wrfxpy Documentation

Release 1.0.0

Martin Vejmelka

Nov 25, 2018

Contents

wrfxpy is a set of software modules that provide functionality related to running WPS and WRF.

In particular, the modules herein can:

- manipulate wps/input/fire namelists
- place and setup domains dynamically
- download GRIB files from various GRIB sources
- execute geogrid, ungrib, metgrid, real, WRF
- monitor WRF execution
- perform fuel moisture data assimilation using RAWS observations from the Mesowest network
- postprocess netCDF files to generate raster images or KMZ raster files
- assemble simulation outputs into coherent packages for visualization and synchronize them with a remote *wrfxweb* server

CHAPTER 1

Basic topics

1.1 Installation

1.1.1 Recommended method

We recommend using the Anaconda Python distribution. Alternative installation instructions are given at the bottom.

WPS/WRF-SFIRE

Please follow the instructions on OpenWFM to run WPS/WRF with real data. Ensure that you have working WPS/WRF installation is available and compatible with the MPI on your cluster. Note that *wrfxpy* will not modify the WPS/WRF installation, instead for each job, it will clone their directories into its workspace.

Attention: Past this point, you should be able to run a fire simulation yourself, that is have a working `WPS/WRF-SFIRE` installation including `WPS-GEOG` and fuels/topography downloaded. You should be able to submit a parallel job into your cluster/supercomputer to run `wrf.exe`

Python and packages

Download and install the Python 2 Anaconda Python distribution for your platform. We recommend an installation into the users home directory.

:: wget https://repo.continuum.io/archive/Anaconda2-5.3.0-Linux-x86_64.sh chmod +x Anaconda2-5.3.0-Linux-x86_64.sh ./Anaconda2-5.3.0-Linux-x86_64.sh

Install pre-requisites:

```
conda install basemap netcdf4 pyproj paramiko dill
conda install -c conda-forge simplekml pygrib f90nml
conda install -c anaconda numpy
```

(continues on next page)

(continued from previous page)

```
pip install MesoPy
pip install python-cmr
```

Add your environment (~/.custom.csh) the following line setenv PROJ_LIB "$HOME/anaconda2/share/proj"

or if you are using bash to ~/.profile or ~/.bashrc export PROJ_LIB="$HOME/anaconda2/share/proj"

Note that `conda` and `pip` are package managers available in the Anaconda Python distribution.

wrfxpy

Next, clone the *wrfxpy* code:

```
git clone https://github.com/openwfm/wrfxpy.git
```

configuration

And finally, a `etc/conf.json` file must be created with the keys discussed below. A template file `etc/conf.json.initial` is provided as a starting point.

Configure the system directories, `WPS/WRF-SFIRE` locations and workspace locations by editing the following keys:

```
"workspace_path": "wksp"
"wps_install_path": "path/to/WPS"
"wrf_install_path": "path/to/WRF"
"sys_install_path": "/path/to/wrfxpy"
"wps_geog_path" : "/path/to/wps-geogrid"
```

Optionally, the *wrfxpy* installation can be connected to a visualization server wrfxweb. The following keys are all optional (and only used if the postprocessed results of simulations are uploaded).

```
"shuttle_ssh_key": "path/to/your/priv_ssh_key/to/remote/host",
"shuttle_remote_user" : "remote_username",
"shuttle_remote_host" : "remote_hostname",
"shuttle_remote_root" : "remote directory for output storage"
```

This concludes the `etc/conf.json` file.

Next, *wrfxpy* needs to know how jobs are submitted on your cluster. Create an entry for your cluster, here we use `speedy` as an example:

```
{
  "speedy" : {
    "qsub_cmd" : "qsub",
    "qsub_script" : "etc/qsub/speedy.sub"
  }
}
```

And then the file `etc/qsub/speedy.sub` should contain a submission script template, that makes use of the following variables supplied by *wrfxpy* based on job configuration:

- `%(nodes)d` the number of nodes requested
- `%(ppn)d` the number of processors per node requested
- `%(wall_time_hrs)d` the number of hours requested

- `%(exec_path)d` the path to the wrf.exe that should be executed
- `%(cwd)d` the job working directory
- `%(task_id)d` a task id that can be used to identify the job
- `%(np)d` the total number of processes requested, equals `nodes` x `ppn`

Note that not all keys need to be used, as shown in the `speedy` example:

```
#$ -S /bin/bash
#$ -N %(task_id)s
#$ -wd %(cwd)s
#$ -l h_rt=%(wall_time_hrs)d:00:00
#$ -pe mpich %(np)d
mpirun_rsh -np %(np)d -hostfile $TMPDIR/machines %(exec_path)s
```

The script template should be derived from a working submission script.

Attention: You are now ready for your first fire simulation, continue with *Quickstart*.

1.1.2 Custom installation

If Anaconda python is not practical, a different python distribution can be used. Below is a list of packages the system requires:

- Python 2.7+
- Basemap to render the rasters
- simplekml to build KMZ files
- f90nml to manipulate Fortran namelists
- pyproj to place domains dynamically in LCC projection
- paramiko to communicate over SSH with remote hosts
- netCDF4 to manipulate WPS and WRF files
- MesoPy to retrieve fuel moisture observations from Mesowest

wrfxpy is installed by cloning a GitHub repository

```
git clone https://github.com/openwfm/wrfxpy.git
```

Configure *wrfxpy* by editing `etc/conf.json` as above and then continue with *Quickstart*.

1.2 Quickstart

Important: It is imperative that a working WRF-SFIRE installation is available. Please first follow the installation instructions *Installation*.

1.2.1 First fire forecast

The simplest way to start is to invoke the standalone script

```
./simple_forecast.sh
```

The script will ask you a series of questions with sensible defaults and at the end will create a JSON configuration file and finish with instructions on how to run the simulation.

1.2.2 Example fire forecast

To perform a fire forecast, the script `forecast.sh` has to be executed with a JSON configuration file as an argument, for example:

```
./forecast.sh <json-configuration-file>
```

An example configuration script is `examples/simple_fire.json`, also listed here for convenience. The script has most of the values filled out but there are some placeholders.

Please set the following values:

- `geogrid_path` should point to the directory with your WPS-GEOG data
- `num_nodes` are the number of nodes to use for the parallel job
- `ppn` the number of processors per node to use
- `wall_time_hrs` number of hours of wall time to reserve for the job
- `qsys` the queueing subsystem id which point into `etc/clusters.json`

```
{
  "grid_code": "test",
  "grib_source": "NAM",
  "wps_namelist_path": "etc/nlists/default.wps",
  "wrf_namelist_path": "etc/nlists/default.input",
  "fire_namelist_path": "etc/nlists/default.fire",
  "emissions_namelist_path": "etc/nlists/default.fire_emissions",
  "geogrid_path": "/path/to/your/WPS-GEOG",
  "num_nodes": 10,
  "ppn": 12,
  "wall_time_hrs": 3,
  "qsys": "sge",
  "start_utc": "T-30",
  "end_utc": "T+300",
  "domains" : {
    "1" : {
      "cell_size" : [1000, 1000],
      "domain_size" : [91, 91],
      "center_latlon" : [39.1, -105.9],
      "truelats" : [38.5, 39.6],
      "stand_lon" : -105.9,
      "time_step" : 5,
      "history_interval" : 15,
      "geog_res" : "0.3s",
      "subgrid_ratio" : [50, 50]
    }
  },
```

(continues on next page)

(continued from previous page)

```
  "ignitions" : {
     "1" : [ {
      "start_delay_s" : 600,
      "duration_s" : 240,
      "lat" : 39.894264,
      "long" : -103.903222
     } ]
  },
 "postproc" : {
    "1" : [ "T2", "PSFC", "WINDSPD", "WINDVEC", "FIRE_AREA", "FGRNHFX", "FLINEINT",
↪"SMOKE_INT" ]
  }
}
```

This example configuration runs a fire simulation with the following settings:

- a single domain configuration with a domain placed approximately around an ignition point
- use NAM as the source for initial and boundary conditions
- start at time now minus 30 mins and run a 5 hour simulation
- use 10 nodes, 12 CPU cores per node, allow a wall time of 3 hrs, the queue manager is `SGE` (Sun Grid Engine)
- use the default WPS/WRF/fire/emissions namelists as base
- ignite the fire 600s after the start of the simulation and deactivate the ignition after 4 minutes.
- generate surface temperature maps, wind information and fire fields for domain 1, the where the fire is burning

Tip: To learn how to configure jobs in more detail, refer to *Forecasting*.

1.3 Forecasting

The script `forecast.sh` serves to run weather forecasts, fire danger forecasts and fire simulations depending on its settings.

The script requires a JSON configuration file to control its execution, for an example refer to the *Quickstart*. The configuration file is JSON dictionary with the keys described in the following sections. Not all keys are required.

1.3.1 Domains

Domains are shared by WPS and by WRF and as they are very important, they have their own section. All domains are given in the key `domains : [dict]`.

The first domain is always the top-level domain, and subsequent domains are always child domains such that their parent is always defined before they are. Each domain can be precomputed or dynamically placed. In the following, we detail configuration of each type.

Note: in the input namelist, wrfxpy will set the top level domain as `specified`, while all other domains will be `nested`.

All domains must have the following keys:

- `history_interval : [integer]` the history interval in minutes, default is 60

- `parent_id : [integer]` the id of the parent domain (can omit for parent domain with id=1)

A top-level precomputed domain must have the following keys:

Example:

```
"domains" : {
  "1" : {
    "time_step" : 5,
    "precomputed" : "path/to/precomputed/geo_em.dYY.nc",
    "history_interval" : 15
  }
```

This will cause the use of the domain as precomputed in `colorado_domain_1.nc` with a time step of 5 seconds and a history interval of 15 mins. All spatial information will be retrieved from the netCDF file.

Top-level dynamically placed

A top-level dynamically placed domain must have the following keys:

- `time_step : [integer]` time step in seconds
- `subgrid_ratio : [integer, integer]` the refinement ratio for x and y direction for the fire grid, default is 1, 1
- `cell_size : [integer, integer]` the size of one grid cell in meters in DX, DY order (placed)
- `domain_size : [integer, integer]` the number of grid points in longitudinal and latitudinal direction (placed)
- `geog_res : [string]` the resolution of geographical/fuel data to use for the domain (placed)
- `center_latlon : [float, float]` the latitude and longitude of grid center (placed)
- `truelats : [float, float]` the true lattitudes of the LCC projection
- `stand_lon : [float]` the standard longitude of the LCC projection

Example:

```
"domains" : {
  "1" : {
    "cell_size" : [1000, 1000],
    "domain_size" : [101, 101],
    "center_latlon" : [39.894264, -103.903222],
    "truelats" : [39.2, 40.5],
    "stand_lon" : -103.903222,
    "time_step" : 5,
    "history_interval" : 15,
    "geog_res" : ".3s",
    "subgrid_ratio" : [40, 40]
  }
}
```

This will set up a top level domain domain with cell size 1km and 101x101 grid points, centered on the location [39.894264, -103.903222], with true latitudes and standard longitude as provided. The time step will be 5 seconds and history interval will be 15 mins. High resolution `.3s` geographical/fuel data will be used to construct the domain. The domain will have a fire grid 25m x 25m.

If the domain is precomputed, the following key must be set:

- `precomputed : [string]` the precomputed `geo_em.dYY.nc` file path relative to wrfxpy installation

Then, all values are loaded from the `geo_em.dYY.nc` file except for `time_step` and `history_interval`.

Child domain statically placed

A child domain requires the following keys:

- `parent_time_step_ratio : [int]` ratio of child time step to parent time step
- `parent_cell_size_ratio : [int]` ratio of the size of the child cell to the parent cell
- `parent_start : [int, int]` the x, y coordinates of the parent grid where this grid starts
- `parent_end : [int, int]` the x, y coordinates of the parent grid where this grid ends

Example

```
"domains" : {
  "1" : {
    "time_step" : 5,
    "history_interval" : 30,
    "precomputed" : "precomputed/my_grids/colorado_domain_1.nc"
  },
  "2" : {
    "parent_id" : 1,
    "parent_time_step_ratio" : 4,
    "history_interval" : 30,
    "precomputed" : "precomputed/my_grids/colorado_domain_2.nc"
  }
}
```

If the child domain is precomputed, again all these values are read in from the `geo_em` file automatically except timing information: `parent_time_step_ratio` must still be set.

- `precomputed : [string]` the precomputed `geo_em.dYY.nc` file path relative to wrfxpy installation

Child domain placed by bounding box

- `parent_cell_size_ratio : [int]` the ratio of cell size to parent cell size
- `parent_time_step_ratio : [int]` ratio of child time step to parent time step
- `bounding_box : [float, float, float, float]` the bounding box the domain should enclose as [min_lon, min_lat, max_lon, max_lat]

Examples

```
"domains" : {
  "1" : {
    "time_step" : 50,
    "history_interval" : 30,
    "precomputed" : "precomputed/my_grids/colorado_domain_1.nc"
  },
  "2" : {
    "parent_cell_size_ratio" : 3,
    "parent_time_step_ratio" : 3,
    "bounding_box" : [-105, 39, -105.5, 39.5],
    "history_interval" : 15,
    "geog_res" : ".3s",
    "subgrid_ratio" : [50, 50]
```

(continues on next page)

(continued from previous page)

```
    "parent_time_step_ratio: [int]``
  }
}
```

The value must be a dictionary mapping `geo_em.dYY.nc` files to their actual location.

1.3.2 WRF-SFIRE inputs

All of the following keys are required.

- `grid_code : [string]` the grid code is part of the job id and semantically should identify the configured grid
- `grib_source : [string]` must be one of NAM, NARR or HRRR
- `geogrid_path : [string]` the path to `WPS GEOG` data directory
- `start_utc : [esmf_time]` the start time of the simulation in ESMF format
- `end_utc : [esmf_time]` the end time of the simulation in ESMF format

The keys in the remainder of this section are optional.

- `ignitions : [dict]` (optional) is a dictionary of domains (string identifier, e.g. "1") to a list of ignitions that should be added to the domain, each being a dictionary with the following keys:
- `time_utc : [esmf_time]` time of ignition
- `duration_s : [int]` the length of time the ignition is active
- `latlon : [int]` the latitude and longitude of the ignition point

  Including this option causes the fire model to be switched on in each domain listed. A total of five ignitions is allowed (combined for all domains). For example

```
"ignitions" : {
  "1" : [],
  "2" : [ {
    "time_utc" : "2016-03-30_13:14:00",
    "duration_s" : 240,
    "latlon" : [39.894264, -103.903222]
  } ]
}
```

This would ignite a single fire 10 minutes after simulation start at the given lat/lon, hold the ignition for 4 minutes. In the first domain, the fuel moisture model will be switched on and fire danger calculations will be performed.

Important:

- All ignitions are point ignitions.
- All ignitions have a rate of spread parameter set to 1m/s and the maximum radius 200 m, see WRF-SFIRE documentation
- If a domain is listed without any ignitions, the fire model is switched on and computes quantities related to fire danger, such as fire spread rates, fuel moisture values, etc.

1.3.3 Namelist templates

All of the following keys are required.

- `wps_namelist_path : [string]` the WPS namelist template
- `wrf_namelist_path : [string]` the WRF namelist template
- `fire_namelist_path : [string]` the fire namelist template
- `emissions_namelist_path : [string]` the file_emissions namelist template

1.3.4 Parallel job configuration

The following keys are required.

- `num_nodes : [int]` the number of parallel nodes to use for WRF execution
- `ppn : [int]` the number of processors per node to request
- `wall_time_hrs : [int]` the wall time to request from the schedule in hours
- `qman : [string]` the queue manager to use, must be `sge`

1.3.5 Fuel moisture data assimilation

The key `fuel_moisture_da` is optional. If given, it needs to contain two keys:

- `domains : [list(int)]` a list of domains for which to run data assimilation

Important: In addition to this, the file `etc/tokens.json` must contain the key `mesowest_token : [string]`, which will be used to access the Mesowest API (you must obtain one here Mesowest).

The data assimilation code will download 10-hr fuel moisture observations from stations in the domain area and assimilate them into the equilibrium.

Example:

```
"fuel_moisture_da" : {
  "domains" : [ 1 ]
}
```

1.3.6 Postprocessing

The key `postproc`, when present contains a dictionary keyed by domain id (string), which identifies the variables to postprocess for each domain. For each listed variable, a PNG and a KMZ file is created and if required, a colorbar (configured in `var_wisdom`).

Additionally, if a remote visualization server is configured in `etc/conf.json`, the postprocessed rasters can be automatically sent either during the forecast itself or after the forecast is complete.

Example without remote shuttling:

```
"postproc" : {
  "1" : ["T2", "PSFC", "WINDSPD" ],
  "2" : ["T2", "FIRE_AREA", "WINDVEC"]
}
```

In this example, the postprocessed raster files are generated in the `products` subdirectory of the workspace directory where the job is executing.

Example with remote shuttling:

```
"postproc" : {
  "1" : ["T2", "PSFC", "WINDSPD" ],
  "shuttle" : "on_completion",
  "description" : "This should be a user-readable string that will be displayed to␣
↪the user"
}
```

The second example will send the complete visualization package to the remote server after the forecast is complete. The `description` string should be a short descriptive identifier of the simulation. This text will be shown to the user in the initial catalog menu on *wrfxweb* and thus also shouldn't be too long.

1.4 Standalone scripts

Although *wrfxpy* is meant to be an integrated system, some functionalities are exposed through separate scripts. These are detailed in this section.

1.4.1 Domain setup

The script `domain_setup.sh` accepts a domain configuration description and injects the domain configuration into a WPS nmelist file and into an input namelist file. Please refer to the domain configuration description in *Forecasting*.

Example:

```
./domain_setup.sh my_domains.json namelist.wps namelist.input
```

Assuming that `my_domains.json` contains the following:

```
{
  "1" : {
    "cell_size" : [1000, 1000],
    "domain_size" : [91, 91],
    "center_latlon" : [39.1, -105.9],
    "truelats" : [38.5, 39.6],
    "stand_lon" : -105.9,
    "time_step" : 5,
    "history_interval" : 15,
    "geog_res" : "0.3s",
    "subgrid_ratio" : [50, 50]
  }
}
```

Then both namelists will be setup for a single-domain configuration (1km grid cell size, 91 x 91 domain size, 20m fire grid).

Note: The namelist files are *overwritten*.

1.4.2 Grib retrieval and examination

The script `grib_retr.sh` accepts fourth arguments, the grib source identifier, the UTC start, the end time of a simulation in ESMF format and the ingest directory.

Example:

```
./grib_retr.sh HRRR 2016-03-26_14:00:00 2016-03-26_19:00:00 ingest
```

This will find out which GRIB2 files are required to perform this simulation and will download them into subdirectories of the `ingest` directory.

Tip: Using the *wrfxpy* ingest directory (or the same directory) consistently will make best use of the transparent local caching functionality. Any files that have already been downloaded are not re-downloaded.

The script `grib_tool.sh` allows the user to list the contents of a GRIB1/2 file and to convert it to a netCDF file.

Examples:

```
./grib_tool.sh list <grib-filename>

./grib_tool.sh to_netcdf <input-grib-filename> <message-to-convert> <output-netcdf-
↪file>
```

1.4.3 Postprocessing

The script `postprocess.sh` accepts four arguments, the wrfout file to process, the variables to postprocess (or an instruction file, see below), the prefix on which to base the filenames and the skip (the script will process every skip-th frame). The script always generates PNG files and KMZ files for each variable and timestamp.

Example:

```
./postprocess.sh /path/to/wrfout T2,PSFC my_directory/file_prefix 1
```

Alternatively, instead of listing the variables, a more detailed configuration controlling the colormaps, ranges and other parameters can be specified:

```
./postprocess.sh /path/to/wrfout @var_instructions my_directory/file_prefix 1
```

Where the file `var_instructions` contains:

```
{
  "FGRNHFX" : {
      "name" : "Grnd Heat flux",
      "colorbar" : "W/m^2",
      "colormap" : "jet",
      "transparent_values" : [0, 1],
      "scale" : [0, 6]
  }
}
```

Will show the colorbar in `W/m^2` units and change the displayed variable name to `Grnd Heat flux`, set the colormap to `jet`, ensure that values between 0 and 1 are not shown and fix the scale from 0 to 6.

Tip: For the default and more information on values that can be set, examine `src/vis/var_wisdom.py`.

1.4.4 Fuel moisture DA

The script `apply_fmda.sh` accepts a single wrfinput path argument and performs a data assimilation step using background covariance.

Example:

```
./apply_fmda.sh wrfinput_d01
```

The script will read in the timestamp from the wrfinput file, determine it's physical extent (lat/lon) and download all observations of 10-hr fuel moisture valid at that time available in the region. Then the equilibrium fuel moisture content is computed and adjusted with respect to the observations using the background covariance. The updated values are written back into the fuel moisture file.

1.4.5 SSH Shuttle

The script `ssh_shuttle.sh` accepts a local directory a remote directory name and an identifier and uploads the entire local directory with simulation results to the remote host configured in `conf.json` and registers the simulation in the `catalog.json` file on the remote server.

Examples:

```
./ssh_shuttle.sh wksp/my-simulation/products test_fire_april test_fire_april
```

The script scans all the files in `wksp/my-simulation/products` and uses SFTP to put them onto the remote host. The remote directory must be either an absolute path or (recommended) should be relative to the remote host root setup in `conf.json`. The identifier will be used as the description and also as the key under which the simulation is stored in `catalog.json` on the remote host.

1.4.6 Data cleanup

The script `cleanup.sh` provides functionality to:

- list all simulations that are available on a configured visualization server,
- remove a selected simulation, freeing up diskspace.

Examples:

```
./cleanup.sh list

./cleanup.sh delete <simulation-id-from-list>
```

CHAPTER 2

Advanced topics

2.1 Catalog and manifest

This document describes the catalog format and the manifest format. The catalog file collects computed simulations for the visualizaion server and points to the manifest file for each simulation. The manifest contains postprocessed rasters pertaining to a single simulation.

2.1.1 Catalog

The file `catalog.json` in the root visualization directory of the *wrfxweb* visualization system is a JSON file which stores the following information about each simulation:

- `manifest_path : [string]` the path to the manifest string
- `description : [string]` a description that is shown in the selection menu to a user
- `from_utc : [esmf_time]` the start time of the simulation in ESMF format
- `to_utc : [esmf_time]` the end time of the simulation in ESMF format

Example:

```
{
  "patch_fire": {
    "manifest_path": "patch3/patch.json",
    "description": "Patch Springs Fire [UT]",
    "to_utc": "2013-08-19_09:00:00",
    "from_utc": "2013-08-11_00:00:00"
  },
  "test_fire_3": {
    "manifest_path": "test_fire_3/wfc-two-domain-fire.json",
    "description": "2-domain test fire, viscosity=0",
    "to_utc": "2016-04-08_23:00:00",
    "from_utc": "2016-04-08_18:00:00"
```

(continues on next page)

(continued from previous page)

```
  },
  .
  .
  .
}
```

2.1.2 Manifest

The manifest file is a JSON file that collects information on which domains, timestamps and variables are generated from a simulation. The top-level object is a dictionary keyed by string domain identifier ("1", "2", ...) and contains an object keyed by ESMF time ("2016-03-30_00:00:00", ...) which in turn contains a dictionary keyed by variable names (e.g. "T2", "WINDVEC", ...). The postprocessing results for each variable are represented by dictionary keys as follows:

- `colobar : [string]` (optional) path to the colorbar, if any
- `raster : [string]` path to the display raster (PNG file)
- `kml : [string]` (optional) path to the KMZ file for possible download
- `coords : [string]` the corner coordinates of the PNG file (geolocation)

An partial example of one variable in a file is below:

```
{
  "1" : {
    .
    .
    "2016-03-30_16:15:00" : {
    .
    .
      "FMC_G": {
        "colorbar": "patch-2013-08-14_12:00:00-FMC_G-cb.png",
        "raster": "patch-2013-08-14_12:30:00-FMC_G-raster.png",
        "kml": "patch-2013-08-14_12:30:00-FMC_G.kmz",
        "coords": [
            [ -113.15496826171875, 39.978614807128906 ],
            [ -112.26193237304688, 39.978614807128906 ],
            [ -112.26193237304688, 40.65946960449219 ],
            [ -113.15496826171875, 40.65946960449219 ]
        ]
      }
    }
  }
}
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search