DTC HWRF Scripts Home

This webpage documents the project to rewrite the HWRF automation scripts in Python. At the time this project started, there were a total of four different scripting systems within NCEP and DTC alone, which independently implemented the HWRF, using a combination of ksh, Perl, awk, sed, grep and other languages. The systems had insufficient fault tolerance and error detection, and were a major drain on manpower. Management and developers came to the decision that the scripts need to be replaced.

We document here both the ongoing work to add functionality to the Python-based HWRF, hereafter called the pyHWRF, and also provide high-level documentation of the existing functionality and design of the system. Low-level documentation will be provided automatically generated from the Python docstrings within the source code itself. This website only attempts to provide a high-level description of the modules, packages, scripts and how to use them and modify them. Detailed descriptions of every single function are available in the docstrings.

Presentations and Workgroup Notes

Original Project Proposal (September 19, 2013)

This is the original project proposal, given to management in the NCEP Environmental Modeling Center (EMC), NCEP Central Operations (NCO) and the Developmental Testbed Center (DTC) on September 19, 2013. It was at that meeting that management decided on Python for the rewrite (alternatives being Ruby or 1993 ksh). Their decision was primarly due to prior existing knowledge of Python throughout the atmosphere and ocean modeling community, especially within NOAA.

HWRF Script Rewrite and Unification

September 19, 2013

Timothy Brown and Samuel Trahan

Overview

The HWRF scripting system in its present state is over-complicated, has insufficient fault tolerance and is a major drain on manpower; it is sorely in need of replacement. HWRF scripts exist in three different forms at present: an operational system run by NCO, a similar system run by EMC, and a less capable but more portable system run by everyone else and maintained by DTC. The system used by NCO and EMC has insufficient fault tolerance, causing occasional operational failures and frequent failures in development parallels on less reliable machines. Furthermore, the system is over-complicated: more than 38000 lines for the base workflow, and tens of thousands more for graphics and automation. That makes any development or debugging major tasks rather than the simple efforts they should be. Continuing to use the current system, as it expands and gets increasingly complex, will be infeasible and will result in ever-increasing drain on resources on EMC, DTC and NCO.

One of the reasons for the issues in the present system is the choice of ksh88 as primary language. That language lacks basic functionality such as message passing between jobs, internal state beyond simple strings and numbers (and limited 1D arrays), nor failure information beyond 8 byte integers. The language has no built-in support for any basic operations such as date manipulation, string processing, binary I/O, numerical computation and the like, which are part of the standard library of most non-shell languages. All of this forces the inefficient use of numerous subprocesses, temporary files and extraneous tiny fortran programs, which slow the scripting system and drastically increase its complexity.

A second cause of the problems are that everyone is working on a different set of scripts than EMC. The DTC scripting system is simpler, and incorporates some of the fault tolerance features that EMC's scripts should have, but lacks our real-time capabilities. Having to maintain two separate scripts wastes an enormous amount of development time in both EMC and DTC. NCO has to make many custom changes, controlled by numerous checks to the $PARAFLAG variable simply to change data paths and other features that should be much easier to modify than they presently are. This wastes development resources our organizations could otherwise be putting towards improving forecast skill or reliability, or towards our other duties.

To solve these problems, the EMC and DTC HWRF groups would like to rewrite the present ksh-based HWRF systems to create a single Python-based system that is simpler, easier to debug and develop, more fault tolerant, easier to reconfigure, and less error-prone. The choice of Python 2.6.6 has come from a discussion between EMC, NCO and DTC, and will be explained later in this document. We believe this rewrite would be of great benefit for all three of our organizations as well as for our external collaborators whose contributions have been invaluable.

However, before we begin the rewrite, we need confirmation from NCO that Python 2.6.6, presently installed on WCOSS compute nodes, is acceptable for an operational system. We are not asking that anyone approve our entire implementation in advance; we just want to be assured that the use of Python 2.6.6, in and of itself, will not be the reason the implementation will be refused. This is because it is infeasible to maintain the Python-based system as a third system from now until March 2014, and it will also be infeasible to rewrite an entirely new ksh-based system at the last second if stakeholders request that all Python be removed.

EMC and DTC have approved this plan. We just need confirmation from NCO.

Language

First: a bit of history. The current EMC and DTC scripts are written in Korn shell. EMC uses the 88 variant, while the DTC variant is 93. The DTC scripts are more functionally oriented and rely on features only present in the newer 93 variant. The reason for the use of ksh88 in EMC is that it was mandated in the original implementation in 2007 due to limitations of NCEP's contract with IBM. Those limitations no longer exist: Python is used even by the operating system scripts, and is installed on WCOSS compute nodes.

The DTC and EMC have agreed a script unification is of vital importance. It will streamline the R2O and O2R transition processes by the same scripts and framework used in operations to all researchers. This unification will reduce the time and money spent by both the DTC and EMC annually by eliminating the duplication of work and providing a more robust framework that will be less error prone. We believe it will also reduce NCO's expenditure of resources due to the new system being simpler, easier to debug, more fault-tolerant and easier to configure (ie.: less SPA time to implement it!)

Upon consulting all the stakeholders, including NCO, it was decided that Python 2.6 provides the best path forward for HWRF system. As this scripting language and version is currently installed on WCOSS, Jet, Zeus and Yellowstone. The new scripts for driving the HWRF system will be called pyHWRF. In the future we suggest a migration to Python 2.7 since it is the long-term support release of Python 2.x and has many upgrades, fixes and forward-compatibility features. The choice of Python 2.6 instead of 2.7 is merely due to RedHat Linux being several years behind on Python updates.

Timeline

The ideal timeline is to have pyHWRF ready for the 2014 operational implementation. This is currently slated for December 2013. In order to reach this goal, the DTC will contribute 0.5 FTE (Timothy Brown) with a matching contribution by EMC (Sam Trahan and Zhan Zhang). The following milestones have been agreed upon:

Milestone	Earliest Date	Latest Date
NCO approval on Python	8/1/2013	9/31/2013
Postprocessing in pyHWRF	9/31/2013	10/14/2013
Rapid Prototyping of pyHWRF	10/14/2013	10/31/2013
pyHWRF Framework	10/31/2013	Ongoing

The post-processing phase is an ideal candidate to start the migration to pyHWRF, as it is a small self contained component of the current system that needs to be modified to enable EMC's method of running the tracker (continuously waiting for WRF output, versus a single instance at the end of the WRF forecast).

Tools

Python provides a wealth of tools to interact with the operating system and perform string and date manipulation. pyHWRF will build upon the core python module and leverage already existing community modules to provide IO, plotting and numerical routines.

- Base workflow (the only part given to NCO)
  - Python 2.6.6 currently installed on WCOSS, Zeus, Jet and Yellowstone.
  - The latest 2.7.x release (presently 2.7.5) is preferred as soon as possible.
- Additional modules to be used in the non-NCO version for graphics and other tasks:
  - IO modules
    - NetCDF4-python -- NetCDF 3 & 4 manipulation
    - pygrib -- wrapper around NCEP g2c GRIB 2 library
    - ECMWF pybufr -- buffer (to be determined)
  - Plotting modules
    - Matplotlib
    - Basemap

- Numerical modules
  - Scipy
  - Numpy
- Optional (to be determined if they will be useful)
  - ESMPy -- interface to ESMF

Modules Abstraction Layer

pyHWRF will follow the object oriented paradigm providing polymorphism and a message abstraction layer. Key components in the implementation will be:

- exception handling
- built in system interaction
- portability to other platforms (GAEA will be one of the early tests of this)
- long term support
- message passing
- logging
- unit tests

All development and maintenance will occur within the Subversion revision control system.

Notes PyHWRF Telecon Friday, June 20, 2014

Notes PyHWRF telecon Friday June 20, 2014

Vijay, Sam, Tim, Ligia, Christina

Today is the last day of Sam's visit to NOAA-ESRL-GSD-DTC. We have made a lot of progress in the PyHWRF scripts. Most parts of the end-to-end system are working now, including the atmospheric and ocean initializations, GSI, coupled forecast model, postprocesing, and tracking. A few parts still need some debugging and testing, and there a few missing parts. Somoe of the missing parts are needed for operations, others for the community release.

Jet reservations for the realtime parallels start on Tuesday 7/24, and Sam would like to have the system running for WP by then. Sam will set it up to work with HSS originally, and Tim will follow up with making it work with Rocoto.

Missing

- pre-atmos (only needed for wcoss to get data from remote locations)
- delivery of tracks (for operations)
- emails SDM (for operations)
- archive

Needed for public release

- Option to disable scrubbing (Sam; this is halfway done already)
- Decide on default directory structure (Sam and DTC)
- Write a modified pre-master to create holdvars and directory structure, so community users can run individual steps using what is in scripts (Sam)
- Fix some problems in relocate (there are many paths in the relocate procedure. PyHWRF is working for most but not all) (Tim and Sam)

Test, test, test

- Goal of test is to make sure that PyHWRF script behave like ksh scripts. There are some reproducibility issues with the ksh (especially with GSI and waiting for ocean initialization files), so need to be careful.
- It is necessary to run cases using the ksh scripts. We cannot use the H214 results because of differences between that and what is in the trunk. E. g., the surface layer parameterization in WRF is different between H214 and the trunk (trunk and operations have the same parameterization as the 2013 model but H214 has a change to remove bias correction).
- Decided to test approximately 20 cases with no GSI and look for match between PyHWRF and ksh results. Then launch PyHWRF for large number of cases to make sure it runs reliably. Cases will be mostly AL and EP, with a couple WP.
- Mingjing and Qingfu will provide list of cases that exercise all paths of initialization (approx 20).
- Vijay said that Mingjing and Qingfu may be able to help with debugging some cases of the initialization that are not working.
- Initial 20-case test will be run by DTC on zeus "by hand", that is, using the Py experiment (no HHS or Rocoto). Larger test may use EMC's allocation.
- Code will point to branches/HWRF, except for ocean, which will point to mpipomtc_wp (aka, global branch). Note that PyHWRF scripts do not work for mpipom_tc code in b/HWRF.
- For WRF component, Ligia discussed with Laurie. Laurie will update WRF in b/HWRF today so it has all fixes needed to make H214 WRF compile correctly for ARW/NMM.
  In order to compare ksh and PyHWRF scripts, the ksh scripts will be from branch mpipomtc_wp (aka, global branch). Ligia spoke with Biju, who will update the scripts in mpipomtc_wp (aka, global branch) in the next couple of hours. This update is needed to bring to branch a few fixes made by Sam this week.
- Later we need to commit the scripts and ocean code from mpipomtc_wp (aka, global branch) to b/HWRF. This is needed for the release, since PyHWRF scripts only work with this new code. Before that, Ligia will run a case of Sandy and compare results between b/HWRF and mpipomtc_w p (aka, global branch). Ligia confirmed with Biju that results should be identical.

Python Post-Processing Presentation from EMC to NCO (April 4, 2014)

This presentation was given to describe the Python post-processing and delivery systems that are expected to be in the operational HWRF in May 2014. This presentation was given to the NCEP Central Operations developers that would be implementing HWRF in operations, as well as Environmental Modeling Center researchers that would be supporting them. The purpose was to describe the product delivery system, the implementation of the HWRF components, and how they link to one another.

- http://www.emc.ncep.noaa.gov/HWRF/weeklies/APR14/python-post-processing.pdf

Repository

Due to the rapid development of the 2014 HWRF system over the past weeks, this project is presently advancing in three branches. Soon, all three will be merged into the HWRF trunk. For now, relocation work is ongoing in the following branch, which runs the 2013 HWRF system:

- https://svn-dtc-hwrf.cgd.ucar.edu/branches/pyHWRF/

The production 2014 system is in the "H214" branch, which only runs pyHWRF post-processing. This branch will be deleted once the WRF and UPP updates for 2014 are merged to the trunk:

- https://svn-dtc-hwrf.cgd.ucar.edu/branches/H214

The pyH214 branch runs the 2014 system, and has additional experimental code added in to run most of the initialization and the forecast in Python. It will soon be merged to the pyHWRF branch, and the pyH214 will be deleted:

- https://svn-dtc-hwrf.cgd.ucar.edu/branches/pyH214

Documentation

The documentation of the pyHWRF is split into several pages due to the large size of the available documentation. New readers are directed to the Technical Overview which provides an overview of the entire system, and links to additional documentation.

FIXME: add the missing pages.

- Technical Overview – an overview of the pyHWRF system
  - Intercycle and Workflow Layers – describes the available and planned implementations of the optional top two layers of the system, which implements the interaction with the batch system
    - ecflow HWRF – the automation system used by NCEP Central Operations to run the operational HWRF. It is based on the ECMWF ecflow automation system.
    - HHS and kick_scripts – the system used by the NCEP Environmental Modeling Center (EMC) to automate EMC HWRF simulations for the past few years.
    - Rocoto HWRF – Rocoto is a NOAA-originated workflow management system with capabilities similar to ecflow. EMC plans on using this instead of the HHS+kick_scripts combination.
  - Scripting Layer – sets the pyHWRF data locations, and controls when and where each HWRF component is run
    - NCEP jobs/scripts/ush – the three tier implementation used by NCEP
  - Experiment Layer – sets the experiment configuration
  - HWRF Layer – runs the HWRF components. This is the `hwrf` Python package.
  - Platform Compatibility Layer – utilities that sit between the `hwrf` package and Python itself, providing a platform-independent environment.
- Reasons for Design Decisions – explanations about why certain decisions were made in the design of the pyHWRF
- Detailed documentation – automatically generated from Python docstrings. This details every function, class and property in the Python code.

UML

This section is a placeholder for additional future documentation of the project structure.

- Classes
- Packages

Tasks

In order to complete the re-write to obtain a system that will produce results equivalent to the 2014 HWRF implementation the following needs to be achieved.

- High level model/system flow
- Low level components

The high level system.

The flow is defined within `ush/hwrf_expt.py`, while the system configuration is defined within `parm/hwrf.conf`.

To run an experiment/storm you need to modify the initial kick script `pyhwrf_driver.py` and then execute for the storm.

```
fe2$ cd kick
fe2$ $EDITOR pyhwrf_driver.py
fe2$ ./pyhwrf_driver.py 2012102806 18L HISTORY
```

The items to edit within `pyhwrf_driver.py` are

Variable	Value (examples)
diskgroup	'dtc-hurr'
rungroup	'dtc-hurr'
projdir	'/pan2/projects/dtc-hurr'

Note: That the values have to be quoted.

Low level components

- [x] **post (upp, tracker)**
- [ ] **track analysis**
- [x] **wps (geogrid, ungrib, metgrid)**
- [ ] **wrf (main forecast)**
- [ ] **wrf analysis**
- [ ] **wrf ghost**
- [ ] **vortex relocation**
- [ ] **mpi pom init**
- [ ] **GSI**

Team Calendars