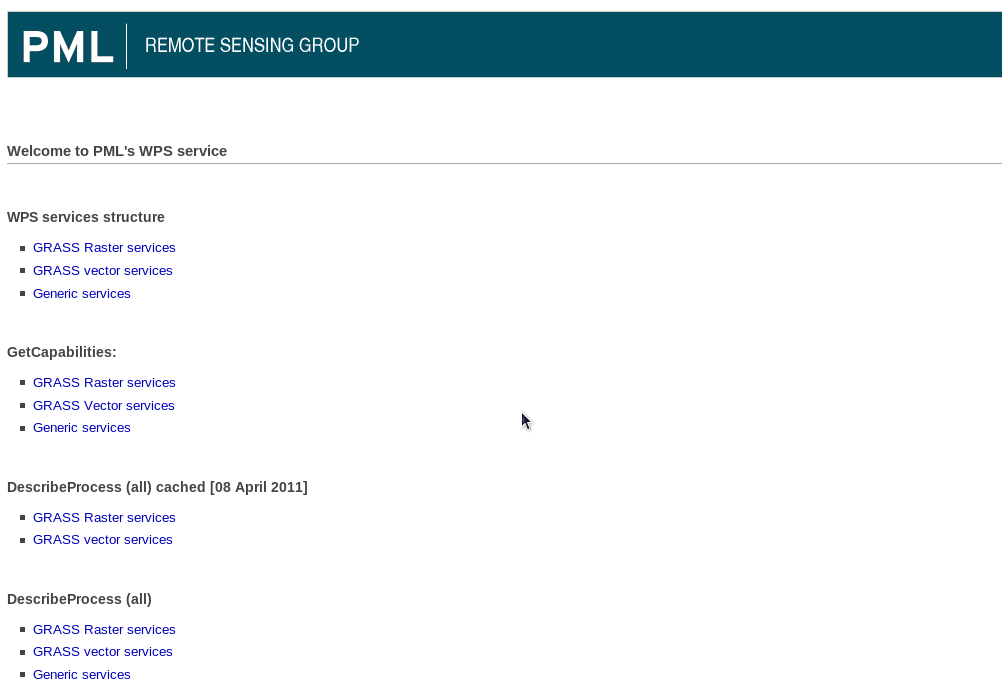


Figure 3-1 WPS service welcome page

The server has 3 WPS instances: Raster, Vector and Generic services.

The WPS instances can be accessed by clicking on the following URLs:

WPS services structure

- GRASS Raster services
- GRASS vector services
- Generic services

Figure 3-2 WPS service instance link

These links don't contain a valid request they simply will point to the following URLs:

<http://rsg.pml.ac.uk/wps/raster.cgi>
<http://rsg.pml.ac.uk/wps/vector.cgi>
<http://rsg.pml.ac.uk/wps/generic.cgi>

These links above may be used as a starting point for further requests. However, to obtain a full WPS XML service description it is advisable to use the GetCapabilities link (Figure 3-3):

GetCapabilities:

- GRASS Raster services
- GRASS Vector services
- Generic services

Figure 3-3 WPS getCapabilities links

For example, the raster GetCapabilities link will generate the following output extract (Figure 3-4):

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```

--<wps:Capabilities service="WPS" version="1.0.0" xml:lang="en-CA" 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 Server</ows:Title>
    <ows:Abstract>PML WPS server</ows:Abstract>
  --<ows:Keywords>
    <ows:Keyword>PML</ows:Keyword>
    <ows:Keyword>NETMAR</ows:Keyword>
    <ows:Keyword>RSG</ows:Keyword>
    <ows:Keyword>Raster</ows:Keyword>
    <ows:Keyword>Image</ows:Keyword>
  </ows:Keywords>
  <ows:ServiceType>WPS</ows:ServiceType>
  <ows:ServiceTypeVersion>1.0.0</ows:ServiceTypeVersion>
  <ows:Fees>None</ows:Fees>
  <ows:AccessConstraints>none</ows:AccessConstraints>
</ows:ServiceIdentification>
--<ows:ServiceProvider>
  <ows:ProviderName>Plymouth Marine Laboratory</ows:ProviderName>
  <ows:ProviderSite xlink:href="http://rsg.pml.ac.uk"/>
  --<ows:ServiceContact>
    <ows:IndividualName>Jorge Samuel Mendes de Jesus</ows:IndividualName>
    <ows:PositionName>Scientific Programmer</ows:PositionName>
  --<ows:ContactInfo>
    --<ows:Address>
      <ows:DeliveryPoint>Prospect Place,The Hoe</ows:DeliveryPoint>
      <ows:City>Plymouth</ows:City>
    
```

Figure 3-4 GetCapabilities output example extract

The web site also provides links to the WPS WSDL content description (cached and dynamic versions), and a link that contains test data (Figure 3-5).

WSDL cached [08 April 2011]

- [GRASS Raster services](#)
- [GRASS Vector services](#)
- [GRASS Generic services](#)

WSDL

- [GRASS Raster services](#)
- [GRASS vector services](#)
- [Other web services](#)

Testdata

- [Test Data](#)

Figure 3-5 WSDL and Testdata content of WPS support web site

The WSDL links point to a WSDL XML description file that can be used by generic web service orchestration structures.

WSDL file generation can be time consuming and it is advisable to use already cached files:

<http://rsg.pml.ac.uk/wps/raster.wsdl>
<http://rsg.pml.ac.uk/wps/vector.wsdl>
<http://rsg.pml.ac.uk/wps/generic.wsdl>

WPS 1.0.0 defines that a WPS service shall return a WSDL document when using the following KVP structure, this sort of request is dynamic and not cached:

<http://rsg.pml.ac.uk/wps/raster.cgi?WSDL>
<http://rsg.pml.ac.uk/wps/vector.cgi?WSDL>
<http://rsg.pml.ac.uk/wps/generic.cgi?WSDL>

4 Development summary

4.1 WSDL/SOAP development

WSDL is a generic web service description language using XML, it provides a generic web service description that should be understood by generic web service orchestration structures. The WPS 1.0.0 standard, defines that a getCapabilities shall include a link to a WSDL XML document describing the WPS instance and services. Normally, a WPS instance does not provide a valid WSDL document, as that part of the specification is seldom implemented.

The WSDL document defines what sort of transport protocol shall be used to pass information; SOAP is the most commonly used and suggested in WPS 1.0.0 documentation [WPS07].

Despite the inclusion of WSDL/SOAP support in WPS, some technical aspects are not clear. The examples given in WPS 1.0.0 annex D and E are not so clear, especially for SOAP support. Therefore a significant effort was allocated to develop a WSDL/SOAP implementation that would properly integrate the WPS standard into WSDL/SOAP. For example, special considerations were taken to port WPS exceptions and async capabilities into WSDL/SOAP. Please see:

<http://wiki.rsg.pml.ac.uk/pywps/Orchestration>

The development followed a transparent approach, meaning that at the time a standard WPS process is generated in a getCapabilities/describeProcess XML response document it will also be integrated in the WSDL document and accessible using SOAP. This was possible by combining XML transformations (XSLT) and direct access to the PyWPS class structures.

4.2 Test Driven Development (TDD)

PyWPS developments followed a Test Driven Development (TDD) (http://en.wikipedia.org/wiki/Test-driven_development), where each new function has a corresponding test unit that checks for proper functionality. Development using TDD is initially slower, since it is necessary to program extra testing code, but code developed using TDD is more robust and less likely to have bugs and need refactoring during alpha and beta code release.

PyWPS-3.2-soap and the code contained in the SVN-trunk was submitted to extensive unit tests to check for XML schema compliance and WPS 1.0.0 compliance. The current code (8/April/2011) is completely compliant with WPS1.0.0 schemas.

TDD usage in PyWPS development identified two WPS 1.0.0 implementation problems that are currently being addressed by the OGC community:

- Incorrect Bounding Box (BBOX) parsing (a OGC change request is under consideration)
- xlink reference namespace (<http://lists.opengeospatial.org/pipermail/wps-dev/2011-February/000094.html>)

4.3 WPS-GRASS-Bridge

PyWPS is a WPS framework API, meaning, it provides WPS support for code that will be developed by users, and therefore it is not PyWPS's intention to provide WPS processes. Nevertheless, PyWPS provides generic GRASS GIS integration.

The WPS-GRASS-Bridge (<http://code.google.com/p/wps-grass-bridge/>) is a project that automatically generates WPS processes from GRASS GIS modules. The current development GRASS GIS 7.0 version provides a WPS XML description of all its modules; therefore it is possible to convert the XML description into Python code that integrates the module's input/outputs directly into PyWPS. Part of the PyWPS development focused on API refactoring facilitating the development needs of WPS-GRASS-Bridge and supporting the debugging processes.

The PML WPS instance runs WPS-GRASS-Bridge, providing 83 raster and 57 vector processes. For complete install information see: http://code.google.com/p/wps-grass-bridge/wiki/PyWPS_Integration

5 Wiki

PML hosts the current PyWPS wiki, where new development is documented and explained (http://wiki.rsg.pml.ac.uk/pywps/Main_Page), Figure 5-1, the wiki is a reference point for the community and developers, and contains extra information concerning PyWPS-3.2-SOAP implementation.

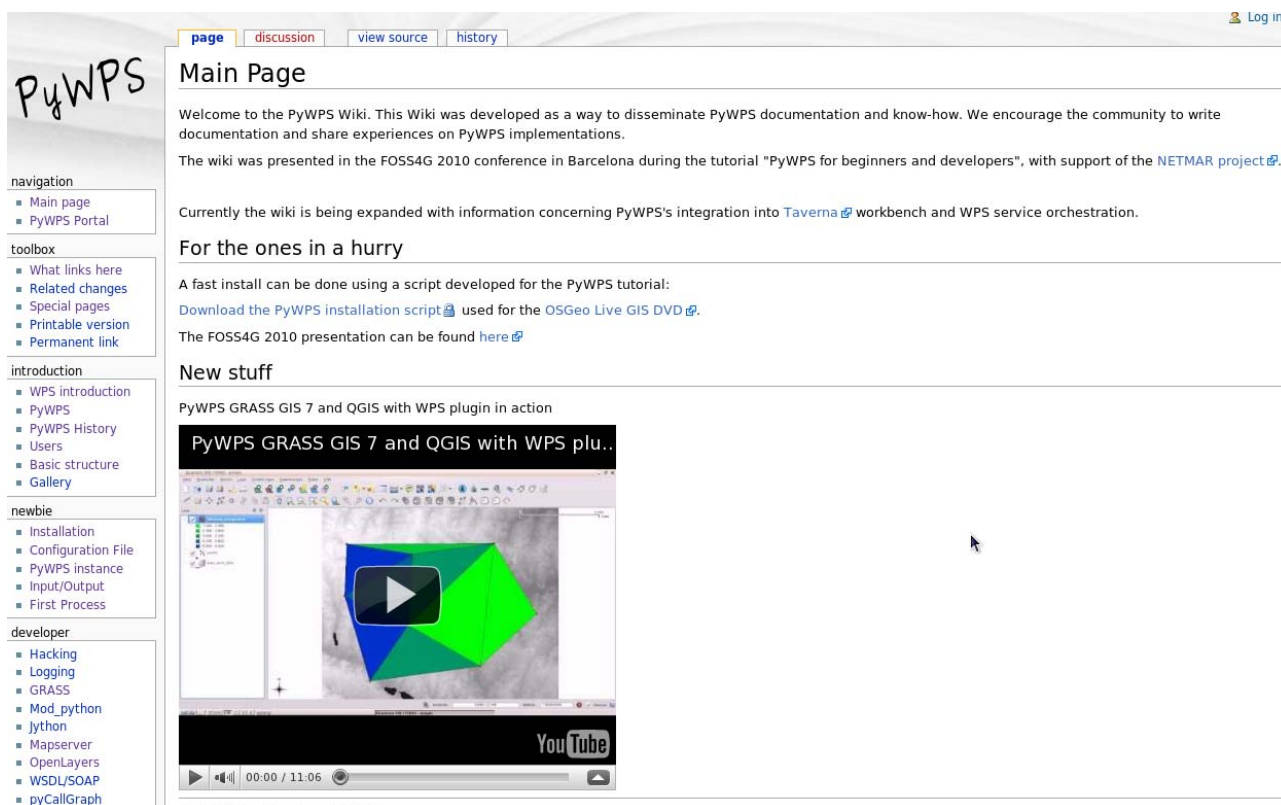


Figure 5-1 PyWPS wiki (<http://wiki.rsg.pml.ac.uk/pywps>) supported by NETMAR project and hosted on PML's servers

6 Taverna workbench

Taverna (<http://www.taverna.org.uk>) workbench is an open-source tool designed for composition and enactment of bioinformatics workflows, nevertheless it is generic enough to be used for web services from other scientific fields (like geoinformatics) [TAV06]. Taverna workflow is a linked graph of processors that are processes or executable components which accept input data, process it and create an output. Processes can be of WSDL, SOAP or RESTful nature while executable components are created using the beanshell script language; R language, using Rserver; and Xpath expressions. Each process node consumes data that arrives at its input ports and produces data on its output ports; data dependences are created by linking output ports (sources) and input ports (sinks) of different processes. A similar approach will be used for the Service Chaining Editor component of EUMIS.

Taverna uses SCUFL (<http://www.taverna.org.uk/developers/taverna-1-7-x/architecture/scufl>, Simple Conceptual Unified Flow Language) to describe workflow construction and interaction between processors, and how data flows through it (data flow controllers). This workflow language is more data oriented compared to BPEL (which is more process oriented). For example, Taverna/SCUFL will run processes as soon as possible, based on data availability, therefore, processes will be called in parallel as default, while BPEL would require an explicit definition of the control flow determining the order of execution of the processes. Future EUMIS developments will address the use of SCUFL2.0 as generic orchestration language since Taverna can be run as a server that accepts a SCUFL2.0 document describing a workflow structure.

The PyWPS-3.2-SOAP implementation was also designed to properly integrate with Taverna. This addressed issues such as SOAP/WSDL bugs, exception handling, proper data type transfer between web services etc.

A demo video can be seen at <http://www.youtube.com/watch?v=JNAtoOejVlo>.

Figure 6-1 shows a Taverna workflow example using 2 WPS services: reducer and histogram process. In the work flow an initial image (http://rsg.pml.ac.uk/wps/testdata/basin_50K_nc.tif) is gathered and its size reduced to 20% (the reduction factor is between 0.0 and 1.0 as defined in the WPS service) using the WPS reducer process, in parallel another WPS service (histogramprocess) is used to generate a generic image histogram output. The WPS used are available in PML's generic WPS instance.

Workflow examples have been uploaded to the myExperiment platform and are publically available:

<http://www.myexperiment.org/workflows/2089.html>

<http://www.myexperiment.org/workflows/2066.html>

<http://www.myexperiment.org/workflows/1928.html>

All the created workflows have tags that relate them to the NETMAR project.

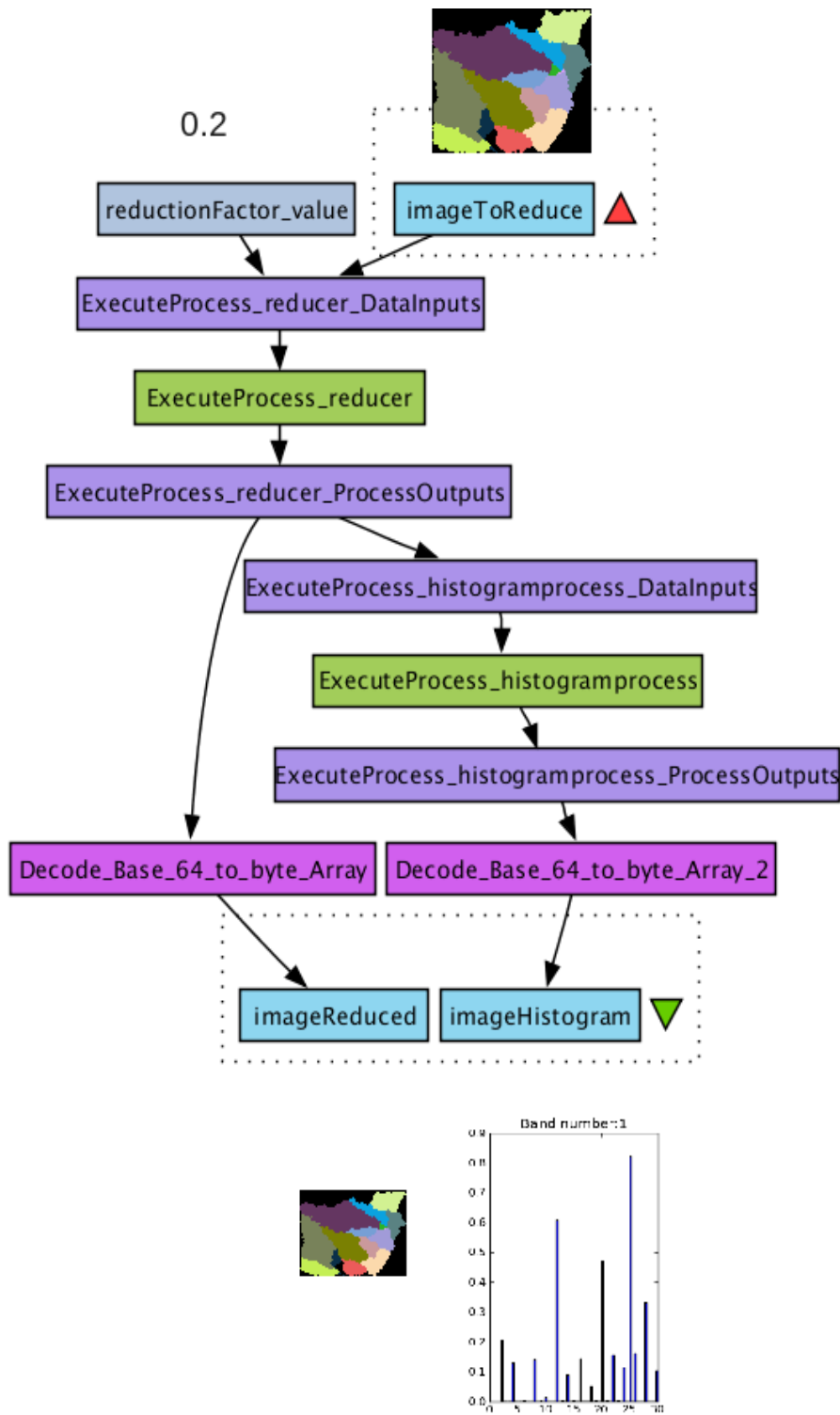


Figure 6-1 Workflow example using WPS services (green boxes), whose input/output is defined in the purple boxes, fuchsia boxes are Taverna workbench services that convert base64 coded data into binary. Blue polygons represent inputs and outputs.

7 References

[SOAP00] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H., Thatte, S., Winer, D., 2000. Simple object access protocol (SOAP) 1.1, W3C Note 08 May 2000. W3C Note. The World Wide Web Consortium (W3C).

[GRASS06] Cepicky, J., 2006. GRASS goes web: PyWPS, in: Free and Open Source Software for Geoinformatics, Lausanne, Switzerland.

[WSDL01] Christensen, E., Curbera, F., Meredith, G., Weerawarana, S., 2001. Web services description language (WSDL) 1.1, W3C Note 15 March 2001. W3C Note. The World Wide Web Consortium (W3C).

[TAV06] Hull, D., Wolstencroft, K., Stevens, R., Goble, C., Pocock, M., Li, P., Oinn, T., 2006. 29 Taverna: a tool for building and running workflows of services. *Nucleic Acids Research* 34, 729–732.

[WPS07] Schut, P., 2007. OpenGIS Web Processing Service 1.0.0. OpenGIS standard 05-007r7. Open Geospatial Consortium Inc.