Пример оформления кода в виде WPS сервиса.

Правила чтения данного руководства.

Замечание! Данное руководство актуально для версии PyWPS 4.2.4.
Комментарии обозначаются знаком #.
Важные элементы кода будут выделены **жирным шрифтом**.

Примеры основаны на документации для PyWPS и микрофреймворка Flask.

Рассмотрим пример из документации.
Данный сервис принимает на вход <имя> и возвращает фразу ‘Hello <имя>’

```
from pywps import Process, LiteralInput, LiteralOutput, UOM

class SayHello(Process):
  def __init__(self):
    inputs = [LiteralInput('name', 'Input name', data_type='string')]
    outputs = [LiteralOutput('response', 'Output response', data_type='string')]

    super(SayHello, self).__init__(
      self._handler,
      identifier='say_hello',
      title='Process Say Hello',
      abstract='Returns a literal string output with Hello plus the inputed name',
      version='1.3.3.7',
      inputs=inputs,
      outputs=outputs,
      store_supported=True,
      status_supported=True
    )

  def _handler(self, request, response):
    response.outputs['response'].data = 'Hello ' + request.inputs['name'][0].data
    response.outputs['response'].uom = UOM('unity')
    return response
```

Оформим код в виде сервиса.

Создадим файл с названием **wps_tutorial.py**

```
# Для работы со строками понадобится импортировать базовый класс Process и классы для работы со строковыми данными LiteralInput, LiteralOutput
from pywps import Process, LiteralInput, LiteralOutput

class Tutorial(Process): # Указать уникальное имя класса
  def __init__(self):
    # Описание входных и выходных параметров оставим без изменений за исключением имени входного параметра, вместо name используем data
    inputs = [LiteralInput('data', 'Input data', data_type='string')]
    outputs = [LiteralOutput('response', 'Output response', data_type='string')]

    super(Tutorial, self).__init__( # Подставляем имя своего класса
      self._handler,
      identifier='wps_tutorial', # В качестве идентификатора указываем имя файла без расширения
      title='Tutorial Process', # Изменить название сервиса
```

```
            abstract='Human readable defenition', # Добавить описание
            version='1.3.3.7',
            inputs=inputs,
            outputs=outputs,
            store_supported=True,
            status_supported=True
        )

    def _handler(self, request, response):
        # В данной части можем приступить к написанию кода, например, подсчитаем факториал числа n
        n = int(request.inputs['data'][0].data) # По имени входного параметра data получаем данные
        factorial = 1
        while n > 1:
            factorial *= n
            n -= 1
        # Модифицируем данные для ответа
        response.outputs['response'].data = 'Factorial ' + request.inputs['data'][0].data + 'is ' + str(factorial)
        return response
```

Рассмотрим передачу файлов в качестве параметра и вызов стороннего приложения для обработки данных.

```
import pywps
import requests
import subprocess
from pywps import Process, LiteralInput, LiteralOutput, UOM, ComplexInput, ComplexOutput, get_format, Format # Модули для работы с файлами ComplexInput, ComplexOutput, get_format, Format

class Reproject_Raster(Process):

    def __init__(self):
        # В качестве параметров передадим файл и проекцию в виде строки
        inputs = [ComplexOutput('source', 'Input source', [Format('GEOTIFF')]), LiteralInput('proj', 'Input proj', data_type='string')]

        # Выходной файл можно вернуть в виде файла либо в виде ссылки и скачать позже, например, если процесс обработки занимает продолжительное время
        # По умолчанию сервис вернет файл
        outputs = [ComplexOutput('dest', 'Output dest', [Format('GEOTIFF')] , None, '', [], None, None, False, 0, None] # Возврат файла

        # Для возврата ссылки, установить параметр as_reference = True, в данном примере это 9-й параметр (может отличаться в зависимости от версии, см. документацию)
        outputs = [ComplexOutput('dest', 'Output dest', [Format('GEOTIFF')] ,None, '', [], None, None, True, 0, None] # Возврат ссылки

        super(Reproject_Raster, self).__init__(
            self._handler,
            identifier='wps_reproject_raster',
            title='',
            abstract='',
            inputs=inputs,
            outputs=outputs,
            store_supported=True,
            status_supported=True
        )

    def _handler(self, request, response):

        source_file = request.inputs['source'][0].data # Считываем входной файл
        proj = request.inputs['proj'][0].data # Считываем целевую проекцию
```

```
        dest_file = source_file # Задаем имя выходного файла

        subprocess.run(['gdal_warp', '-t_srs', proj, source_file, dest_file]) # Вызываем утилиту gdal_warp

        # Return result
        response.outputs['dest'].output_format = Format('GEOTIFF') # Устанавливаем выходной формат файла
        response.outputs['dest'].file = dest_file # В зависимости от значения параметра as_reference будет возвращен файл либо ссылка на него

        return response
```

Замечание! Для передачи и возврата нескольких фалов необходимо описать каждый из них отдельно.
Замечание! Так как форматы файлов PRJ, SHX, BDF не описаны как поддерживаемые, то везде указываем SHP.

```
        inputs = [ ComplexInput('shp','i_shp',[Format('SHP')]), ComplexInput('prj','i_prj',[Format('SHP')]), ComplexInput('shx','i_shx',[Format('SHP')]), ComplexInput('shx','i_shx',[Format('SHP')]), ComplexInput('dbf','i_dbf',[Format('SHP')]) ]
        outputs = [ ComplexOutput('shp', 'o_shp', [Format('SHP')]), ComplexOutput('prj', 'o_prj', [Format('SHP')]), ComplexOutput('shx', 'o_shx', [Format('SHP')]), ComplexOutput('dbf', 'o_dbf', [Format('SHP')]) ]

        # Секция считыавния файлов
        inp_shp = request.inputs['shp'][0].file
        inp_prj = request.inputs['prj'][0].file
        inp_shx = request.inputs['shx'][0].file
        inp_dbf = request.inputs['dbf'][0].file

        # Секция возврата файлов
        response.outputs['shp'].output_format = Format('SHP')
        response.outputs['shp'].file = out_shp
        response.outputs['prj'].output_format = Format('SHP')
        response.outputs['prj'].file = out_prj
        response.outputs['shx'].output_format = Format('SHP')
        response.outputs['shx'].file = out_shx
        response.outputs['dbf'].output_format = Format('SHP')
        response.outputs['dbf'].file = out_dbf
```

Передача и возврат множества файлов, для случая, когда заранее не известно их количество.

```
import os
import pywps
import requests
import subprocess

from pywps import Process, LiteralInput, LiteralOutput, UOM, ComplexInput, ComplexOutput, get_format, Format, FORMATS
from pywps.wpsserver import temp_dir
from pywps.inout.outputs import MetaLink4, MetaFile # Импортируем классы для создания выходного xml файла


class Multiple(Process):

    def __init__(self):
        # Указать диапазон повторов, для соответствующего параметра установив значения min_occurs=0, max_occurs=4
        inputs = [ ComplexInput('input','Input img',[Format('GEOTIFF')], min_occurs=0, max_occurs=4), LiteralInput('new_prj', 'Input new projection', data_type='string') ]
        # Описание выходного xml файла
        outputs = [ ComplexOutput('output', 'Metalink4 output', abstract='A metalink file storing URIs to multiple files', as_reference=True, supported_formats=[FORMATS.META4]) ]

        super(Multiple, self).__init__(
            self._handler,
```

```
            identifier='wps_multiple',
            title='',
            abstract='',
            inputs=inputs,
            outputs=outputs,
            store_supported=True,
            status_supported=True
        )

    def _handler(self, request, response):

        # Создаем список поступивших на вход файлов
        path_inputs = []

        # Заполняем его
        for i in range(len(request.inputs['input'])):
            path_inputs.append(os.path.abspath(request.inputs['input'][i].file))

        # Считываем целевую проекцию
        proj = request.inputs['new_prj'][0].data

        # Обрабатываем поступившие файлы
        for i in path_inputs:
            subprocess.run(['gdal_warp', '-t_srs', proj, i, i])

        # Создаем результирующий файл
        ml = MetaLink4('test-ml-1', 'MetaLink with links to text files.', workdir=self.workdir)

        # Наполняем его данными
        for i in path_inputs:
            # Для каждого файла в списке создаем ссылку
            mf = MetaFile('output_{}'.format(i), 'Test output', fmt=FORMATS.GEOTIFF) # Устанавливаем формат
            mf.file = i # Указываем путь к файлу
            ml.append(mf) # Добавляем эту информацию в xml

        response.outputs['output'].data = ml.xml # Возвращаем xml файл, содержащий список ссылок

        return response
```