pyww3 Documentation

Release 0.1.4

Caio Stringari

Apr 02, 2022

CONTENTS:

CHAPTER

ONE

PYWW3

Python wrapper for NOAA's WaveWatchIII (WW3) Model.

This package wraps around the WW3's executables by properly defining the namelists (`.nml`) required to drive the model's executables.

1.1 Requirements

`pyww3` requires `WaveWatchIII` to be properly compiled with `netCDF4` available in your `$PATH`. Please follow the installation instructions from NOAA.

Programs supported: `ww3_grid`, `ww3_prnc`, `ww3_shel`, `ww3_ounf`, `ww3_ounp` and `ww3_bounc`.

Note that I don't have plans to support programs that require ASCII input (such as `ww3_outf`) even tough they may have an associated namelist.

You will need `python 3.7+` because of the extensive usage of `dataclasses`.

1.2 Getting Started

All the implemented classes have the same structure and methods. For example, to run simulation with `ww3_shel` you do:

```
import datetime
from pyww3.shel import WW3Shel
W = WW3Shel(nproc=8,
            runpath="tests/test_run/",
            mod_def="tests/test_data/GLOB_60_MIN.ww3grid",
            domain_start=datetime.datetime(2010, 1, 1, 0),
            domain_stop=datetime.datetime(2010, 1, 1, 2),
            input_forcing_winds=True,
            input_forcing_ice_conc=True,
            date_field_stride=3600,
            date_point_stride=3600,
            date_restart_stride=3600,
            type_point_file="tests/test_data/boundary_point_list.txt")
```

(continues on next page)

(continued from previous page)

```
W.to_file() # writes ww3_shel.nml in the run path
W.run()  # run the simulation
print(W.stdout) # print the output given by WW3
```

1.3 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

WW3 is maintained and distributed by NOAA's Environmental Modeling Center (EMC).

1.4 Disclaimer

There is no warranty for the program, to the extent permitted by applicable law except when otherwise stated in writing the copyright holders and/or other parties provide the program "as is" without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. the entire risk as to the quality and performance of the program is with you. should the program prove defective, you assume the cost of all necessary servicing, repair or correction.

CHAPTER

TWO

INSTALLATION

2.1 Stable release

To install pyww3, run this command in your terminal:

```
$ pip install pyww3
```

This is the preferred method to install pyww3, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for pyww3 can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/caiostringari/pyww3
```

Or download the tarball:

```
$ curl -OJL https://github.com/caiostringari/pyww3/tarball/master
```

Once you have a copy of the source, you can install it with:

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

CHAPTER

THREE

USAGE

The main goal of pyww3 is programmatically to provide access to the parameters that the user has to manually input the WW3 namelist files.

For example, if a namelist defines the following:

```
&SPECTRUM_NML
  SPECTRUM%XFR            =  1.1
  SPECTRUM%FREQ1          =  0.04118
  SPECTRUM%NK             =  32
  SPECTRUM%NTH            =  24
/
```

You are able to write the same in python as:

```
spectrum_xfr: int = 1.1
spectrum_freq1: int = 0.04118
spectrum_nk: int = 32
spectrum_thoff: int = 0
```

All pyww3 classes use type hinting and all parameters are converted to the strings that WW3 is expecting. Data validation is done in the `__post_init__` method of each class. The checks include searching for required files and, for some parameters, checking if the inputs are compatible to the namelist definitions.

All classes inherit from `WW3Base`. This class implements the methods `to_file()` which writes the namelist text to file, `run()` which runs a given program and `update_text()` which can be used to manipulate the namelist text ad-hoc. Each class implements the method `populate_namelist()` that generates the namelist text.

3.1 WW3Grid

Program used to create grids. Only regular spherical grids and unstructured meshes were tested.

Example:

```
from pyww3.grid import WW3Grid

# dictionary with grid information. This is normally read from a JSON file.
grid_info = {"grid_nml": "GLOB_60_MIN.nml",
             "grid_name": "GLOB_60M",
             "grid_type":"RECT",
             "grid_coord":"SPHE",
             "grid_clos": "NONE",
```

(continues on next page)

(continued from previous page)

```
              "rect_nx": 360,
              "rect_ny": 157,
              "rect_sx": 1.0,
              "rect_sy": 1.0,
              "rect_sf": 1.0,
              "rect_x0": -180.0,
              "rect_y0": -78.0,
              "rect_sf0" :1.0,
              "grid_zlim": -0.1,
              "grid_dmin": 2.5,
              "depth_filename": "GLOB_60_MIN.depth_ascii",
              "depth_sf": 0.0010,
              "obst_filename": "GLOB_60_MIN.obstr_lev1",
              "obst_sf": 0.010,
              "mask_filename": "GLOB_60_MIN.maskorig_ascii"}

# create the instance
W = WW3GRid(runpath="some/valid/path/",
            grid_name=grid_info["grid_type"],
            grid_nml=grid_info["grid_nml"],
            grid_type=grid_info["grid_type"],
            grid_coord=grid_info["grid_coord"],
            grid_clos=grid_info["grid_clos"],
            rect_nx=grid_info["rect_nx"],
            rect_ny=grid_info["rect_ny"],
            rect_sx=grid_info["rect_sx"],
            rect_sy=grid_info["rect_sy"],
            rect_sf=grid_info["rect_sf"],
            rect_x0=grid_info["rect_x0"],
            rect_y0=grid_info["rect_y0"],
            rect_sf0=grid_info["rect_sf0"],
            grid_zlim=grid_info["grid_zlim"],
            grid_dmin=grid_info["grid_dmin"],
            depth_filename=os.path.join(datapath, grid_info["depth_filename"]),
            depth_sf=grid_info["depth_sf"],
            obst_filename=os.path.join(datapath, grid_info["obst_filename"]),
            obst_sf=grid_info["obst_sf"],
            mask_filename=os.path.join(datapath, grid_info["mask_filename"]))

# update namelist blocks that could cause errors.
W.update_text("&SED_NML", action="remove")
W.update_text("&SLOPE_NML", action="remove")
W.update_text("&CURV_NML", action="remove")
W.update_text("&UNST_NML", action="remove")

# Update the timesteps. Don't forget to update the namelist text!
W.timesteps_dtmax = 480.0 * 3
W.timesteps_dtxy = 160.0 * 3
W.timesteps_dtkth = 240.0 * 3
W.timesteps_dtmin = 10.0 * 3
W.text = W.populate_namelist()
```

(continues on next page)

(continued from previous page)

```
# write ww3_grid.namelist in the `runpath`.
W.to_file()

# run `ww3_grid` in the `runpath`.
W.run()
```

3.2 WW3Bounc

Program used to post-process spectral netcdf files generated with `ww3_ounp`. Usually used to created boundary conditions to a nested simulation.

Example:

```
from pyww3.bounc import WW3Bounc

# create the instance
W = WW3Bounc(runpath="some/valid/path/",
             mod_def="mod_def.ww3",
             bound_file="somefile.nc")

# write ww3_bounc.namelist in the `runpath`.
W.to_file()

# run `ww3_bounc` in the `runpath`.
W.run()
```

CHAPTER

FOUR

HISTORY

4.1 0.1.4 (2021-09-01)

- First release on PyPI.

4.2 0.1.0 (2021-08-25)

- First more or less usable version.

CHAPTER

FIVE

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/caiostringari/pyww3/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

pyww3 could always use more documentation, whether as part of the official pyww3 docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/caiostringari/pyww3/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *pyww3* for local development.

1. Fork the *pyww3* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/pyww3.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv pyww3
$ cd pyww3/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 pyww3 tests
$ python setup.py test or pytest
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.5, 3.6, 3.7 and 3.8, and for PyPy. Check https://travis-ci.com/caiostringari/pyww3/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ pytest tests.test_pyww3
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER

SIX

API

6.1 WW3Base

class pyww3.ww3.**WW3Base**

Define the base class WW3(). Everything else enhirits from this class.

run(*mpi=False*, *nproc=2*)

Run a program using mpi or not.

to_file()

Write namelist text to file **ww3**_?.nml.

update_text(*block: str*, *action: str = 'add'*, *index: int = - 1*)

Update namelist block in the text with an action.

6.2 WW3Bounc

6.3 WW3Grid

class pyww3.grid.**WW3GRid**(*runpath: str*, *grid_name: str*, *grid_nml: str*, *grid_type: str*, *grid_coord: str*, *grid_clos: str*, *output: str = 'ww3_grid.nml'*, *spectrum_xfr: int = 1.1*, *spectrum_freq1: int = 0.04118*, *spectrum_nk: int = 32*, *spectrum_nth: int = 24*, *spectrum_thoff: int = 0*, *run_fldry: bool = False*, *run_flcx: bool = True*, *run_flcy: bool = True*, *run_flcth: bool = True*, *run_flck: bool = True*, *run_flsou: bool = True*, *timesteps_dtmax: float = 480.0*, *timesteps_dtxy: float = 160.0*, *timesteps_dtkth: float = 240.0*, *timesteps_dtmin: float = 10.0*, *grid_zlim: float = 0.0*, *grid_dmin: float = 0.0*, *rect_nx: int = 0*, *rect_ny: int = 0*, *rect_sx: float = 0.0*, *rect_sy: float = 0.0*, *rect_sf: float = 1.0*, *rect_x0: float = 0.0*, *rect_y0: float = 0.0*, *rect_sf0: float = 1.0*, *curv_nx: int = 0*, *curv_ny: int = 0*, *curv_xcoord_sf: float = 1.0*, *curv_xcoord_off: float = 0.0*, *curv_xcoord_filename: str = ''*, *curv_xcoord_idf: int = 21*, *curv_xcoord_idla: int = 1*, *curv_xcoord_idfm: int = 1*, *curv_xcoord_format: str = '(....)'*, *curv_ycoord_sf: float = 1.0*, *curv_ycoord_off: float = 0.0*, *curv_ycoord_filename: str = ''*, *curv_ycoord_idf: int = 22*, *curv_ycoord_idla: int = 1*, *curv_ycoord_idfm: int = 1*, *curv_ycoord_format: str = '(....)'*, *unst_sf: float = 1.0*, *unst_filename: str = ''*, *unst_idla: int = 20*, *unst_idfm: int = 1*, *unst_format: int = '(20f10.2)'*, *unst_ugobcfile: str = ''*, *depth_sf: float = 1.0*, *depth_filename: str = ''*, *depth_idf: int = 50*, *depth_dla: int = 1*, *depth_dfm: int = 1*, *depth_format: str = '(....)'*, *mask_filename: str = ''*, *mask_idf: int = 60*, *mask_idla: int = 1*, *mask_idfm: int = 1*, *mask_format: str = '(....)'*, *obst_sf: float = 1.0*, *obst_filename: str = ''*, *obst_idf: int = 70*, *obst_idla: int = 1*, *obst_idfm: int = 1*, *obst_format: str = '(....)'*, *slope_sf: float = 1.0*, *slope_filename: str = ''*, *slope_idf: int = 80*, *slope_idla: int = 1*, *slope_idfm: int = 1*, *slope_format: str = '(....)'*, *sed_sf: float = 1.0*, *sed_filename: str = ''*, *sed_idfm: int = 1*, *sed_idla: int = 1*, *sed_format: str = '(....)'*)

This class abstracts the program ww3_grid. It is an extension of the class *pyww3.ww3.WW3Base()*.

populate_namelist()

Create the namelist text using NOAA's latest template.

6.4 WW3Ounf

class pyww3.ounp.**WW3Ounp**(*runpath: str*, *mod_def: str*, *output: str = 'ww3_ounp.nml'*, *ww3_pnt: str = 'out_pnt.ww3'*, *point_timestart: datetime.datetime = datetime.datetime(1900, 1, 1, 0, 0)*, *point_timestride: int = 0*, *point_timecount: int = 1000000000*, *point_timesplit: int = 6*, *point_list: str = 'all'*, *point_samefile: bool = True*, *point_buffer: int = 150*, *point_type: int = 1*, *file_prefix: str = 'ww3.'*, *file_netcdf: int = 4*, *spectra_output: int = 3*, *spectra_scale_fac: int = 1*, *spectra_output_fac: int = 0*, *param_output: int = 4*, *source_output: int = 4*, *source_scale_fac: int = 0*, *source_output_fac: int = 0*, *source_table_fac: int = 0*, *source_spectrum: bool = True*, *source_input: bool = True*, *source_interactions: bool = True*, *source_dissipation: bool = True*, *source_bottom: bool = True*, *source_ice: bool = True*, *source_total: bool = True*)

This class abstracts the program ww3_ounf. It is an extension of the class *pyww3.ww3.WW3Base()*.

populate_namelist()

Create the namelist text using NOAA's latest template.

6.5 WW3Ounp

class pyww3.ounf.**WW3Ounf**(*runpath: str*, *mod_def: str*, *ww3_grd: str = 'out_grd.ww3'*, *output: str = 'ww3_ounf.nml'*, *field_timestart: datetime.datetime = datetime.datetime(1900, 1, 1, 0, 0)*, *field_timestride: int = 0*, *field_timecount: int = 1000000000*, *field_timesplit: int = 6*, *field_list: typing.List[str] = <factory>*, *field_samefile: bool = True*, *field_partition: typing.List[int] = <factory>*, *field_type: int = 3*, *file_prefix: str = 'ww3.'*, *file_netcdf: int = 4*, *file_ix0: int = 1*, *file_ixn: int = 1000000000*, *file_iy0: int = 1*, *file_iyn: int = 1000000000*, *smc_type: int = 1*, *smc_sxo: int = 0*, *smc_exo: int = 0*, *smc_syo: int = 0*, *smc_eyo: int = 0*, *smc_celfac: int = 1*, *smc_noval: int = -999*)

This class abstracts the program ww3_ounf. It is an extension of the class *pyww3.ww3.WW3Base()*.

populate_namelist()

Create the namelist text using NOAA's latest template.

6.6 WW3Prnc

class pyww3.prnc.**WW3Prnc**(*runpath: str*, *mod_def: str*, *forcing_field: str*, *forcing_grid_latlon: bool*, *file_filename: str*, *file_longitude: str*, *file_latitude: str*, *file_var_1: str*, *output: str = 'ww3_prnc.nml'*, *file_var_2: str = ' '*, *file_var_3: str = ' '*, *forcing_timestart: datetime.datetime = datetime.datetime(1900, 1, 1, 0, 0)*, *forcing_timestop: datetime.datetime = datetime.datetime(2900, 12, 31, 0, 0)*, *file_timeshift: str = '00000000 000000'*)

This class abstracts the program ww3_prnc. It is an extension of the class *pyww3.ww3.WW3Base()*.

populate_namelist()

Create the namelist text using NOAA's latest template.

reverse_latitudes(*newfilename*)

Reverse latitudes if requested

6.7 WW3Shel

Abstracts the ww3_shel program.

class pyww3.shel.**WW3Shel**(*runpath: str*, *mod_def: str*, *output: str = 'ww3_shel.nml'*, *nproc: int = 1*, *domain_iostyp: int = 1*, *domain_start: datetime.datetime = datetime.datetime(1900, 1, 1, 0, 0)*, *domain_stop: datetime.datetime = datetime.datetime(2900, 12, 31, 0, 0)*, *input_forcing_water_levels: bool = False*, *input_forcing_currents: bool = False*, *input_forcing_winds: bool = False*, *input_forcing_ice_conc: bool = False*, *input_forcing_ice_param1: bool = False*, *input_forcing_ice_param2: bool = False*, *input_forcing_ice_param3: bool = False*, *input_forcing_ice_param4: bool = False*, *input_forcing_ice_param5: bool = False*, *input_forcing_mud_density: bool = False*, *input_forcing_mud_thickness: bool = False*, *input_forcing_mud_viscosity: bool = False*, *input_assim_mean: bool = False*, *input_assim_spec1d: bool = False*, *input_assim_spec2d: bool = False*, *type_field_list: typing.List[str] = <factory>*, *type_point_file: str = 'mylist'*, *type_track_format: bool = True*, *type_partition_x0: int = 0*, *type_partition_xn: int = 0*, *type_partition_nx: int = 0*, *type_partition_y0: int = 0*, *type_partition_yn: int = 0*, *type_partition_ny: int = 0*, *type_coupling_sent: str = ' '*, *type_coupling_received: str = ' '*, *date_field_start: datetime.datetime = datetime.datetime(1900, 1, 1, 0, 0)*, *date_field_stride: int = 0*, *date_field_stop: datetime.datetime = datetime.datetime(2900, 12, 31, 0, 0)*, *date_point_start: datetime.datetime = datetime.datetime(1900, 1, 1, 0, 0)*, *date_point_stride: int = 0*, *date_point_stop: datetime.datetime = datetime.datetime(2900, 12, 31, 0, 0)*, *date_track_start: datetime.datetime = datetime.datetime(1900, 1, 1, 0, 0)*, *date_track_stride: int = 0*, *date_track_stop: datetime.datetime = datetime.datetime(2900, 12, 31, 0, 0)*, *date_restart_start: datetime.datetime = datetime.datetime(1900, 1, 1, 0, 0)*, *date_restart_stride: int = 0*, *date_restart_stop: datetime.datetime = datetime.datetime(2900, 12, 31, 0, 0)*, *date_boundary_start: datetime.datetime = datetime.datetime(1900, 1, 1, 0, 0)*, *date_boundary_stride: int = 0*, *date_boundary_stop: datetime.datetime = datetime.datetime(2900, 12, 31, 0, 0)*, *date_partition_start: datetime.datetime = datetime.datetime(1900, 1, 1, 0, 0)*, *date_partition_stride: int = 0*, *date_partition_stop: datetime.datetime = datetime.datetime(2900, 12, 31, 0, 0)*, *date_coupling_start: datetime.datetime = datetime.datetime(1900, 1, 1, 0, 0)*, *date_coupling_stride: int = 0*, *date_coupling_stop: datetime.datetime = datetime.datetime(2900, 12, 31, 0, 0)*, *date_restart: str = "'19000101 000000' '0' '29001231 000000'"*, *homog_count_n_ic1: int = 0*, *homog_count_n_ic2: int = 0*, *homog_count_n_ic3: int = 0*, *homog_count_n_ic4: int = 0*, *homog_count_n_ic5: int = 0*, *homog_count_n_mdn: int = 0*, *homog_count_n_mth: int = 0*, *homog_count_n_mvs: int = 0*, *homog_count_n_lev: int = 0*, *homog_count_n_cur: int = 0*, *homog_count_n_wnd: int = 0*, *homog_count_n_ice: int = 0*, *homog_count_n_mov: int = 0*, *homog_input_1_name: str = 'unset'*, *homog_input_1_date: datetime.datetime = datetime.datetime(1900, 1, 1, 0, 0)*, *homog_input_1_value1: int = 0*, *homog_input_1_value2: int = 0*, *homog_input_1_value3: int = 0*, *homog_input_2_name: str = 'unset'*, *homog_input_2_date: datetime.datetime = datetime.datetime(1900, 1, 1, 0, 0)*, *homog_input_2_value1: int = 0*, *homog_input_2_value2: int = 0*, *homog_input_2_value3: int = 0*, *homog_input_3_name: str = 'unset'*, *homog_input_3_date: datetime.datetime = datetime.datetime(1900, 1, 1, 0, 0)*, *homog_input_3_value1: int = 0*, *homog_input_3_value2: int = 0*, *homog_input_3_value3: int = 0*)

This class abstracts the program ww3_ounf. It is an extension of the class *pyww3.ww3.WW3Base()*.

populate_namelist()

Create the namelist text using NOAA's latest template.

PYTHON MODULE INDEX

p

INDEX