Unified Post Processor Users Guide

Release

May 10, 2024

CONTENTS

CHAPTER

ONE

BACKGROUND INFORMATION

1.1 Introduction

The Unified Post Processor (UPP) is a software package designed to generate useful products from raw model output.

The UPP is currently used in operations with the Global Forecast System (*GFS*), GFS Ensemble Forecast System (GEFS), North American Mesoscale (*NAM*), Rapid Refresh (*RAP*), High Resolution Rapid Refresh (HRRR), Short Range Ensemble Forecast (SREF), and Hurricane WRF (HWRF) applications. It is also used in the Unified Forecast System (*UFS*), including the Rapid Refresh Forecast System (*RRFS*), Hurricane Analysis and Forecast System (HAFS), and the Medium-Range Weather (*MRW*) and Short-Range Weather (*SRW*) Applications.

This software package can be run inline—built as a library to be used by the model—and offline—built standalone and run separately from the model. This documentation primarily details the offline procedures.

1.1.1 Terms of Use Notice

The UPP Terms of Use Notice is available at: https://github.com/NOAA-EMC/UPP/wiki/UPP-Terms-of-Use-Notice

1.2 Technical Overview

The Unified Post Processor (UPP) is used to post-process model forecasts. It can compute a variety of diagnostic fields and interpolate output from the model's native grids to National Weather Service (*NWS*) standard levels (e.g., pressure, height) and standard output grids (e.g., AWIPS, Lambert Conformal, polar-stereographic) in NWS and World Meteorological Organization (WMO) GRIB2 format. There is also an option to output fields on the model's native vertical levels. Output from the UPP can be used directly by visualization, plotting, or verification packages or used in further downstream post-processing (e.g., statistical post-processing techniques).

HIGHLIGHTS

The UPP reads and processes *GFS* and *LAM* (Limited Area Model) data from the *FV3 dynamical core* and generates output in *GRIB2* format. It uses *MPI* parallel code and produces hundreds of products like those used operationally on the same operational grids. Examples of UPP products include:

- T, Z, humidity, wind, cloud water, cloud ice, rain, and snow on pressure levels
- SLP, shelter-level T, humidity, and wind fields
- Precipitation-related fields
- PBL-related fields
- Severe weather products (e.g. CAPE, Vorticity, Wind shear)
- Radiative/Surface fluxes

- Cloud-related fields
- Aviation products
- Radar reflectivity products
- Satellite look-alike products

A full list of fields that can be generated by the UPP is provided in ../tables/UPP_GRIB2_Table_byID.

As of v11.0.0, the UPP has 2D decomposition capabilities and is also backwards compatible for 1D decomposition. The functionality demonstrates runtime improvements, especially for larger domains. Support for this feature is available for standalone UPP applications.

Support for the community UPP is provided through GitHub Discussions.

SYNTHETIC SATELLITE PRODUCTS

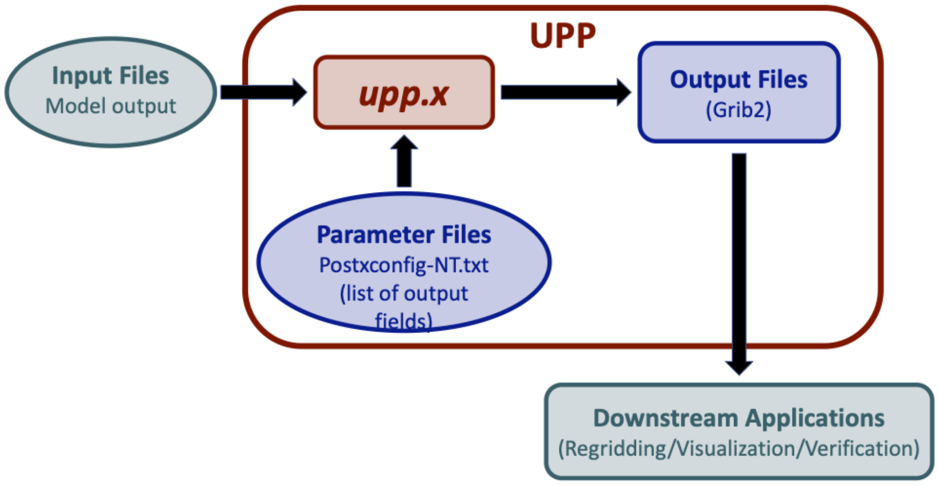
The UPP also incorporates the Joint Center for Satellite Data Assimilation (*JCSDA*) Community Radiative Transfer Model (*CRTM*) to compute model-derived brightness temperature (TB) for various instruments and channels. This additional feature enables the generation of simulated satellite products such as:

- Geostationary Operational Environmental Satellites (GOES)
- Advanced Microwave Scanning Radiometer (AMSRE)
- Special Sensor Microwave Imager/Sounder (SSMIS)

For CRTM documentation, refer to the CRTM User Guide.

1.2.1 System Architecture Overview

The basic components of the *offline UPP* are shown in the schematic below.



The `upp.x` component performs the bulk of the post-processing. Its functions include:

- Vertical interpolation from model levels/surfaces to isobaric, height, and other levels/surfaces
- Computing diagnostics from model data such as CAPE, relative humidity, radar reflectivities, etc.

1.2.2 UPP Directory Structure

The main UPP repository is named `UPP`; it is available on GitHub at https://github.com/NOAA-EMC/UPP. Under the main **UPP** directory reside the following relevant subdirectories (the * indicates a directory that exists only after the build is complete):

exec*: Contains the `upp.x` executable after successful compilation

modulefiles: Contains modulefiles for specific platforms and compilers for building on preconfigured machines.

parm: Contains parameter files, which can be modified by the user to control how the post processing is performed.

scripts: Contains a sample run script to process fv3 history files.
- `run_upp`: Script that runs the standalone UPP package (`upp.x`)

sorc: Contains source code for:
- `ncep_post.fd`: Source code for the UPP

tests: Contains the scripts used to install UPP
- `compile_upp.sh`: UPP build script
- `build*`: Contains the UPP build
- `install*`: Contains the installed executable (`bin/upp.x`), modules and libraries

When the `develop` branch of the UPP repository is cloned, the basic directory structure follows the example below. Some files and directories have been removed for brevity.

```
UPP
├── ci                          -------- Automated testing files
├── cmake                       -------- CMake build files
├── docs                        -------- User's Guide files
│   └── Doxyfile.in             -------- Doxygen configuration file
├── exec*
├── fix
├── jobs                        -------- Scripts that set up the environment and call␣
↪ex-scripts from the scripts directory
├── modulefiles
├── parm
│       ├── post_avblflds.xml   -------- List of all fields available in UPP
│       ├── postcntrl*.xml      -------- User-editable control files that list the␣
↪variables to be output
│       └── postxconfig-NT-*.txt    -------- Text file of requested output that UPP reads␣
↪(processed from postcntrl)
├── scripts
│       └── run_upp             -------- Script that runs the stand-alone UPP package␣
↪(upp.x)
```

(continues on next page)

(continued from previous page)

```
├── sorc
│   ├── libIFI.fd            -------- Private repository (submodule) for in-flight␣
↪icing
│   └── ncep_post.fd         -------- Main post-processing routines
├── tests
│   ├── build*
│   ├── install*
│   └── compile_upp.sh       -------- UPP build script
├── ush                      -------- Utility scripts (referenced & run in /scripts)
├── CMakeLists.txt
├── LICENSE.md
├── README.md
└── VERSION
```

1.3 Acknowledgments

The adaptation of the original WRF Post Processor package and User's Guide (by Mike Baldwin of NSSL/CIMMS and Hui-Ya Chuang of NCEP/EMC) was done by Lígia Bernardet (NOAA/ESRL/DTC) in collaboration with Dusan Jovic (NCEP/EMC), Robert Rozumalski (COMET), Wesley Ebisuzaki (NWS/HQTR), and Louisa Nance (NCAR/RAL/DTC). Upgrades to WRF Post Processor versions 2.2 and higher were performed by Hui-Ya Chuang, Dusan Jovic, and Mathew Pyle (NCEP/EMC). Transitioning of the documentation from the WRF Post Processor to the Unified Post Processor was performed by Nicole McKee (NCEP/EMC), Hui-Ya Chuang (NCEP/EMC), and Jamie Wolff (NCAR/RAL/DTC). Implementation of the Community Unified Post Processor was performed by Tricia Slovacek, Kate Fossell, and Tracy Hertneky (NCAR/RAL/DTC). Currently, community user support is provided by the Earth Prediction Innovation Center (EPIC) UPP team.

Acknowledging the UPP Team:

If significant help was provided via the UPP helpdesk for work resulting in a publication, please acknowledge the EPIC UPP team.

For referencing this document please use:

UPP User's Guide V11.0.0, 24 pp.

CHAPTER

TWO

BUILDING, RUNNING, AND TESTING THE UPP

2.1 UPP Inputs and Outputs

This section describes the input files used to run the UPP and the resulting output files.

2.1.1 Input Files

The UPP requires the following input files:

- The model forecast file
- The `itag` namelist file
- The *GRIB2* control file (e.g., `postxconfig-NT.txt`)
- Additional data files (e.g., lookup tables, coefficient files for satellites)

Model Forecast

The UPP ingests FV3 *write component* files in parallel *netCDF* format.

The table below is a list of the unified model variables available from the *FV3* model core. Whether a specific variable is able to be read by the UPP relies on dependencies such as physics options and choice of model. This table does not include variables that are diagnosed when running the UPP.

UFS Unified Model Variables

- ../tables/UFS_unified_variables_table

ITAG

The file called `itag` is a Fortran namelist file that contains two sections: `&model_inputs` and `&nampgb`. It is read in by the `upp.x` executable from standard input (stdin – unit 5). Most UFS applications generate it automatically based on user-defined application-level options. All UPP namelist choices are described here.

&model_inputs

The `&model_inputs` section parameterizes choices about the set of model output files that will be used for the UPP.

Table 1: *Description of the &model_inputs namelist section.*

Variable Name	Description	Data Type	Default Value
`datestr`	Time stamp being processed (e.g., 2022-08-02_19:00:00).	character(len=19)	n/a
`filename`	Name of input dynamics file; name of full 3-D model output file.	character(len=256)	n/a
`filenameflat`	Input configuration text file defining the requested fields.	character(len=256)	postxconfig-NT.txt
`filenameflux`	Name of input physics file; name of 2-D model output file with physics and surface fields.	character(len=256)	n/a
`grib`	Grib type (Note that UPP only supports Grib2 currently)	character(5)	grib2
`ioform`	Input file format. Choices: binarynemsio or netcdf	character(len=20)	n/a
`modelname`	Model name used by UPP internally (e.g., FV3R for RRFS, 3DRTMA, HAFS; GFS for GFS and GEFS; RAPR for RAP and HRRR; NMM for NAM)	character(len=4)	n/a

&nampgb

The `&nampgb` section parameterizes choices concerning the processing done in UPP.

Table 2: *Description of the &nampgb namelist section.*

Variable Name	Description	Data Type	Default Value
`aqf_on`	Turn on Air Quality Forecasting (CMAQ-based)	logical	.false.
`d2d_chem`	Turn on option to process the 2D aerosol/chemical tracers	logical	.false.
`d3d_on`	Turn on option to use dynamic 3D fields from GFS	logical	.false.
`filenameaer`	aerosols file name	character(len=256)	""
`gccpp_on`	Turn on option to process the aerosol/chemical tracers related output from UFS-Chem (CCPP-Chem) model	logical	.false.
`gocart_on`	Turn on option to process the aerosol/chemical tracers related output from GEFS-Aerosols model (GOCART)	logical	.false.
`gtg_on`	Turn on GTG (Graphical Turbulence Guidance)	logical	.false.
`hyb_sigp`	Not used	logical	.true.
`kpo`	The number of pressure levels, if different than standard one specified by `SPLDEF` described below.	integer	0
`kpv`	The number of output potential vorticity levels	integer	8
`kth`	The number of output isentropic levels	integer	6
`method_blsn`	Turn on blowing snow effect on visibility diagnostic (default=true)	logical	.true.
`nasa_on`	Turn on option to process the aerosol/chemical tracers related output from UFS-Aerosols model (NASA GOCART)	logical	.false.
`numx`	The number of i regions in a 2D decomposition; Each i row is distibuted to numx ranks; numx=1 is the special case of a 1D decomposition in Y only.	integer	1
`po`	List indicating pressure levels in output	real,dimension(70)	0
`popascal`	Switch to indicate if pressure levels are in pascals (multply by 100 if false)	logical	.false.
`pv`	List indicating the potential vorticity level output	real,dimension(70)	(/0.5,-0.5,1.0,-1.0,1.5,-1.5,2.0,-2.0,(0.,k=kpv+1,70)/)
`rdaod`	Turn on the option to process the AOD from the GFS scheme	logical	.false.
`slrutah_on`	Calculate snow to liquid ratio (SLR) using method from University of Utah.(default=false)	logical	.false.
`th`	List indicating isentropic level output	real,dimension(70)	(/310.,320.,350.,450.,550.,650.,(0.
`vtimeunits`	valid time units, default="", Choices: FMIN	character(len=4)	""
`write_ifi_debug_files`	Write debug files for In-Flight Icing (IFI), a restricted option in UPP	logical	.false.

Control File

The user interacts with the UPP through the control file to define what fields and levels to output. It is composed of a header and a body. The header specifies the output file information. The body includes which fields and levels to process.

A default control file, `postxconfig-NT.txt`, is provided and read by the UPP. Users who wish to customize the control file to add or remove fields and/or levels may do so by modifying `postcntrl.xml` and then remaking the text file as described in the later section: *Creating the Flat Text File*.

Note: The control file names `postxconfig-NT.txt` and `postcntrl.xml` are generic names and are different depending on the application used. Control files for various operational models are located in the `UPP/parm` directory.

Selecting Which Variables the UPP Outputs

To output a field, the body of the control file needs to contain an entry for the appropriate variable. If an entry for a particular field is not yet available in the control file, it may be added to the control file with the appropriate entries for that field. For variables found on vertical levels (e.g., isobaric or height levels), the desired levels to be output must be listed (see next section: *Controlling which levels the UPP outputs*). A list of available GRIB2 fields that can be output by UPP can be found in the table ../tables/UPP_GRIB2_Table_byID. Please note that some fields are dependent on model, physics, and other fields.

Controlling which levels the UPP outputs

The `<level>` tag in the `postcntrl.xml` file is used to list the desired levels for output. The following levels are currently available for output:

- For isobaric output, 46 levels are possible, from 2 to 1000 hPa (*2, 5, 7, 10, 20, 30, 50, 70 mb and then every 25 mb from 75 to 1000 mb*). The complete list of levels is specified in `sorc/ncep_post.fd/CTLBLK.f`.
  - Modify specification of variable `LSMDEF` to change the number of pressure levels: LSMDEF=47
  - Modify specification of `SPLDEF` array to change the values of pressure levels: (/200.,500.,700.,1000.,2000.,3000.,5000.,7000.,7500.,10000.,12500.,15000.,17500.,20000., …/)
- For model-level output, all model levels are possible, from the highest to the lowest.
- When using the Noah LSM, the soil layers are 0-10 cm, 10-40 cm, 40-100 cm, and 100-200 cm.
- When using the RUC LSM, the soil levels are 0 cm, 1 cm, 4 cm, 10 cm, 30 cm, 60 cm, 100 cm, 160 cm, and 300 cm. (For the old RUC LSM, there are only 6 layers, and if using this, you will need to change `NSOIL` for "RUC LSM" from 9 to 6 in the `sorc/ncep_post.fd/WRFPOST.f` routine.)
- When using Pliem-Xiu LSM, there are two layers: 0-1 cm, 1-100 cm
- For low, mid, and high cloud layers, the layers are $\geq$642 hPa, $\geq$350 hPa, and <350 hPa, respectively.
- For PBL layer averages, the levels correspond to 6 layers with a thickness of 30 hPa each.
- For flight level, the levels are 30 m, 50 m, 80 m, 100 m, 305 m, 457 m, 610 m, 914 m, 1524 m, 1829 m, 2134 m, 2743 m, 3658 m, 4572 m, 6000 m, 7010 m.
- For AGL radar reflectivity, the levels are 4000 and 1000 m.
- For surface or shelter-level output, the `<level>` is not necessary.

Creating the Flat Text File

If the control file requires any modifications, a preprocessing step will be required by the user to convert the modified XML file `parm/postcntrl.xml` to a flat text file `parm/postxconfig-NT.txt`. The user will first need to edit the `postcntrl.xml` file to declare which fields are to be output from the UPP.

In order to ensure that the user-edited XML files are error free, XML stylesheets (`parm/EMC_POST_CTRL_Schema.xsd` and `EMC_POST_Avblflds_Schema.xsd`) can be used to validate both the `postcntrl.xml` and `post_avblflds.xml` files respectively. Confirmation of validation will be given (e.g., `postcntrl.xml` validates) or otherwise return errors if it does not match the schema. This step is optional, but acts as a safeguard to avoid run-time failures with the UPP. To run the validation:

```
xmllint --noout --schema EMC_POST_CTRL_Schema.xsd postcntrl.xml
xmllint --noout --schema EMC_POST_Avblflds_Schema.xsd post_avblflds.xml
```

Once the XMLs are validated, the user will need to generate the flat file. The command below will run the Perl program `parm/PostXMLPreprocessor.pl` to generate the post flat file:

```
/usr/bin/perl PostXMLPreprocessor.pl your_user_defined_xml post_avblflds.xml your_user_
↪defined_flat
```

where `your_user_defined_xml` is your modified XML and `your_user_defined_flat` is the output text file.

2.1.2 Output Files

Upon a successful run, `upp.x` will generate GRIB2 output files in the post processor working directory. These files will include all fields that were requested in the control file.

When running UPP standalone, the following GRIB2 output files will be generated:

GFS Model: `GFSPRS.HHH`
LAM (Limited Area Model): `NATLEV.HHH` and `PRSLEV.HHH`

When executed with the provided run script, UPP provides log files in the post-processor working directory named `upp.fHHH.out`, where `HHH` is the forecast hour. These log files may be consulted for further runtime information in the event of an error.

2.2 Building UPP Stand-Alone

The UPP uses a CMake-based build system to integrate all the required components for building the UPP. Once built, the UPP can be run standalone (outside the *UFS* Applications) to post-process model output.

2.2.1 Software Requirements

The UPP is tested on a variety of research platforms, including NOAA HPC systems (e.g., Hera, Orion). These supported platforms are preconfigured for building and running the UPP and already have the required libraries available via spack-stack in a centralized location. The *spack-stack* is a *Spack*-based method for installing UFS prerequisite software libraries.

Users working on unsupported platforms will need to install spack-stack on their system and can do so following the instructions in the spack-stack User's Guide.

Note: Users can install *HPC-Stack* instead of spack-spack by following the instructions in the HPC-Stack User's Guide. However, support for HPC-Stack is being deprecated, and limited assistance is available for use of HPC-Stack with the UPP.

Common Modules

As of February 1, 2024, the UPP uses the following common modules from spack-stack:

```
cmake 3.16.1+
hdf5/1.14.0
netcdf-c 4.9.2
netcdf-fortran 4.6.1
jasper 2.0.32
libpng 1.6.37 / png 1.6.35
zlib 1.2.13
g2 3.4.5
g2tmpl 1.10.2
parallelio 2.5.10
bacio 2.4.1
ip 4.3.0
sp 2.5.0
crtm 2.4.0.1
w3emc 2.10.0
nemsio 2.5.4
sigio 2.3.2
sfcio 1.4.1
wrf_io 1.2.0
```

Individual machines may subsequently load slightly different versions. The most updated list of modules for a given machine can be viewed in each machine's modulefile. Users on non-Tier-1 systems should look at the modulefile for the system whose architecture most closely resembles their own system's architecture to determine which modules they may need.

2.2.2 Obtaining and Installing UPP

Building and running UPP v11.0.0 has been tested and is supported on the following pre-configured platforms.

System	Compiler and Version
NOAA Hera	Intel 18.0.5.274
NOAA Orion	Intel 2018.4

To install the UPP, navigate to the directory where you want to install UPP and clone the repository.

```
git clone -b branch-or-tag-name https://github.com/NOAA-EMC/UPP
```

where, `branch-or-tag-name` is the release branch or tag you wish to clone (e.g., `upp_v11.0.0`). (Leaving off the `-b` argument will clone all branches of the repository.)

Move to the directory with the build script and build the UPP.

```
cd UPP/tests

./compile_upp.sh
```

Note: To build in debug mode, you can add `-DCMAKE_BUILD_TYPE=Debug` to the `cmake_opts` parameter in the `compile_upp.sh` script. This removes compiler optimization flags and adds `-g` to the Fortran compilation. You can also use `-DCMAKE_BUILD_TYPE=RELWITHDEBINFO`, which gives the `-g`, but keeps the `-O2` optimization for the Fortran compilation.

Move back to the top-level UPP directory and create a directory where the CRTM fix files will be unpacked. Download the fix files from the GitHub release page or use the `wget` command. Unpack the tar file.

```
cd ../
mkdir crtm && cd crtm
wget https://github.com/NOAA-EMC/UPP/releases/download/upp_v11.0.0/fix.tar.gz
tar -xzf fix.tar.gz
```

Note: To make a clean build, simply remove both the `tests/build` and `tests/install` directories and the `exec/upp.x` executable and then rerun the `compile_upp.sh` script. This is recommended if a mistake is made during the installation process.

2.3 Running UPP Stand-Alone

A script (`run_upp`) for running the UPP package is included in the `/scripts` directory.

Before running the script, perform the following instructions:

1. `cd` to your `DOMAINPATH` directory. This is the top working directory for the forecast run.
2. Make a directory to put the UPP results in.

   ```
   mkdir postprd
   ```

3. Make a directory for staging a copy of the desired control file.

   ```
   mkdir parm
   ```

4. Optional: If desired, edit the control XML file(s) in `/UPP/parm` to reflect the fields and levels you want UPP to output. It is recommended that you make copies of the original beforehand.

   GFS XMLs: `postcntrl_gfs_f00.xml` (0-hour lead time) and `postcntrl_gfs.xml` (all other lead times)
   LAM (Limited Area Model) XML: `fv3lam.xml`

   Remake the flat text file(s) following the steps in the "Control File: Creating the Flat Text File" section.

5. Copy the flat text file(s) to the `/parm` directory in your `DOMAINPATH`. These are the files that UPP reads directly.

GFS text files: `postxconfig-NT-GFS-F00.txt` (0-hour lead time) and `postxconfig-NT-GFS.txt` (all other lead times).

LAM text file: `postxconfig-NT-fv3lam.txt`

6. Copy the `/scripts/run_upp` script to the `/postprd` directory.
7. Edit the run script as outlined in the *"Run Script Overview"* section below. Once these directories are set up and the edits outlined below are complete, the script can be run interactively from the `/postprd` directory by simply typing the script name on the command line.

2.3.1 Run Script Overview

Note: It is recommended that the user refer to the `run_upp` script while reading this overview. All user-modified variables are contained at the top of the `run_upp` script in the user-edit section, along with a brief description. Descriptions below follow the `run_upp` script.

1. Set up basic path variables:
   - `TOP_DIR`: Top level directory for building and running UPP
   - `DOMAINPATH`: Working directory for this run
   - `UPP_HOME`: Location of the **UPP** directory
   - `POSTEXEC`: Location of the **UPP** executable
   - `modelDataPath`: Location of the model output data files to be processed by the UPP
   - `txtCntrlFile`: Name and location of the flat text file that lists desired fields for output.

   Note: For FV3, the scripts are configured such that UPP expects the flat text file to be in `/parm`, and the post-processor working directory to be called `/postprd`, all under `DOMAINPATH`. This setup is for user convenience to have a script ready to run; paths may be modified, but be sure to check the run script to make sure settings are correct.

2. Specify dynamical core being run:
   - `model`: Which model is used? ("GFS" or "LAM" - Limited Area Model)
3. Specify the format for the input model files and output UPP files:
   - `inFormat`: Format of the model data ("netcdfpara")
   - `outFormat`: Format of output from UPP ("grib2")
4. Specify the forecast cycles to be post-processed:
   - `startdate`: Forecast start date (YYYYMMDDHH)
   - `fhr`: First forecast hour to be post-processed
   - `lastfhr`: Last forecast hour to be post-processed
   - `incrementhr`: Increment (in hours) between forecast files

Attention: Do not set `incrementhr` to 0 or the script will loop continuously!

5. Set/uncomment the run command for your system (e.g., `mpirun`).

   - `RUN_COMMAND`: System run commands

     - The default execution command in the distributed scripts is for a single processor:

       `./upp.x > upp.${fhr}.out 2>&1`

     - To run UPP using *MPI* (dmpar compilation), the command line should be:

       >> LINUX-MPI systems: `mpirun -np N upp.x > outpost 2>&1`

       (Note: On some systems a host file also needs to be specified: `-machinefile "host"`)

       >> IBM: `mpirun.lsf upp.x < itag > outpost`

       >> SGI MPT: `mpiexec_mpt upp.x < itag > outpost`

6. Set the value for `numx`.

   - `numx`: The number of subdomains in the x-direction used for decomposition.

     - For 1D decomposition, set numx=1 (default)
     - For 2D decomposition, set numx>1

7. Set naming convention for prefix and extension of output file name.

   - `comsp` is the initial string of the output file name. By default, it is not set, and the prefix of the output file will be the string set in the `postcntrl.xml` file `DATSET` parameter. If set, it will concatenate the setting to the front of the string specified in the XML file `DATSET` parameter.
   - `tmmark` is used for the file extension (in `run_upp`, `tmmark=tm00`; if not set, it is set to `.GrbF`)

Upon a successful run, UPP will generate output files for each forecast hour in the `/postprd` directory.

When executed with the provided run script, UPP provides log files in the post-processor working directory named `upp.fHHH.out`, where `HHH` is the forecast hour. These log files may be consulted for further runtime information in the event of an error.

2.4 Testing the UPP

2.4.1 Running UPP Regression Tests

To run the full regression test (RT) suite in preparation for opening a pull request (PR):

1. Navigate to the local clone of your UPP fork containing the changes you would like to introduce, and run the included RT script within `/ci`

```
cd /path/to/UPP/ci
nohup ./rt.sh -a <my_account> -r $PWD/rundir -t $PWD/../ &
```

   where `my_account` is the name of an account where you have permissions to run jobs. The terminal will print a message like:

```
nohup: ignoring input and appending output to ‘nohup.out’
```

The user can continue to issue commands in the Terminal while the RTs run in the background.

Note: The time it takes for tests to run is queue-dependent. RTs can take as little as half an hour to run, but on machines with long queue times, it can take several hours to complete the full set of tests.

1. Check `rt.log.<machine>/nohup.out` for a short summary of any changes in results. The tests are finished when there are 16 timestamps and a final results summary (e.g., "No changes in test results detected.").
   - The `/work` directory generated in `UPP/ci` contains `out.post.<test_name>` files, which list output from each test, including any unexpected errors during runtime.
   - The `/rundir` directory generated within `UPP/ci` will include test case results, and `.diff` files located within each test's directory will outline changes in fields with the current baselines.
   - Confirm expected changes within the run directory `.diff` files if any are present.
     - Changes in the `rap_pe_test` case only consisting of field 708 Convective Cloud Layer may be ignored; this is a known bug and will always be present within the `WRFPRS.diff` file.

2.4.2 Additional Configuration

For repeated regression test runs, users can edit the `rt.sh` file and disable the specified test cases by changing their respective values to "no." Users can disable the build step as well with the same value for the build variable above the tests. Please be sure to enable all test cases and build settings and conduct a full RT run in preparation for a pull request so that code managers (CMs) can confirm all changes in results are expected and consistent with the developer's results.

`rt.sh` will allow for changing the configuration of the regression tests if users desire to do so with the following available options:

- `w` – specify the work directory for test case job output
- `r` – specify the run directory containing baselines and `.diff` files for comparison of changes in results

The following are legacy options for when `rt.sh` was not included within the UPP repository and may be ignored by developers: `-b`, `-u`, `-c`, `-t`.

CHAPTER

THREE

CUSTOMIZING THE UPP

3.1 Adding a New Variable

This chapter provides general procedures and an example of how to add a new variable to the UPP code. Please keep in mind that it may not be an exhaustive step-by-step process depending on your particular situation. While we can provide general assistance for adding a new variable, users should be aware that this requires good knowledge of Fortran and a thorough understanding of the code.

NOAA UPP developers who wish to add new variables to the UPP will need to:

1. Read and follow procedures on the UPP wiki page on how to contribute your code changes to the UPP main development branch. Doing so will ensure your changes are merged to the UPP development branch quickly.
2. Submit your pull request with small incremental changes. Advantages of doing this include avoiding conflicts with other UPP developers in terms of using the UPP internal index and variables.
3. Please do not modify existing algorithms without coordinating with UPP code managers (Wen Meng and Hui-Ya Chuang). UPP supports many NOAA operational models, and we cannot change operational products without coordination and advanced notice.

We encourage non-NOAA UPP developers to contact EPIC via GitHub Discussions to make them aware of modifications you are making. In some cases, if they determine the changes you are making may be relevant for operational and/or community purposes, they will be interested in incorporating your changes into the code base for support and future release. We would then work with you to make this possible.

3.1.1 Process Overview: Adding a New Variable

The following steps outline the process for adding a new variable. This description is followed by a detailed example in Section 3.1.2 below.

1. Check whether your new variable has been defined in the file `parm/post_avblflds.xml` in your UPP working directory. This file defines all available *GRIB2* fields in the UPP. Users may also check the table showing ../tables/UPP_GRIB2_Table_byID.

   A. If NO (not available in `post_avblflds.xml`), check whether your new variable has been defined in the NCEP Grib2 Table (Product Discipline and Parameter Category).

      i. If NO (not available in the NCEP Grib2 Table):

         a. NOAA users can email Andrew.Benjamin@noaa.gov with the following information for your new variable: variable definition, unit, and what Grib2 discipline and category you think this variable should belong to. Andrew will define your new variable in the NCEP Grib2 Table and inform you of the Grib2 discipline and category numbers you should use.

b. Contact Andrew to update `parm/params_grib2_tbl_new.text` with your new variable and generate a `params_grib2_tbl_new` that lists variables in alphabetical order to improve post-processing efficiency.

c. Save new `params_grib2_tbl_new.text` and `params_grib2_tbl_new` under `parm/` of your UPP working directory.

d. Non-NOAA users should coordinate through EPIC for the above three steps. Users may post a GitHub Discussions topic and tag @FernandoAndrade-NOAA and @gspetro-NOAA for directions in steps a-c.

e. Add a new entry in `post_avblflds.xml` with your new variable; then follow step B below, then step 2 and beyond. You should assign a new UPP ID for your new variable.

ii. If YES (variable is available in the NCEP Grib2 Table):

a. Add a new entry in `post_avblflds.xml` with your new variable, then follow step B below, then step 2 and beyond. You should assign a new UPP ID for your new variable.

B. If YES (variable is in `post_avblflds.xml`), then your new variable is already available in the UPP. Follow steps i) and ii), make a test UPP run, and then look for your new variable in your output. You can skip the remaining steps about modifying the source code.

i. Add a new entry in your application's control xml file (e.g., `fv3lam.xml` for the FV3LAM application, `postcntrl_gfs.xml` for the FV3GFS application). This file lets users control which variables to output from UPP for Grib2.

ii. Generate `your_user_defined_flat` file (e.g., `postxconfig-NT-fv3lam.txt` for FV3LAM application) by executing:

```
/usr/bin/perl PostXMLPreprocessor.pl your_user_defined_xml post_avblflds.
↪xml your_user_defined_flat
```

This flat file (instead of the xml file) is read in by the UPP because it is much faster to read a text file than an XML file.

2. Allocate and initialize the field in `sorc/ncep_post.fd/ALLOCATE_ALL.f`.

   This file contains the instantiation or allocation of each variable. Note that the variables are defined based on the parallel processing capability of UPP. Use an example from the file.

3. Deallocate the field in `sorc/ncep_post.fd/DEALLOCATE.f`.

   All good programmers give back their resources when they are done. Please update this routine to return your resource to the system.

4. Declare the new variable in `VRBLS2D_mod.f`, `VRBLS3D_mod.f`, or `VRBLS4D_mod.f`.

   The variable is declared in one of these module-defining files depending on its dimension.

5. Read model output if necessary using `INITPOST_NETCDF.f`.

   Check first to see if all variables needed to derive your new variable are already available in the UPP. If not, you will need to use this file (or another appropriate `INITPOST_*.f` file) for reading the model output files. The appropriate one should be chosen based on the model and the model output format.

6. Add to appropriate routine(s) for filling the new variable (e.g., `SURFCE.f`, `MDLFLD.f`, `MDL2P.f`).

   This is the place where you will derive your new variable and then fill the Grib2 array with the data to be written out later on.

7. Build or rebuild the code for changes to take effect before running your UPP run script.

3.1.2 Example Procedure: Steps for adding a new variable 'TG3'

This example adds TG3 to the UPP. TG3 is the averaged climatology of surface temperature, which the land surface models (LSMs) use to specify bottom soil temperature, where the depth of the bottom is LSM-dependent. For this example, a depth of 500cm is used.

- This example illustrates adding a new variable from GFS output that will be read into UPP and directly output into the Grib2 output files (i.e., no additional computations/calculations are needed for the field).
- Additions to each of the routines are highlighted.
- Locations of routines are in `UPP/sorc/ncep_post.fd` unless specified otherwise.
- The new variable, TG3, added in this example is found in the `gfs.t00z.sfcf006.nc` file; however, both the `gfs.t00z.sfcf006.nc` and `gfs.t00z.atmf006.nc` output files are required to run UPP for GFS.

  New variable to add:

  ```
  float tg3(time, grid_yt, grid_xt) ;
        tg3:long_name = "deep soil temperature" ;
        tg3:units = "K" ;
        tg3:missing_value = 9.99e+20 ;
        tg3:cell_methods = "time: point" ;
        tg3:output_file = "sfc" ;
  ```

1. Check whether your new variable has been defined in the file `parm/post_avblflds.xml` in your UPP working version.

   A. This variable is not available in `parm/post_avblflds.xml`.

      i. Check whether your new variable has been defined in the NCEP Grib2 Table.

         1) This variable is not defined in the NCEP Grib2 Table.

            a)-d) For the purpose of this example alone, steps a) - d) are not executed as instructed.
            Instead, manual instructions are provided here for adding to the `params_grib2_table_new` in order to create a working example.

            For this example, the variable will be added to `parm/params_grib2_tbl_new` manually. You would only do this if you had no plans to contribute your addition to the UPP `develop` branch; otherwise, follow the instructions as a NOAA or Other user in steps a) - d).

            For all current UPP output fields, the `params_grib2_table_new` lists, in order, the following attributes:

            - Discipline (https://www.nco.ncep.noaa.gov/pmb/docs/grib2/grib2_doc/grib2_table0-0.shtml)
            - Category (https://www.nco.ncep.noaa.gov/pmb/docs/grib2/grib2_doc/grib2_table4-1.shtml)
            - Parameter Number (https://www.nco.ncep.noaa.gov/pmb/docs/grib2/grib2_doc/grib2_table4-2.shtml)
            - Table information (0 for parameters from the WMO table; 1 for parameters from the local NCEP table)
            - Abbreviated Variable Name (from the parameters table)

            User Procedure

            - Add this variable as TG3.
            - TG3 is a land surface product (discipline=2)

- TG3 is a vegetation/biomass product (category=0)
- Pick an unused parameter number from the table defined by discipline=2 and category=0 (Table 4.2-0-0: https://www.nco.ncep.noaa.gov/pmb/docs/grib2/grib2_doc/grib2_table4-2-2-0.shtml). The parameter number should not be in use in Table 4.2 or the current `params_grib2_tbl_new`. In this case, the unused parameter number 251 was chosen.
- Add using the NCEP local table (table=1)
- Choose an abbreviated parameter name to describe your field (e.g., TG3)
- Add alphabetically (by variable name) to the table as:

```
2 0 251 1 TG3
```

e) **Add the new variable to** `UPP/parm/post_avblflds.xml`**, which lists all fields available for output in GRIB2 format.** This file is generally not modified unless adding a new field or modifying an existing one. Users should indicate the following variable attributes in the XML file:

- `post_avblfldidx`: the unique array index number used to store this variable. The number chosen here is just an example, and it is important to pick one that is not yet in use.
- `shortname`: name describing the variable and level type
- `pname`: the abbreviation for your variable (should match what is used in `params_grib2_tbl_new`)
- `table info`: table used if not standard WMO
- `fixed_sfc1_type`: level type
- `level`: generally only used here if it is a fixed level specific to the variable (e.g., T2m, TSOIL5m)
- `scale`: precision of data written out to Grib2 file

User procedure

- Add as:

```
<param>
  <post_avblfldidx>1063</post_avblfldidx>
  <shortname>DEEP_TSOIL_ON_DEPTH_BEL_LAND_SFC</shortname>
  <pname>TG3</pname>
  <fixed_sfc1_type>depth_bel_land_sfc</fixed_sfc1_type>
  <table_info>NCEP</table_info>
  <level>500.</level>
  <scale>3.0</scale>
</param>
```

B. Add the variable to the user-defined control file.

i. Add a new entry in your application's control XML file (e.g., `fv3lam.xml` for the FV3LAM application, `postcntrl_gfs.xml` for the `FV3GFS` application). This file lets users control which variables to output from the UPP for Grib2.

User procedure

- Add as:

```
<param>
  <shortname>DEEP_TSOIL_ON_DEPTH_BEL_LAND_SFC</shortname>
  <scale>4.0</scale>
</param>
```

ii. Generate `your_user_defined_flat` file (e.g., `postxconfig-NT-fv3lam.txt` for the FV3LAM application) by executing:

```
>> /usr/bin/perl PostXMLPreprocessor.pl your_user_defined_xml post_avblflds.
↪xml your_user_defined_flat
```

This flat file (instead of the XML file) is read in by the UPP.

2. Allocate and initialize the new variable in `ALLOCATE_ALL.f` using an example from the file. Note that the variables are defined based on the parallel processing capability of the UPP.

   User Procedure

   - TG3 is a 2-dimensional field, so allocate it in the VRBLS2D GFS section of `ALLOCATE_ALL.f` as:

   ```
   allocate(tg3(ista_2l:iend_2u,jsta_2l:jend_2u))
   ```

   - Initialize TG3 in the initialization section that comes after the allocation section you added to.

   ```
   tg3(i,j)=spval
   ```

3. Deallocate the variable to give the resources back in `DEALLOCATE.f`. Updating this routine returns your resources to the system.

   User procedure

   - Add in VRBLS2D GFS section of `DEALLOCATE.f` as:

   ```
   deallocate(tg3)
   ```

4. Declare the new variable in the appropriate file (e.g., `VRBLS2D_mod.f`, `VRBLS3D_mod.f`, or `VRBLS4D_mod.f`) depending on its dimensions.

   User procedure

   - TG3 is a 2-dimensional field, so declare it in `VRBLS2D_mod.f`.
   - Add to the GFS section as:

   ```
   tg3(:,:)
   ```

5. Read the field from the GFS model output file by adding the new variable into `INITPOST_NETCDF.f`. This file is used for reading the GFS model FV3 output files in parallel netCDF format.

   User procedure

   - Add to top section of the routine in the 'use vrbls2d' section to initiate the new variable as:

   ```
   tg3
   ```

   - Read in the new variable in the section for reading the 2D netCDF file. Look at other 2D variables, such as `hpbl`, for an example. Add as:

```
! deep soil temperature
      VarName='tg3'
      call read_netcdf_2d_para(ncid2d,ista,ista_2l,iend,iend_2u,jsta,jsta_2l,
↪jend,jend_2u, &
      spval,VarName,tg3)
```

6. Determine the appropriate routine to add the new variable to (e.g., `SURFCE.f`, `MDLFLD.f`, `MDL2P.f`). The appropriate routine will depend on what your field is. For example, if you have a new diagnostic called *foo*, and you want it interpolated to pressure levels, you would need to add it to `MDL2P.f`. If *foo* were only a surface variable, you would add it to `SURFCE.f`. If you wanted *foo* on native model levels, you would add it to `MDLFLD.f`. If you are not sure which routine to add the new variable to, choose a similar variable as a template, and add it in the same places.

   Note: This is also where you would add any calculations needed for your new variable, should they be required.

   User procedure

   - Treat TG3 like a surface field, similar to the other soil fields, and add it to `SURFCE.f`.
   - Use another 2D variable, such as 'SNOW WATER EQUIVALENT' as a template. This variable is also being read through and output, similar to what we want.
   - Add to top section in 'use vrbls2d, only' to initiate the new variable as:

```
tg3
```

   - Add in main section using a template variable as a guide.

```
! DEEP SOIL TEMPERATURE
IF ( IGET(1063).GT.0 ) THEN
  ID(1:25) = 0
  If(grib=='grib2') then
    cfld=cfld+1
    fld_info(cfld)%ifld=IAVBLFLD(IGET(1063))
!$omp parallel do private(i,j,jj)
    do j=1,jend-jsta+1
      jj = jsta+j-1
      do i=1,iend-ista+1
      ii = ista+i-1
        datapd(i,j,cfld) = TG3(ii,jj)
      enddo
    enddo
  endif
ENDIF
```

7. Build or rebuild the code for changes to take effect before running your UPP run script.

   User procedure for building on preconfigured machines:

```
>> cd UPP/tests
>> ./compile_upp.sh
```

   Assuming the modified code built successfully, and you were able to produce Grib2 output, you can check the Grib2 file for your new variable.

GRIB2 output of the new variable from this example procedure (using the wgrib2 utility if available on your system):

```
wgrib2 -V GFSPRS.006

716:37731711:vt=2019061506:500 m underground:6 hour fcst:var discipline=2␣
↪center=7 local_table=1 parmcat=0 parm=251:
    ndata=73728:undef=0:mean=278.383:min=215.47:max=302.4
    grid_template=40:winds(N/S):
    Gaussian grid: (384 x 192) units 1e-06 input WE:NS output WE:SN
    number of latitudes between pole-equator=96 #points=73728
    lat 89.284225 to -89.284225
    lon 0.000000 to 359.062500 by 0.937500
```

- For this example, since the new variable was not added to the NCEP Grib2 table, it will not be defined by the variable name. Instead it will be defined using the Grib2 parameter information entered into `params_grib2_tbl_new` from step 1 of this procedure.

3.2 Regridding

Users who wish to interpolate their UPP output to a different grid may do so with the *wgrib2* utility. The general format for regridding to various common projections are outlined in the following examples.

Wgrib2 is a versatile program that has the ability to convert grib2 files from one grid to another for various user-defined grids as well as predefined *NCEP* grids. Complete documentation with examples of regridding for all available grid definitions can be found at: https://www.cpc.ncep.noaa.gov/products/wesley/wgrib2/new_grid.html

3.2.1 Examples of wgrib2

Example 1: Latitude-Longitude Grid

`-new_grid latlon lon0:nlon:dlon lat0:nlat:dlat outfile`

Variable	Description
lon0	Longitude of first grid point in degrees
nlon	Number of longitudes
dlon	Grid resolution in degrees of longitude
lat0	Latitude of first grid point in degrees
nlat	Number of latitudes
dlat	Grid resolution in degrees of latitude

Example 2: Lambert Conic Conformal Grid

`-new_grid lambert:lov:latin1:latin2 lon0:nx:dx lat0:ny:dy outfile`

Variable	Description
lov	Longitude where y-axis is parallel to meridian in degrees
latin1	First latitude from pole which cuts the secant cone in degrees
latin2	Second latitude from pole which cuts the secant cone in degrees
lon0	Longitude of the first grid point in degrees
nx	Total number of grid points along x
dx	Grid cell size in meters in x-direction
lat0	Latitude of the first grid point in degrees
ny	Total number of grid points along y
dy	Grid cell size in meters in y-direction

Example 3: Polar Stereographic Grid

`-new_grid nps:lov:lad lon0:nx:dx lat0:ny:dy outfile` OR `-new_grid sps:lov:lad lon0:nx:dx lat0:ny:dy outfile`

Variable	Description
nps/sps	North/south polar stereographic
lov	Longitude where y-axis is parallel to meridian in degrees
lad	Latitude where dx and dy are specified
lon0	Longitude of the first grid point in degrees
nx	Total number of grid points along x
dx	Grid cell distance in meters in x-direction at *lad*
lat0	Latitude of the first grid point in degrees
ny	Total number of grid points along y
dy	Grid cell distance in meters in y-direction at *lad*

Winds

`-new_grid_winds grid` OR `-new_grid_winds earth`

Variable	Description
grid	U-wind goes from grid (i,j) to (i+1,j)
earth	U-wind goes eastward, V-wind goes northward

Interpolation

The default interpolation type is bilinear, but it can be set to another type (e.g., neighbor, budget).

`-new_grid_interpolation type`

Operational Example

Interpolates to a 0.25 degree latitude-longitude grid using various interpolation types depending on the variable.

```
wgrib2 infile -set_grib_type same -new_grid_winds earth |
-new_grid_interpolation bilinear |
-if ":(CRAIN|CICEP|CFRZR|CSNOW|ICSEV):" -new_grid_interpolation neighbor -fi |
-set_bitmap 1 -set_grib_max_bits 16 |
-if ":(APCP|ACPCP|PRATE|CPRAT):" -set_grib_max_bits 25 -fi |
-if ":(APCP|ACPCP|PRATE|CPRAT|DZDT):" -new_grid_interpolation budget -fi |
-new_grid "latlon 0:1440:0.25 90:721:-0.25" outfile
```

Note: *wgrib2* is not distributed as part of the *UFS*, but it can be installed via *spack-stack* or *HPC-Stack* along with other UFS prerequisite software. Users may also download and install it directly from https://www.cpc.ncep.noaa.gov/products/wesley/wgrib2/.

CHAPTER

FOUR

REFERENCE

4.1 Frequently Asked Questions

4.1.1 Is UPP compatible with NetCDF4?

The UPP is compatible with NetCDF4 when used on UFS model output.

4.1.2 How do I compile on another platform/compiler?

We are not able to support all platform and compiler combinations out there but will try to help with specific issues when able. Users may request support on the UPP GitHub Discussions page. We always welcome and are grateful for user-contributed configurations.

4.1.3 How can I output satellite fields with the Unified Post Processor (UPP)?

Currently, the standalone release of the UPP can be utilized to output satellite fields if desired. The UPP documentation lists the grib2 fields, including satellite fields, produced by the UPP. After selecting which fields to output, the user must *adjust the control file* according to the instructions in the UPP documentation to output the desired fields. When outputting satellite products, users should note that not all physics options are supported for outputting satellite products. Additionally, for regional runs, users must ensure that the satellite field of view overlaps some part of their domain.

Most UFS application releases do not currently support this capability, although it is available in the Short-Range Weather (SRW) Application. This SRW App pull request (PR) added the option for users to output satellite fields using the SRW App. The capability is documented in the SRW App User's Guide.

4.1.4 How do I add a new variable to UPP output?

If the desired variable is already available in the UPP code, then the user can simply add that variable to the `postcntrl.xml` file and *remake the postxconfig-NT.txt file* that the UPP reads. Please note that some variables may be dependent on the model and/or physics used.

If the desired variable is not already available in the UPP code, it can be added following the instructions for *adding a new variable* in the UPP User's Guide.

4.1.5 Why is the variable I requested not present in the UPP output?

There are a few possible reasons why a requested variable might not appear in the UPP output:

1. The variable may be dependent on the model.
2. Certain variables are dependent on the model configuration. For example, if a variable depends on a particular physics suite, it may not appear in the output when a different physics suite is used.
3. The requested variable may depend on output from a different field that was not included in the model output.

4.1.6 If the UPP fails, how do I troubleshoot the problem?

If the user suspects that the UPP failed (e.g., no UPP output was produced or console output includes an error message like `mv: cannot stat `GFSPRS.GrbF00`: No such file or directory`), the best way to diagnose the issue is to consult the UPP runtime log file for errors. When using the standalone UPP with the `run_upp` script, this log file will be located in the `postprd` directory under the name `upp.fHHH.out`, where `HHH` refers to the 3-digit forecast hour being processed. When the UPP is used with the SRW App, the UPP log files can be found in the experiment directory under `log/run_post_fHHH.log`.

4.1.7 How do I regrid UPP output to another domain or projection?

UPP output is in standard grib2 format and can be interpolated to another grid using the third-party utility wgrib2. Some basic examples can also be found in Section 3.2.

4.1.8 I am running UPP in parallel, but it fails.

This may be a memory issue; try increasing the number of CPUs or spreading them out across nodes (e.g., increase `ptiles`). We also know of one version of MPI (mpich v3.0.4) that does not work with UPP. A work-around was found by modifying the `UPP/sorc/ncep_post.fd/WRFPOST.f` routine to change all `unit 5` references (which is standard I/O) to `unit 4` instead.

4.1.9 My FV3GFS unipost output is on a Gaussian grid. How can I process it to another grid such as a lat-lon grid or other user-defined grid?

For regridding grib2 unipost output, the wgrib2 utility can be used. See complete documentation on grid specification with examples of regridding for all available grid definitions. The *Regridding section* of this UPP User's Guide also gives examples (including an example from operations) of using wgrib2 to interpolate to various common grids.

4.1.10 What does this warning mean in my compile.log? `libnemsio.a(nemsio_module_mpi.o): In function '__nemsio_module_mpi_MOD_readmpi4': nemsio_module_mpi.f90:(.text+0x1088): undefined reference to 'mpi_type_create_indexed_block_'`

This warning appears for some platforms/compilers because a call in the *nemsio* library is never used or referenced for a serial build. This is just a warning and should not hinder a successful build of UPP or negatively impact your UPP run.

4.1.11 Why do I see `** FATAL ERROR: Statistical processing bad n=0 **` when using the wgrib2 utility on my UPP output?

This error message is displayed when using more recent versions of the wgrib2 utility on files for forecast hour zero that contain accumulated or time-averaged fields. This is due to the newer versions of wgrib2 no longer allowing the `n` parameter to be zero or empty.

Users should consider using a separate control file (e.g., `postcntrl_gfs_f00.xml`) for forecast hour zero that does not include accumulated or time-averaged fields, since they are zero anyway. Users can also continue to use an older version of *wgrib2*; v2.0.4 is the latest known version that does not result in this error.

4.2 Glossary

CAPE
Convective Available Potential Energy.

CCPP
The Common Community Physics Package is a forecast-model agnostic, vetted collection of code containing atmospheric physical parameterizations and suites of parameterizations for use in Numerical Weather Prediction (*NWP*) along with a framework that connects the physics to the host forecast model.

CIN
Convective Inhibition.

CRTM
The Community Radiative Transfer Model (CRTM) is a fast and accurate radiative transfer model developed at the Joint Center for Satellite Data Assimilation (JCSDA) in the United States. It is a sensor-based radiative

transfer model and supports more than 100 sensors, including sensors on most meteorological satellites and some from other remote sensing satellites.

Component
A software element that has a clear function and interface. In Earth system models, components are often single portions of the Earth system (e.g., atmosphere, ocean, or land surface) that are assembled to form a whole.

Component Repository
A *repository* that contains, at a minimum, source code for a single component.

CONUS
Continental United States

CAM
convection-allowing models
Convection-allowing models (CAMs) are models that run on high-resolution grids (usually with grid spacing at 4km or less). They are able to resolve the effects of small-scale convective processes. They typically run several times a day to provide frequent forecasts (e.g., hourly or subhourly).

data assimilation
Data assimilation is the process of combining observations, model data, and error statistics to achieve the best estimate of the state of a system. One of the major sources of error in weather and climate forecasts is uncertainty related to the initial conditions that are used to generate future predictions. Even the most precise instruments have a small range of unavoidable measurement error, which means that tiny measurement errors (e.g., related to atmospheric conditions and instrument location) can compound over time. These small differences result in very similar forecasts in the short term (i.e., minutes, hours), but they cause widely divergent forecasts in the long term. Errors in weather and climate forecasts can also arise because models are imperfect representations of reality. Data assimilation systems seek to mitigate these problems by combining the most timely observational data with a "first guess" of the atmospheric state (usually a previous forecast) and other sources of data to provide a "best guess" analysis of the atmospheric state to start a weather or climate simulation. When combined with an "ensemble" of model runs (many forecasts with slightly different conditions), data assimilation helps predict a range of possible atmospheric states, giving an overall measure of uncertainty in a given forecast.

dycore
dynamical core
Global atmospheric model based on fluid dynamics principles, including Euler's equations of motion.

echo top
The radar-indicated top of an area of precipitation. Specifically, it contains the height of the 18 dBZ reflectivity value.

EMC
The Environmental Modeling Center.

EPIC
The Earth Prediction Innovation Center seeks to accelerate scientific research and modeling contributions through continuous and sustained community engagement in order to produce the most accurate and reliable operational modeling system in the world.

ESG
Extended Schmidt Gnomonic (ESG) grid. The ESG grid uses the map projection developed by Jim Purser of NOAA *EMC*.

ESMF
Earth System Modeling Framework. The ESMF defines itself as "a suite of software tools for developing high-performance, multi-component Earth science modeling applications."

FV3
The Finite-Volume Cubed-Sphere *dynamical core* (dycore). Developed at NOAA's Geophysical Fluid Dynamics

Laboratory (GFDL), it is a scalable and flexible dycore capable of both hydrostatic and non-hydrostatic atmospheric simulations. It is the dycore used in the UFS Weather Model.

GFS
The Global Forecast System. The GFS is a National Centers for Environmental Prediction (*NCEP*) weather forecast model that generates data for dozens of atmospheric and land-soil variables, including temperatures, winds, precipitation, soil moisture, and atmospheric ozone concentration. The system couples four separate models (atmosphere, ocean, land/soil, and sea ice) that work together to accurately depict weather conditions.

GRIB2
The second version of the World Meterological Organization's (WMO) standard for distributing gridded data.

GSI
Gridpoint Statistical Interpolation (GSI) is a variational data assimilation system, designed to be flexible, state-of-art, and run efficiently on various parallel computing platforms. It supports *RRFS* features. GSI code is publicly available on GitHub, and fix file data is publicly available here.

HPC-Stack
HPC-Stack is a repository that provides a unified, shell script-based build system for building the software stack required for numerical weather prediction (NWP) tools such as the Unified Forecast System (UFS) and the Joint Effort for Data assimilation Integration (JEDI) framework. It is being phased out in favor of *spack-stack*. HPC-Stack documentation is available, but the repository and documentation is rarely updated since it is being deprecated.

HRRR
High Resolution Rapid Refresh. The HRRR is a NOAA real-time 3-km resolution, hourly updated, cloud-resolving, convection-allowing atmospheric model initialized by 3-km grids with 3-km radar assimilation. Radar data is assimilated in the HRRR every 15 min over a 1-hour period adding further detail to that provided by the hourly data assimilation from the 13-km radar-enhanced Rapid Refresh.

JCSDA
Joint Center for Data Satellite Assimilation
The Joint Center for Satellite Data Assimilation is a multi-agency research center hosted by the University Corporation for Atmospheric Research (UCAR). JCSDA is dedicated to improving and accelerating the quantitative use of research and operational satellite data in weather, ocean, climate, and environmental analysis and prediction systems.

LAM
Limited Area Model (grid type), formerly known as the "Stand-Alone Regional" or SAR. LAM grids use a regional (rather than global) configuration of the *FV3 dynamical core*.

MPI
MPI stands for Message Passing Interface. An MPI is a standardized communication system used in parallel programming. It establishes portable and efficient syntax for the exchange of messages and data between multiple processors that are used by a single computer program. An MPI is required for high-performance computing (HPC) systems.

MRW
Medium-Range Weather Application
The Medium-Range Weather Application is a UFS Application that targets predictions of atmospheric behavior out to about two weeks. It packages a prognostic atmospheric model (the UFS Weather Model), pre- and post-processing tools, and a community workflow.

NAM
North American Mesoscale Forecast System. NAM generates multiple grids (or domains) of weather forecasts over the North American continent at various horizontal resolutions. Each grid contains data for dozens of weather parameters, including temperature, precipitation, lightning, and turbulent kinetic energy. NAM uses additional numerical weather models to generate high-resolution forecasts over fixed regions, and occasionally to follow significant weather events like hurricanes.

namelist
A namelist defines a group of variables or arrays. Namelists are an I/O feature for format-free input and output of variables by key-value assignments in Fortran compilers. Fortran variables can be read from and written to plain-text files in a standardised format, usually with a `.nml` file ending.

NCAR
The National Center for Atmospheric Research.

NCEP
National Centers for Environmental Prediction (NCEP) is an arm of the National Weather Service consisting of nine centers. More information can be found at https://www.ncep.noaa.gov.

NEMSIO
A binary format for atmospheric model output from *NCEP*'s Global Forecast System (*GFS*).

netCDF
NetCDF (Network Common Data Form) is a file format and community standard for storing multidimensional scientific data. It includes a set of software libraries and machine-independent data formats that support the creation, access, and sharing of array-oriented scientific data.

NUOPC
The National Unified Operational Prediction Capability Layer "defines conventions and a set of generic components for building coupled models using the Earth System Modeling Framework (*ESMF*)."

NWP
Numerical Weather Prediction (NWP) takes current observations of weather and processes them with computer models to forecast the future state of the weather.

NWS
The National Weather Service (NWS) is an agency of the United States government that is tasked with providing weather forecasts, warnings of hazardous weather, and other weather-related products to organizations and the public for the purposes of protection, safety, and general information. It is a part of the National Oceanic and Atmospheric Administration (NOAA) branch of the Department of Commerce.

offline UPP
Refers to cases where UPP is built standalone and run separately from the model.

RAP
Rapid Refresh. The continental-scale NOAA hourly-updated assimilation/modeling system operational at *NCEP*. RAP covers North America and is comprised primarily of a numerical forecast model and an analysis/assimilation system to initialize that model. RAP is complemented by the higher-resolution 3km High-Resolution Rapid Refresh (*HRRR*) model.

Repository
A central location in which files (e.g., data, code, documentation) are stored and managed.

RRFS
The Rapid Refresh Forecast System (RRFS) is NOAA's next-generation convection-allowing, rapidly-updated, ensemble-based data assimilation and forecasting system currently scheduled for operational implementation in 2024. It is designed to run forecasts on a 3-km *CONUS* domain.

SDF
Suite Definition File. An external file containing information about the construction of a physics suite. It describes the schemes that are called, in which order they are called, whether they are subcycled, and whether they are assembled into groups to be called together.

SRW
Short-Range Weather Application
The Short-Range Weather Application is a UFS Application that targets predictions of atmospheric behavior

on a limited spatial domain and on time scales from minutes out to about two days. It packages a prognostic atmospheric model (the UFS Weather Model), pre- and post-processing tools, and a community workflow.

Spack
Spack is a package management tool designed to support multiple versions and configurations of software on a wide variety of platforms and environments. It was designed for large supercomputing centers, where many users and application teams share common installations of software on clusters with exotic architectures.

spack-stack
The spack-stack is a collaborative effort between the NOAA Environmental Modeling Center (*EMC*), the UCAR Joint Center for Satellite Data Assimilation (*JCSDA*), and the Earth Prediction Innovation Center (*EPIC*). *spack-stack* is a repository that provides a *Spack*-based method for building the software stack required for numerical weather prediction (NWP) tools such as the Unified Forecast System (UFS) and the Joint Effort for Data assimilation Integration (JEDI) framework. *spack-stack* uses the Spack package manager along with custom Spack configuration files and Python scripts to simplify installation of the libraries required to run various applications. The *spack-stack* can be installed on a range of platforms and comes pre-configured for many systems. Users can install the necessary packages for a particular application and later add the missing packages for another application without having to rebuild the entire stack.

UFS
The Unified Forecast System is a community-based, coupled, comprehensive Earth modeling system consisting of several applications (apps). These apps span regional to global domains and sub-hourly to seasonal time scales. The UFS is designed to support the *Weather Enterprise* and to be the source system for NOAA's operational numerical weather prediction applications. For more information, visit https://ufscommunity.org/.

Updraft helicity
Helicity measures the rotation in a storm's updraft (rising) air. Significant rotation increases the probability that the storm will produce severe weather, including tornadoes. See http://ww2010.atmos.uiuc.edu/(Gh)/guides/mtr/svr/modl/fcst/params/hel.rxml for more details on updraft helicity.

Weather Enterprise
Individuals and organizations from public, private, and academic sectors that contribute to the research, development, and production of weather forecast products; primary consumers of these weather forecast products.

Weather Model
A prognostic model that can be used for short- and medium-range research and operational forecasts. It can be an atmosphere-only model or an atmospheric model coupled with one or more additional components, such as a wave or ocean model. The SRW App uses the UFS Weather Model.

Workflow
The sequence of steps required to run an experiment from start to finish.

write component
The output files written by the UFS Weather Model use an Earth System Modeling Framework (ESMF) component, referred to as the write component because the UPP cannot directly process output on the native grid types (e.g., "GFDLgrid", "ESGgrid"). Output fields are interpolated to a write component grid before writing them to an output file.

INDEX