Community Unified Post Processor (UPP) Code Management Plan

April 15, 2019
Point of Contact: Kate Fossell [fossell@ucar.edu]

1 Introduction

The Unified Post Processor (UPP) software package was developed by the Environmental Modeling Center (EMC) of the National Centers for Environmental Prediction (NCEP) and is used operationally for models maintained by NCEP. This package provides the Numerical Weather Prediction (NWP) community with a common tool to post-process output from multiple models, ranging from global to regional scales. UPP can be used to post-process output from a variety of NWP models, including the Weather Research and Forecasting (WRF) Advanced Research WRF (ARW) and Non-hydrostatic Mesoscale Model (NMM), Non-hydrostatic Multi-scale Model on the B grid (NMMB), Global Forest System (GFS), Climate Forecast System (CFS), and Finite-Volume Cubed Sphere (FV3).

The Developmental Testbed Center (DTC) will serve to establish a link between the operational and research communities by maintaining a community UPP repository and providing regular public releases and user support for this software package. In order to benefit the entire NWP community, the community UPP repository must be maintained in a way that allows future updates and enhancements to be contributed by, and shared between, both the operational and research communities.

Supporting and managing a version of the UPP package applicable to community and operational users requires a plan for coordinating the sharing of code between developers with diverse needs. The DTC will be responsible for maintaining a community UPP package that is kept in sync with the operational UPP package (O2R). Community users may contribute modifications and enhancements to the UPP package; these modifications will be tested and made available to the operational community (R2O). This document outlines the policies and procedures used to maintain a robust and flexible community UPP software package. The document contains descriptions pertaining to:

- Community UPP code repository
- Code Contributions
- Synchronizing the DTC Community UPP ⇔ NCEP UPP code repositories
- Release Schedule and User Support

2 Community UPP Code Repository

2.1 Physical Storage

The hardware, version control software, and repositories used to store and access the community UPP repository are maintained and administered by the National Oceanic Atmospheric Association (NOAA) / National Weather Service (NWS) via Virtual Laboratory (VLab). VLab is a set of services and an Information Technology (IT) framework developed to facilitate the collaboration of software within NOAA/NWS and its partners in effort to streamline and accelerate Research to Operations (R2O) and Operations to Research (O2R) transitions. It consists of integrated tools for project management, issue tracking, software repositories, continuous integration, and code review. More information can be found at https://vlab.ncep.noaa.gov/home.

VLab makes use of Git as its version control system. In 2018, the community UPP code and repositories were moved from SVN-based version control to Git, and pushed to VLab to be consistent with NOAA's Environmental Modeling Center (EMC) operational UPP code.

2.2 Software Directory Structure

One physical Git repository exists on VLab (described in 2.1) to house both operational EMC code and the community UPP package. This repository is called *EMC_post* and can be found at: https://vlab.ncep.noaa.gov/code-review/EMC_post. The master branch of *EMC_post* constitutes the operational source code. A branch called *DTC_post* exists within the *EMC_post* repository to house the community UPP package. The *DTC_post* branch is further described in section 2.2.1.

2.2.1 Community UPP Repository

The UPP community repository is by design a direct branch of the *EMC_post* repository. This community branch is called *DTC_post* and is a mirror image of the master branch of the *EMC_post* repository, but also contains an extra directory called *comupp* that houses DTC specific build support, library source code and submodule links to the library *src/crtm*, and any other community specific needs. Effort is made to keep the unipost source code within the master and community branches (*EMC_post/sorc/ncep_post.fd* and *DTC_post/comupp/src/unipost*) identical except where the need for difference is determined to be necessary or unavoidable. A DTC preprocessor flag (COMMCODE) is used to facilitate the sharing of files with minor code variations.

The structure of the *EMC_post* repository and *DTC_post* community branch is illustrated below:

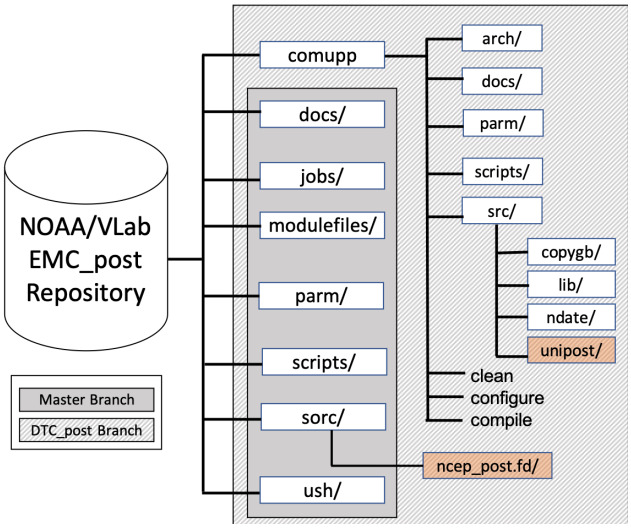

Figure 1. High-level overview EMC_post repository and DTC_post directory content

The *DTC_post comupp/* directory is the central location of the code needed to create executables of *unipost* and associated utility programs (i.e., *ndate, copygb*). The scripts: *clean, configure*, and *compile* are the command directives used to create the executables. These scripts use information in the *arch/* directory to build a version of UPP which is compatible with the system/architecture it has been installed on. The *comupp/src* directory contains all the source code. The source code includes the *lib/* directory which stores library source code and, in some cases (i.e., CRTM), contains a submodule to an external library repository. The main *unipost* code is in *src/unipost/* and is a copy of the *sorc/ncep_post.fd* directory in the NOAA EMC operational master branch of the *EMC_post* repository. Effort will be made to maintain a mirror image of the *sorc/ncep_post.fd*, unless changes are necessary. The utility programs, *ndate* and *copygb*, are also located within the *unipost* repository under the *comupp/src/* directory. These utilities were also developed by NOAA/EMC.

The contents of these directories are further expanded and defined in the following table.

Directory Name	Description
comupp/clean	Script to remove compilation files
comupp/configure	Script to configure for compile
comupp/compile	Script to compile
comupp/arch/	Architecture dependent support code
comupp/docs	User Guide Lyx files
comupp/makefile	Initial makefile called by compile
comupp/parm/	Output control parameter file
comupp/scripts/	Scripts to run unipost
comupp/src/lib/bacio/	binary I/O routines
comupp/src/lib/crtm2/	Community Radiative Transfer Model [Submodule to Github Repository]
comupp/src/lib/ip/	Interpolation Library

comupp/src/lib/sfcio/	NCEP GFS Surface files I/O API
comupp/src/lib/sigio/	NCEP GFS Sigma files I/O API
comupp/src/lib/sp/	Spectral Grid Transform Library
comupp/src/lib/gfsio	GFS I/O routines
comupp/src/lib/nemsio	NEMS I/O routines
comupp/src/lib/w3emc	GRIB1 code/decode Library
comupp/src/lib/w3nco	GRIB1 code /decode Library
comupp/src/lib/g2	GRIB2 support library
comupp/src/lib/g2tmpl	GRIB2 table support library
comupp/src/lib/xml	XML support – GRIB2 parameter file
comupp/src/lib/wrf_io	WRF netcdf I/O Library
comupp/src/lib/wrfmpi_stubs	Serial compilation support
comupp/src/unipost	Unified Post source code
comupp/src/copygb	Horizontal Interpolation Utility for grib1
comupp/src/ndate	Date formatting Utility

Table 1: DTC_post branch comupp directories

A user may create a branch based on the *DTC_post* branch to create a workspace for their development. User branches should be named such that they do not conflict with other branches. An important note is that when a user makes a copy of the *DTC_post*, the submodule to any external library repositories are NOT branched, unless they are explicitly changed.

Tags are used to store snapshots of the community UPP package when a version is officially released to the user community. These releases include the formal annual release and any bug fixes that are posted as a code revision. In the event of numerous user contributions, at any given time *tags* may be used to help insure the order of the code changes to the repository. A user may tag a branch; however, no development is to occur in the *tags/* directory.

2.2.2 Community UPP Libraries Structure

The libraries packaged with the community UPP are derivatives of the NOAA/EMC's *nceplib* repositories on VLab. These libraries will strive to be identical to the versions used by the operational UPP except where the need for difference is determined to be necessary. A DTC preprocessor flag is used to facilitate the sharing of files with minor code variations. The libraries will be upgraded annually with each release to remain current with EMC and more frequently, if needed. There are unit tests available from the EMC repository with each of these libraries. At this time the unit tests are not available in the DTC repository. Due to the size of the complete CRTM library, a subset of only the necessary code is housed in a Github repository and is maintained by the DTC in the same manner as the other libraries. A submodule for the CRTM Github repository is linked to the *DTC_post/src/lib/crtm* directory such that a clone of the *DTC_post* branch automatically populates the *crtm* directory with the proper code. This CRTM library repository can be found at: https://github.com/NCAR/UPP_CRTM. Development of any of the libraries packaged with the UPP should be discussed with the DTC prior to proposing code changes within the repository.

The structure of the DTC library source code within *DTC_post* branch follows:

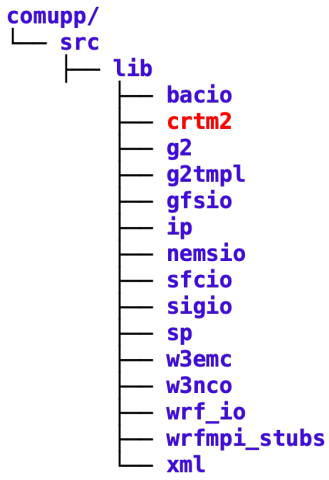

Figure 2. Library directories. Blue text indicates source code housed in DTC_post branch on VLab. Red text indicates source code linked by submodule to external Github repository.

3 Code Contribution Outline

3.1 Access to Repository

A developers' committee (DC) exists, which will govern the contributions made to the community UPP branch. Initial membership of the DC will include representatives from NOAA/EMC and the DTC. Individuals wishing to gain access to the community UPP branch should contact the DTC. The DTC will bring the request to the DC for approval. Access to the repository requires NOAA credentials, access to VLab, Redmine, and Gerrit components. The DC will approve read access to the repository. Read/write access will be granted to a subset of the DC based on need. Upon approval, the DC will coordinate with the user about next steps to gain access to the repository. In the event that NOAA VLab access cannot be granted, the DTC will provide a code package for the individual to do their development, and the DTC will manage checking in new code changes to the repository as needed. The DC will meet on an as needed basis.

3.2 Working in the Repositories

DC members with read/write access are encouraged to make a branch(es) of the *DTC_post* branch to work on changes they intend to make available to the community repository(ies) at a future date. This will facilitate the user in keeping current with the modifications that occur to the community code while they are developing their changes. The changes made under a user branch(es) are not used by the general community. These individual branches allow groups or an individual to create a quasi-permanent UPP version which contains changes their project requires, but are not ready to be incorporated into the community code.

3.3 Request to Accept Change into Community UPP Repositories

To check code modifications into the community repository, a DC member must request a code review. A code review meeting will be scheduled and should be attended by a majority of the DC

members. The DC members in attendance have the authority to accept or reject the proposed changes. For those with VLab access, the procedure to follow in order to prepare code for review is outlined below. Other users/developers should contact their POC at the DTC to obtain specific procedures; while some technical aspects may change, the general procedures will still be followed in this case.

1. Create a local copy of the repository by cloning the *DTC_post* branch of the *EMC_post* repository. Users must clone the specific branch in order to obtain the CRTM submodule as well. (Cloning the master *EMC_post* branch and trying to checkout the *DTC_post* branch will result in an empty *crtm* directory and necessitate extra steps to populate its contents).

   ```
   git clone -b DTC_post --recurse-submodules https://vlab.ncep.noaa.gov/code-review/EMC_post
   ```

2. Create a local branch off of *DTC_post* for development.

   ```
   git checkout -b your_branch
   ```

3. Add modifications, enhancements or additions into your local branch. All code changes proposed for inclusion to the community branch of the repository must meet the coding standards described in Appendix A.

4. Run available regression tests (Appendix B) on modified code, as well as any specific computing and scientific tests the developer may have to ensure the modifications are behaving as intended and have not adversely impacted existing UPP capabilities.

5. Commit your changes to your local branch

   ```
   git checkout your_branch
   git commit -a
   ```

6. Push your branch to the remote *EMC_post* repository on VLab.

   ```
   git checkout your_branch
   git push origin
   ```

7. Ensure your branch is synchronized with the *DTC_post* branch. If not and you need to merge with *DTC_post* first. Retesting after the merge is required.

8. Upon successful completion of testing (4) and pushing your branch to the remote *EMC_post* repository on VLab (6), a proposal of the modifications should be submitted to the DC. The developer's proposal must include a detailed description of the modifications to the system, a summary of the completed testing with results, instructions on usage of new feature(s), files modified, changes to program input/output, and code differences (unified diff format). The DTC will schedule a review meeting and notify the DC membership. The complete developer's proposal must be received by the DC no later than 24 hours prior to the

scheduled review meeting.

9. DC members should review all proposals for system commits prior to the scheduled review meeting. During the DC meeting, all code modifications are described and defended to the DC review members. The DC members may accept or reject any set of proposed changes. (Note: The successful completion of the regression tests is a necessary, but not sufficient, condition for acceptance of proposed changes.)

10. Approved system modifications may then be merged to the *DTC_post* branch of the *EMC_post* repository on VLab. In the event that multiple proposals are submitted prior to the same review meeting, the system changes will be prioritized by the attendees of the review meeting and the order of system upgrades will be determined. The DTC will manage merging of multiple proposals.

11. After all system modifications are in *DTC_post* branch, the DTC will retrieve the branch, compile UPP and supporting utilities, and run regression tests to ensure functional status. If the tests do not complete successfully the DTC will notify the party(ies) involved and work to correct the system error(s). This may include removal of a previously accepted modification. When the DTC determines the community UPP *DTC_post* branch is in a working state a notification will be made to the DC membership to that effect, including any removal of changes that may have occurred. Developers who have their modifications removed may correct the noted error and resubmit their proposal to the DC.

12. The developers whose system modifications have been accepted and verified should delete the branch they made in the *branches/* directory of the repository. If development is anticipated to continue on this branch it may remain with no required intervention.

13. The accepted changes not initiated by NCEP will be made available to NCEP to determine if the community modifications can be incorporated into the operational UPP.

4 Synchronizing the DTC Community UPP ⇔ NCEP UPP code repositories

To maintain a strong connection between the research and operational communities, the DTC community UPP branch must be kept synchronized with the NCEP operational UPP branch. This will require the DTC to merge all NCEP operational changes into the community UPP branch, as well as present the community changes accepted by the DC into the community branch to NCEP for possible inclusion in the operational master branch. A primary and secondary point of contact (POC) will be designated from the DTC staff who will work with the NCEP primary code manager to accomplish this goal. The remainder of this section outlines the communication required for repository synchronization.

4.1 DTC ⇒ NCEP

The DTC will submit non-NCEP initiated changes accepted by the DC into the community UPP branch to NCEP for acceptance into the operational UPP repository. The documentation submitted to initiate the DC approval process will act as the documentation provided to NCEP. The DTC will open a ticket in Redmine for each request submitted to NCEP. The DTC POC will

work directly with EMC code managers to incorporate the proposed changes and will authorize the final acceptance through Gerrit code review for the changes to be merged in the master branch of *EMC_post*.

4.2 NCEP ⇒ DTC

NCEP will send notification of updates via email messages generated by the operational VLab project management system or communicated by the NCEP principal code manager to the DTC POC. There are three forms of updates the DTC will receive from NCEP regarding the Community UPP branch. These include:

1) New operational UPP software capabilities (via VLab/Git)
2) Acceptance of community contributions to the operational repository (via Redmine)
3) Refusal of community contributions to the operational repository (via Redmine)

4.2.1 Operational UPP Software Modification

The DTC POC receives email notification that the operational UPP software has been modified. The Git or Redmine logs should describe the files that changed and provide a description of the change. The DTC will proceed as follows:

- The DTC will update the community branch, *DTC_post*, to pull in NCEP changes.
- The DTC will make a branch based off the community *DTC_post* branch.
- The NCEP changes will be incorporated into the *comupp* directory.
- The DTC will run regression tests.
- Upon successful completion of the regression tests, the modification will be brought before the DC for acceptance into the *comupp* directory of *DTC_post* branch.
- The details in Section 3.3 (Request to Accept Change into Community UPP Repositories) further explain the remaining procedure. The DTC will act as the originator during this procedure.

4.2.2 Acceptance of a Community Contribution

The DTC point of contact receives email notification that a community contribution has been accepted in to the operational UPP repository (master branch of the *EMC_post* repository). The DTC determines if the accepted operational implementation matches the initial submission or if modifications have been made. Variations to white space (blanks, tabs, and newlines) are not considered a modification for this purpose. If the change is identical there is no work to be done in the repository. The DC will be notified that the change was accepted by NCEP. The DTC will then update the tracking database as discussed in Section 4.1, DTC ⇒ NCEP, to reflect the acceptance of the contribution. White space changes will be reconciled by the DTC if necessary to keep the code as close as possible with the NCEP operational branch. The DTC may use the procedure outlined in Section 3.3 to modify the community branch with these changes.

In the event the modifications made by NCEP include variations to the initial DTC submittal the DTC will notify the DC of the changes made by NCEP. If any questions arise from the DC, the DTC will communicate those back to NCEP and iterate, as necessary, until a final version is agreed upon. Any code changes from the initial DTC submission to NCEP will terminate the acceptance of the current approval and reinitiate the process with the most recent modifications. Upon acceptance of the modifications, the DTC will follow the outline found in Section 4.2.1. All communication shall be documented in the VLab Redmine ticket initially opened.

4.2.3 Decline of a Community Contribution

NCEP has full authority over what is accepted into the operational UPP repository. They may, for any reason, decide to not accept a community contribution. An explanation of refusal shall be included in the notification email sent to the DTC. Any explanation of why the contribution was declined, including possible changes which would make the modification acceptable, will be supplied to the DC. The VLab Redmine issue tracking system will be updated to identify the contribution as existing in the community UPP repository only (submitted and declined). The compile flag COMMCODE may be used to allow code to exist in the DTC repository while EMC is not ready or willing to accept the change.

5 Community UPP Release and Support

The DTC will be responsible for regular code releases to the community. The DTC will participate in pre-release testing and evaluation that will go beyond the limited regression tests used for on-going maintenance. The release procedure will consist of:

- The community UPP *DTC_post* branch of the *EMC_post* repository will be placed in a "frozen" state.
- During the "freeze" period the DC will only review bug fix contributions.
- Regular release meetings will be held during the freeze period. At the first release meeting, the additions/modifications available in the up-coming community UPP software package will be determined and documented. This document will be made available before or in conjunction with the release.

Community UPP users can utilize the helpdesk email account upp-help@ucar.edu to submit questions regarding any aspect of UPP. The DTC will serve as the frontline for this task. Although the DTC will be responsible for responding to user inquiries, developers will need to assist with inquiries that go beyond the expertise of the DTC staff. Thus, once any new feature is released, the owners of those additions will automatically become part of the support team (helpdesk) who may need to answer questions from users. Code contributors will also be responsible for supplying pertinent documentation upon submission of any new code. The DTC will be responsible for reviewing and updating official UPP documentation with each new release.

Appendix A. CODE PRACTICES

PREAMBLE

The reason for putting this document together, apart from establishing some minimum standard for code quality from developers outside EMC or JCSDA, is to provide a basis for consistency amongst the many UPP developers.

One thing to remember other people will be reading and trying to understand your code – be nice to them.

STYLE

- Use free format syntax.
- Indentation: begin in first column for statements such as `PROGRAM`, `MODULE` and `CONTAINS`, and recursively indent all subsequent blocks by *at least* two spaces.
- Do not use tab characters – they are not part of the Fortran character set.
- Name `END`s fully, including the program unit name.
- When creating new code (this includes refactoring[1] old code), use the style guidelines above within the context of your personal style. If you use a syntax sensitive editor, as an experiment, turn off the syntax coloring to see if your code is still easily readable.
- When modifying old code, adhere to the style of the existing code.

COMMENTS/DOCUMENTATION

- For cryptic variable names, state description in a comment immediately preceding declaration or on end of the declaration line.
- For procedures and modules, insert a contiguous documentation block immediately following its declaration containing a *brief* overview followed by an optional detailed description.
- Ensure procedure argument documentation in the doc block is consistent with additions and/or deletions from the calling list.
- Procedure argument documentation in the doc block should briefly describe what are the arguments and their units. In some cases, this level of documentation may be unnecessary (e.g. the arguments to a generic interpolation procedure.) If in doubt, err on the side of documenting the argument list.

[1] Refactoring involves improving the design of existing code. It doesn't change the observable behavior of the software; it improves its internal structure. Refactoring does not fix bugs or add new functionality. See http://en.wikipedia.org/wiki/Refactoring or Fowler,M., "Refactoring", 2000, Addison-Wesley.

- Document any modifications made by using a short, but descriptive, log message when checking the modified code into the software repository. Don't just say *what* has changed – since differencing versions provides that information – but *why*.
- Do not document changes within the code with comments that include the user's name or initials.

VARIABLE DECLARATIONS

- Declare all variables (`IMPLICIT NONE`)
- Use meaningful, understandable names for variables and parameters.
- Do not use Fortran intrinsic function names for variable names.
- Declare `INTENT` on all dummy arguments.
- Declare `DIMENSION` attribute for all non-scalars.
- Line up attributes within variable declaration blocks.
- Any scalars used to define extent must be declared prior to use.
- Declare a variable name only once in a scope, including `USE MODULE` statements.

MODULES

- Use modules to group related procedures and/or shared data.
- Use the `ONLY` attribute on `USE` statements as required.
- Declare `IMPLICIT NONE`.
- Include a `PRIVATE` statement and explicitly declare public attributes.

SUBROUTINES AND FUNCTIONS

- Group all dummy argument declarations first, followed by local variable declarations.
- Declare `INTENT` on all dummy arguments.
- To avoid null or undefined pointers, pointers passed through an argument list must be allocated.

CONTROL CONSTRUCTS

- Name control constructs (e.g., `DO`, `IF`, `SELECT CASE`) which span a significant number of lines or form nested code blocks.
- No numbered do-loops.
- Name loops that contain `CYCLE` or `EXIT` statements.
- Do not use `GOTO`.
- Use Fortran95-style relational symbols, e.g., `>=` rather than `.GE.`, `/=` rather than `.NE.`.

- For multiple selection tests, use case statements with case defaults rather than if-constructs wherever possible.

MISCELLANEOUS

- Always initialize pointer variables in their declaration statement using the `NULL()` intrinsic, e.g.

```
INTEGER, POINTER :: x => NULL()
```

- Use modules (not common blocks) for sharing large segments of data.
- Remove unused variables.
- Do not use, *if at all possible*, compiler specific functions or calls. Doing so limits portability of the code. If compiler specific functions or calls must be used, localize the impact by wrapping the compiler extensions within a generic procedure and call that generic procedure. Document the potential portability problem in the calling code.
- Always use generic, not specific, intrinsic functions, e.g. `COS` rather than `DCOS`.
- Remove code that was used for debugging purposes once the debugging is complete.
- A standard naming convention has been adopted for variables and routines which refer to tangent linear and adjoint values, use suffixes "_tl" and "_ad" respectively.

APPENDIX A WAS TAKEN FROM THE BOULDER COMMUNITY GSI CODE MANAGEMENT PLAN (2009)

Appendix B. REGRESSION TESTING

A suite of regression tests will be made available for users to run while doing development and prior to requests for committing system modifications back into the community UPP repository. These regression tests will be broken down into code functionality, allowing users to better match their modifications to an appropriate test. To aid developers in understanding the available tests and their application a brief explanation of the software architecture has been included here.

The UPP begins by reading in a file named ITAG. This file specifies the input filename, model, and input data format. There is other information in this file that will not be discussed here. Based on the model and input format, the input data is read. There is minimal calculation done at this point. The code returns to a central flow to read in the control file (***wrf_cntrl.parm or postxconfig-NT.txt***) which specifies which outputs are requested. The computation of the output variables begins at this point. This processing is broken into parameter type. Output of the data occurs as the data is processed. Figure 3. UPP Code Process Flo shows the UPP code process flow.

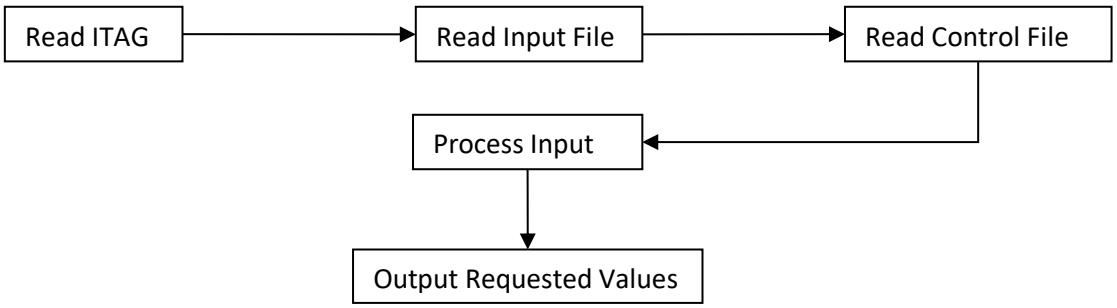

Figure 3. UPP Code Process Flow

Table 2. UPP Input Types shows the actual subroutines called to process each model and input type. UPP is currently set up to ingest model output from WRF in NetCDF format and FV3GFS in NEMSIO binary format. They are binary files with WRF and NEMSIO headers respectively to identify what model variables each record represents. UPP also has two options to read WRF binary files, the sequential read in INITPOST*BIN and record independent MPI IO read in INITPOST*MPIIO. When a user makes a modification based on an input type the modifications should be propagated to all routines which are affected. When testing, all routines should be verified to ascertain that their functionality has not been compromised.

Model	Input Format	UPP "itag" input value	Read Routine	
ARW / RAPR/HRRR	netCDF	netcdf	INITPOST	
ARW/RAPR	WRF binary	binarympiio	INITPOST_BIN_MPIIO	Deprecated with V4.0
NMM	netCDF	netcdf	INITPOST_NMM	
NMM	WRF binary	binarympiio	INITPOST_NMM_BIN_MPIIO	Deprecated with V4.0
GFS	Grib	grib	INITPOST_GFS	Deprecated with V4.0
GFS	NEMSIO binary	binarynemsio	INITPOST_GFS_NEMS	
NMM	NEMSIO binary	binarynemsio	INITPOST_NEMS	Deprecated with V4.0
FV3GFS	NEMSIO binary	binarynemsiompiio	INITPOST_GFS_NEMS_MPIIO	

Table 2. UPP Input Types

Upon successfully reading the input into the UPP data structures, processing to compute desired outputs occurs. Table 3. UPP Calculation Routines outlines a high-level representation of where variables are calculated and output.

Process Routine	Description
MDLFLD	Called first; initializes values used throughout / computes model surface values
MDL2P	Interpolates model data to pressure surfaces
MDLSIGMA	Additional interpolation of model data
MDL2SIGMA2	Additional interpolation of model data
MDL2AGL	Interpolates model data to AGL height surfaces
SURFCE	Handles surface based fields
CLDRAD	Handles sounding, cloud related, and model posted radiation fields
MISCLN	Handles TPAUSE level Z,P,T,U,V and vertical shear; max wind level Z,P, U and V; FD level T, Q, U and V; Freezing level Z and RH; constant mass (boundary) fields, LFM look-alike fields; NGM look-alike fields
FIXED	Handles time independent "fixed" fields
MDL2THANDPV	Interpolates model data to THETA and P surfaces
CALRAD_WCLOUD	Derives model "brightness" T using CRTM

Table 3. UPP Calculation Routines

Regression tests will be created to test capabilities based on model, input format, and type of parameter calculated. Note that testing related to WRF-NMM NetCDF input format is managed by the HWRF group. Table 4. Supported Platforms/Compilers/Data shows the types of models, data formats, platforms, and compilers that require testing.

		NetCDF		binarynemsiompiio	
		ARW	**FV3GFS**	**ARW**	**FV3GFS**
Linux - ifort	serial	X	N/A	N/A	N/A
	dmpar	X	N/A	N/A	X
Linux – GNU	serial	X	N/A	N/A	N/A
	dmpar	X	N/A	N/A	X
Linux - PGI	serial	X	N/A	N/A	N/A
	dmpar	X	N/A	N/A	X

Table 4. Supported Platforms/Compilers/Data