UFS Weather Model Users Guide

Mar 10, 2020

CONTENTS

CHAPTER

ONE

INTRODUCTION

The Unified Forecast System (*UFS*) *Weather Model* (WM) is a prognostic model that can be used for short- and medium-range research and operational forecasts, as exemplified by its use in the operational Global Forecast System (GFS) of the National Oceanic and Atmospheric Administration (NOAA). The UFS WM v1.0 is the first public release of this software and represents a snapshot of a continuously evolving system undergoing open development. More information about the UFS can be found in its portal at https://ufscommunity.org/.

Key architectural elements of the UFS WM, along with links to external detailed documentation for those elements, are listed below:

- The Finite-Volume Cubed-Sphere (FV3) dynamical core.
- The Flexible Modeling System (*FMS*), a software infrastructure used for functions such as parallelization.
- The Common-Community Physics Package (*CCPP*), a library of physical parameterizations and the framework to use it with the model. *Parameterization or physics scheme* is defined here.
- The stochastic physics capability, including the Stochastic Kinetic Backscatter Scheme (SKEBS), the Stochastically Perturbed Parameterization Tendencies (SPPT) scheme, the perturbed boundary layer humidity (SHUM) scheme, and the cellular automata method.
- The NOAA Environmental Modeling System (*NEMS*) model driver used to create the main program.
- **The libraries needed to build the system, such as:**
  - National Centers for Environmental Prediction (NCEP) Libraries
  - Earth System Modeling Framework (ESMF)
  - External libraries
- The build system used to compile the code and generate the executable.
- The regression tests used to maintain software integrity as innovations are added.

For the UFS WM v1.0 release, the following aspects are supported:

- Global configuration with resolutions of C96 (~100 km), C192 (~50 km), C384 (25 km), and C768 (~13 km)
- Sixty-four vertical levels at predetermined locations.
- Four physics suites (*suite*), corresponding to GFS v15.2 (operational at the time of the release) and GFS v16beta (October 2019 version, in preparation for operational implementation in 2021). Variants with and without prediction of Sea Surface Temperature (SST) are included.
- Ability to run with or without SKEBS, SPPT, and SHUM.
- Ability to initialize from GFS files in Gridded Binary v2 (GRIB2) or NEMS Input/Output (NEMSIO) format for past dates, starting January 1, 2018, when the preprocessing utility chgres_cube is employed. Dates before that may work, but are not guaranteed.

- Output files in Network Common Data Form (NetCDF) format.

The GFS_v15p2 physics suite uses the following physical parameterizations: the Simplified Arakawa Schubert shallow and deep convective schemes, the Geophysical Fluid Dynamics Laboratory (GFDL) microphysics scheme, the Noah Land Surface Model (LSM), the Rapid Radiative Transfer Model for Global Circulation Models (RRTMG) radiation scheme, the hybrid eddy-diffusivity mass-flux (EDMF) planetary boundary layer (PBL) scheme based on the Smagorinsky K theory, an orographic gravity wave drag (GWD) parameterization, and the Near SST (NSST) ocean scheme to predict SST. In the GFS_v16beta suite, a moist TKE-based EDMF scheme replaces the K-based one and a non-stationary GWD parameterization is added. The GFS_v15p2_no_nsst and the GFS_v16beta_no_nsst suites use a simple ocean scheme instead of the NSST scheme. This simple ocean scheme keeps the SST constant throughout the forecast and is recommended for use when the initial conditions do not contain all fields needed to initialize the NSST scheme.

Even when using physics suite GFS_v15p2, the UFS WM v1 differs from the operational GFS v15.2 in a few ways. First, the public release code reflects the state of development as of the fall of 2019, and therefore the parameterizations contain innovations beyond what is in GFSv15.2 operations. For example, the GFDL microphysics distributed for use in GFS v15.2 and GFS v16beta is the same scheme and contains development beyond what was transitioned to operations for GFS v15 in June 2019. Second, the public release code uses the CCPP as the interface for calling physics, while in operations the Interoperable Physics Driver (IPD) is used. NOAA is currently working toward phasing out the IPD from UFS applications. Validation tests demonstrated that CCPP and IPD give bit-for-bit identical results when the same physics is employed and selected performance flags are excluded at compilation time. When performance compiler flags employed in operational production are used, runs with CCPP and IPD for the same physics suite yield differences comparable to running the model in different computational platforms. Finally, the operational GFS runs in NOAA Central Operations computational platforms. When users run the model in different platforms, the results will differ.

It should also be noted that further changes are expected to the GFS v16 suite before it is implemented in operations in 2021.

The UFS WM v1 code is portable and can be used with Linux and Mac operating systems with Intel and GNU compilers. It has been tested in a variety of platforms widely used by atmospheric scientists, such as the NOAA research Hera system, the National Center for Atmospheric Research (NCAR) Cheyenne system, the National Science Foundation Stampede system, and Mac laptops.

Note: At this time, the following aspects are unsupported: standalone regional domains, configurations in which a mediator is used to couple the atmospheric model to models of other earth domains (such as ocean, ice, and waves), horizontal resolutions other than the supported ones, different number or placement of vertical levels, physics suites other than GFS v15.2 and GFS v16beta the *cellular automata* stochastic scheme, initialization from sources other than GFS, the use of different file formats for input and output, and the use of the model in different computational platforms. It is expected that the UFS WM supported capabilities will be expanded in future releases.

It should be noted that the UFS WM is a component of the UFS Medium-Range (MR) Weather Application (App), which also contains pre- and post-processing components, a comprehensive build system, and workflows for configuration and execution of the application. At this time, the UFS WM is only supported to the general community for use as part of the UFS MR Weather App. However, those wishing to contribute development to the UFS WM should become familiar with the procedures for running the model as a standalone component and for executing the regression tests described in this guide to make sure no inadvertent changes to the results have been introduced during the development process.

Support for the UFS WM is provided through the UFS Forum by the Developmental Testbed Center (DTC) and other groups involved in UFS development, such as NOAA's Environmental Modeling Center (EMC), NOAA research laboratories (GFDL, NSSL, ESRL, and AOML), and NCAR. UFS users and developers are encouraged not only to post questions, but also to help address questions posted by other members of the community.

This WM User's Guide is organized as follows:

- Chapter 2 (Code Overview) provides a description of the various code repositories from which source code is pulled and an overview of the directory structure.
- Chapter 3 (Building and Running the WM) explains how to use the WM without an application.
- Chapter 4 (Inputs and Outputs) lists the model inputs and outputs and has a description of the key files.
- Chapter 5 (SDF and namelist samples and best practices) contains a description of the *Suite Definition File (SDF)* and namelists needed to configure the model for running with the GFS v15.2 and GFS v16beta physics suites.
- Chapter 6 (Contributing development) goes beyond the capabilities supported in the public release to cover code management for conducting development and proposing contributions back to the authoritative code repositories. It should be noted that the regression tests described here are mandatory for committing code back to the ufs-weather-model authoritative code repository. These regressions tests differ from those distributed with the workflows for UFS applications, which are intended for application users and developers to assess the quality of their installations and the impact of their code changes.

Finally, Chapters 7 and 8 contain a list of acronyms and a glossary, respectively.

CHAPTER

TWO

CODE OVERVIEW

2.1 UFS Weather Model Hierarchical Repository Structure

The ufs-weather-model repository supports the short- and medium-range UFS applications. It contains atmosphere and wave components and some infrastructure components. Each of these components has its own repository. All the repositories are currently located in GitHub with public access to the broad community. Table 2.1 describes the list of repositories that comprises the ufs-weather-model.

Table 2.1: *List of Repositories that comprise the ufs-weather-model*

Repository Description	Authoritative repository URL
Umbrella repository for the UFS Weather Model	https://github.com/ufs-community/ufs-weather-model
Infrastructure: Flexible Modeling System	https://github.com/NOAA-GFDL/FMS
Infrastructure: NOAA Environmental Modeling System	https://github.com/NOAA-EMC/NEMS
Infrastructure: Utilities	https://github.com/NOAA-EMC/NCEPLIBS-pyprodutil
Framework to connect the CCPP library to a host model	https://github.com/NCAR/ccpp-framework
CCPP library of physical parameterizations	https://github.com/NCAR/ccpp-physics
Umbrella repository for the physics and dynamics of the atmospheric model	https://github.com/NOAA-EMC/fv3atm
FV3 dynamical core	https://github.com/NOAA-EMC/GFDL_atmos_cubed_sphere
Stochastic physics pattern generator	https://github.com/noaa-psd/stochastic_physics

In the table, the left column contains a description of each repository, and the right column shows the component repositories which are pointing to (or will point to) the authoritative repositories. The ufs-weather-model currently uses git submodule to manage the sub-components.

The umbrella repository for the UFS Weather Model is named ufs-weather-model. Under this repository reside a number of submodules that are nested in specific directories under the parent repository's working directory. When the ufs-weather-model repository is cloned, the *.gitmodules* file creates the following directories:

```
ufs-weather-model/
├── FMS                                    https://github.com/NOAA-GFDL/FMS
├── FV3                                    https://github.com/NCAR/fv3atm
│   ├── atmos_cubed_sphere                 https://github.com/NCAR/GFDL_atmos_cubed_
↪sphere
│   ├── ccpp
│   │   ├── framework                      https://github.com/NCAR/ccpp-framework
│   │   ├── physics                        https://github.com/NCAR/ccpp-physics
├── NEMS                                   https://github.com/NCAR/NEMS
```

(continues on next page)

(continued from previous page)

```
|    └── tests/produtil/NCEPLIBS-pyprodutil  https://github.com/NOAA-EMC/NCEPLIBS-
↪pyprodutil
├── stochastic_physics                      https://github.com/noaa-psd/stochastic_
↪physics
```

2.2 Directory Structure

When the ufs-weather-model is cloned, the basic directory structure will be similar to the example below. Files and some directories have been removed for brevity.

```
ufs-weather-model/
├── cmake                --------- cmake configuration files
├── compsets             --------- configurations used by some regression tests
├── conf                 --------- compile options for Tier 1 and 2 platforms
├── doc                  --------- README.s with build, reg-test hints
├── FMS                  --------- The Flexible Modeling System (FMS),a software␣
↪framework
├── FV3                  --------- FV3 atmosphere model
|    ├── atmos_cubed_sphere   ---- FV3 dynamic core
|    |    ├── docs
|    |    ├── driver
|    |    ├── model
|    |    └── tools
|    ├── ccpp              -------- Common Community Physics Package
|    |    ├── config
|    |    ├── driver
|    |    ├── framework    -------- CCPP framework
|    |    ├── physics      -------- CCPP compliant physics schemes
|    |    └── suites       -------- CCPP physics suite definition files (SDFs)
|    ├── cpl               -------- Coupling field data structures
|    ├── gfsphysics
|    |    ├── CCPP_layer
|    |    ├── GFS_layer
|    |    └── physics      --------- unused - IPD version of physics codes
|    ├── io                --------- FV3 write grid comp code
|    ├── ipd               --------- unused - IPD driver/interfaces
|    ├── stochastic_physics  ------ Cmakefile for stochastic physics code
├── log                  --------- log files from NEMS compset regression tests
├── modulefiles          --------- system module files for supported HPC systems
├── NEMS                 --------- NOAA Earth Modeling System framework
|    ├── exe
|    ├── src
|    └── test
├── parm                 --------- regression test configurations
├── stochastic_physics    -------- stochastic physics pattern generator
├── tests                --------- regression test scripts
```

The physics subdirectory in the *gfsphysics* directory is not used or supported as part of this release (all physics is available through the *CCPP* using the repository described in Table 2.1).

CHAPTER

THREE

BUILDING AND RUNNING THE UFS WEATHER MODEL

3.1 Prerequisite Libraries

The UFS Weather Model requires a number of libraries for it to compile. There are two categories of libraries that are needed:

1. Bundled libraries (NCEPLIBS). These are libraries developed for use with NOAA weather models. Most have an NCEPLIBS prefix in the repository, e.g. NCEPLIBS-bacio. Select tools from the UFS Utilities repository (UFS-UTILS) are also included in this category. A list of the bundled libraries tested with this WM release is in the top-level `README` of the NCEPLIBS repository (**be sure to look at the tag in that repository that matches the tag on this WM release**).
2. Third-party libraries (NCEPLIBS-external). These are libraries that were developed external to the UFS Weather Model. They are general software packages that are also used by other models in the community. Building these is optional, since existing builds of these libraries can be pointed to instead. A list of the external libraries tested with this WM release is in the top-level `README` of the NCEPLIBS-external repository. Again, be sure to look at the tag in that repository that matches the tag on this WM release.

Note: The libraries in NCEPLIBS-external must be built *before* the libraries in NCEPLIBS.

See this wiki link for an explanation of which platforms and compilers are supported. This will help to determine if you need to build NCEPLIBS and NCEPLIBS-external or are working on a system that is already pre-configured. On pre-configured platforms, the libraries are already available.

If you do have to build the libraries, it is a good idea to check the platform- and compiler-specific `README` files in the doc/ directory of the NCEPLIBS-external repository as a first step, to see if your system or one similar to it is included. These files have detailed instructions for building NCEPLIBS-external, NCEPLIBS, and the UFS Weather Model. They may be all the documentation you need. Be sure to use the tag that corresponds to this version of the WM, and define a WORK directory path before you get started.

If your platform is not included in these platform- and compiler-specific `README` files, there is a more generic set of instructions in the `README` file at the top level of the NCEPLIBS-external repository, and at the top level of the NCEPLIBS repository. It may still be a good idea to look at some of the platform- and compiler-specific `README` files as a guide. Again, be sure to use the tag that corresponds to this version of the WM.

The top-level `README` in the NCEPLIBS-external repository includes a troubleshooting section that may be helpful.

You can also get expert help through a user support forum set up specifically for issues related to build dependencies.

3.2 Downloading the Weather Model Code

To clone the ufs-weather-model repository for this v1.0.0 release, execute the following commands:

```
git clone https://github.com/ufs-community/ufs-weather-model.git ufs-weather-model
cd ufs-weather-model
git checkout ufs-v1.0.0
git submodule update --init --recursive
```

Compiling the model will take place within the *ufs-weather-model* directory you just created.

3.3 Building the Weather Model

3.3.1 Setting environment variables for paths to NCEPLIBS and NCEPLIBS-external

You will need to make sure that the WM has the paths to the libraries that it requires. In order to do that, these environment variables need to be set, as shown in Table 3.1 and Table 3.2 for the bash shell.

Table 3.1: *Bundled libraries (NCEPLIBS) required for the Weather Model*

NCEP Library	Environment Variables
nemsio	export NEMSIO_INC=<path_to_nemsio_include_dir>
	export NEMSIO_LIB=<path_to_nemsio_lib_dir>/libnemsio<version>.a
bacio	export BACIO_LIB4=<path_to_bacio_lib_dir>/libbacio<version>.a
splib	export SP_LIBd=<path_to_sp_lib_dir>/libsp<version>_d.a
w3emc	export W3EMC_LIBd=<path_to_w3emc_lib_dir>/libw3emc<version>_d.a
w3nco	export W3NCO_LIBd=<path_to_w3nco_lib_dir>/libw3nco<version>_d.a

Table 3.2: *Third-party libraries (NCEPLIBS-external) required for the Weather Model*

Library	Environment Variables
NetCDF	export NETCDF=<path_to_netcdf_install_dir>
ESMF	export ESMFMKFILE=<path_to_esmfmk_file>/esmf.mk

The following are a few different ways to set the required environment variables to the correct values. If you are running on one of the pre-configured platforms, you can set them using modulefiles. Modulefiles for all supported platforms are located in `modulefiles/<platform>/fv3`. To load the modules, for example on hera, run:

```
cd modulefiles/hera.intel
module use $(pwd)
module load fv3
cd ../..
```

If you are not running on one of the pre-configured platforms, you will need to set the environment variables in a different way.

If you used one of the platform- and compiler-specific `README` files in the `doc/` directory of NCEPLIBS-external to build the prerequisite libraries, there is a script in the `NCEPLIBS-ufs-v1.0.0/bin` directory called `setenv_nceplibs.sh` that will set the NCEPLIBS-external variables for you.

Of course, you can also set the values of these variables yourself if you know where the paths are on your system.

3.3.2 Setting other environment variables

You will also need to set the `CMAKE_Platform` environment variable. See the `README` files in the `doc/` directories of the NCEPLIBS-external repository for recognized values.

The default value is:

```
export CMAKE_Platform=linux.<compiler>
```

Where <compiler> is either Intel or GNU. You may also wish to set the following environment variables:

- `CMAKE_Platform`: if not set the default is linux.${COMPILER}
- `CMAKE_C_COMPILER`: if not set the default is mpicc
- `CMAKE_CXX_COMPILER`: if not set the default is mpicxx
- `CMAKE_Fortran_COMPILER`: if not set the default is mpif90

In order to have one or more CCPP physics suites available at runtime, you need to select those suites at build time by setting the `CCPP_SUITES` environment variable. Multiple suites can be set, as shown below in an example for the bash shell:

```
export CCPP_SUITES='FV3_GFS_v15p2,FV3_GFS_v16beta'
```

If `CCPP_SUITES` is not set, the default is `'FV3_GFS_v15p2'`.

3.3.3 Building the model

The UFS Weather Model uses the cmake build system. There is a build script called `build.sh` in the top-level directory of the WM repository that ensures all necessary variables are actually set.

After setting all the environment variables, you can build the model by running the following from the *ufs-weather-model* directory:

```
./build.sh
```

Once `build.sh` is finished, you should see the executable, named `ufs_weather_model`, in the top-level directory.

Expert help is available through a user support forum set up specifically for issues related to the Weather Model.

3.4 Running the model

The UFS Weather Model wiki includes a simple test case that illustrates how the model can be run.

CHAPTER

FOUR

INPUTS AND OUTPUTS

This chapter describes the input and output files needed for executing the model in the various supported configurations.

4.1 Input files

There are three types of files needed to execute a run: static datasets (*fix* files containing climatological information), files that depend on grid resolution and initial conditions, and model configuration files (such as namelists).

4.1.1 Static datasets (i.e., *fix files*)

The static input files are listed and described in Table 4.1.

Table 4.1: *Fix files containing climatological information*

Filename	Description
aerosol.dat	External aerosols data file
CFSR.SEAICE.1982.2012.monthly.clim.grb	CFS reanalysis of monthly sea ice climatology
co2historicaldata_YYYY.txt	Monthly CO2 in PPMV data for year YYYY
global_albedo4.1x1.grb	Four albedo fields for seasonal mean climatology: 2 for strong zenith angle dependent (visible and near IR) and 2 for weak zenith angle dependent
global_glacier.2x2.grb	Glacier points, permanent/extreme features
global_h2oprdlos.f77	Coefficients for the parameterization of photochemical production and loss of water (H2O)
global_maxice.2x2.grb	Maximum ice extent, permanent/extreme features
global_mxsnoalb.uariz.t126.384.190.rg.grb	Climatological maximum snow albedo
global_o3prdlos.f77	Monthly mean ozone coefficients
global_shdmax.0.144x0.144.grb	Climatological maximum vegetation cover
global_shdmin.0.144x0.144.grb	Climatological minimum vegetation cover
global_slope.1x1.grb	Climatological slope type
global_snoclim.1.875.grb	Climatological snow depth
global_snowfree_albedo.bosu.t126.384.190.rg.grb	Climatological snowfree albedo
global_soilmgldas.t126.384.190.grb	Climatological soil moisture
global_soiltype.statsgo.t126.384.190.rg.grb	Soil type from the STATSGO dataset
global_tg3clim.2.6x1.5.grb	Climatological deep soil temperature
global_vegfrac.0.144.decpercent.grb	Climatological vegetation fraction
global_vegtype.igbp.t126.384.190.rg.grb	Climatological vegetation type
global_zorclim.1x1.grb	Climatological surface roughness
RTGSST.1982.2012.monthly.clim.grb	Monthly, climatological, real-time global sea surface temperature
seaice_newland.grb	High resolution land mask
sfc_emissivity_idx.txt	External surface emissivity data table
solarconstant_noaa_an.txt	External solar constant data table

4.1.2 Grid description and initial condition files

The input files containing grid information and the initial conditions are listed and described in Table 4.2.

Table 4.2: *Input files containing grid information and initial conditions*

Filename	Description	Date-dependent
C96_grid.tile[1-6].nc	C96 grid information for tiles 1-6	
gfs_ctrl.nc	NCEP NGGPS tracers, ak, and bk	✓
gfs_data.tile[1-6].nc	Initial condition fields (ps, u, v, u, z, t, q, O3). May include spfo3, spfo, spf02 if multiple gases are used	✓
oro_data.tile[1-6].nc	Model terrain (topographic/orographic information) for grid tiles 1-6	
sfc_ctrl.nc	Control parameters for surface input: forecast hour, date, number of soil levels	
sfc_data.tile[1-6].nc	Surface properties for grid tiles 1-6	✓

4.1.3 Model configuration files

The configuration files used by the UFS Weather Model are listed here and described below:

- *diag_table*
- *field_table*
- *input.nml*
- *model_configure*
- *nems.configure*
- *suite_[suite_name].xml* (used only at build time)

diag_table file

There are three sections in file *diag_table*: Header (Global), File, and Field. These are described below.

Header Description

The Header section must reside in the first two lines of the *diag_table* file and contain the title and date of the experiment (see example below). The title must be a Fortran character string. The base date is the reference time used for the time units, and must be greater than or equal to the model start time. The base date consists of six space-separated integers in the following format: `year month day hour minute second`. Here is an example:

```
20161003.00Z.C96.64bit.non-mono
2016 10 03 00 0 0
```

File Description

The File Description lines are used to specify the name of the file(s) to which the output will be written. They contain one or more sets of six required and five optional fields (optional fields are denoted by square brackets `[ ]`). The lines containing File Descriptions can be intermixed with the lines containing Field Descriptions as long as files are defined before fields that are to be written them. File entries have the following format:

```
"file_name", output_freq, "output_freq_units", file_format, "time_axis_units", "time_
↪axis_name"
[, new_file_freq, "new_file_freq_units"[, "start_time"[, file_duration, "file_
↪duration_units"]]]
```

These file line entries are described in Table 4.3.

Table 4.3: *Description of the six required and five optional fields used to define output file sampling rates.*

File Entry	Variable Type	Description
file_name	CHARACTER(len=128)	Output file name without the trailing ".nc"
output_freq	INTEGER	The period between records in the file_name: > 0 output frequency in output_freq_units. = 0 output frequency every time step (output_freq_units is ignored) =-1 output at end of run only (output_freq_units is ignored)
output_freq_units	CHARACTER(len=10)	The units in which output_freq is given. Valid values are "years", "months", "days", "minutes", "hours", or "seconds".
file_format	INTEGER	Currently only the netCDF file format is supported. = 1 netCDF
time_axis_units	CHARACTER(len=10)	The units to use for the time-axis in the file. Valid values are "years", "months", "days", "minutes", "hours", or "seconds".
time_axis_name	CHARACTER(len=128)	Axis name for the output file time axis. The character string must contain the string 'time'. (mixed upper and lowercase allowed.)
new_file_freq	INTEGER, OPTIONAL	Frequency for closing the existing file, and creating a new file in new_file_freq_units.
new_file_freq_units	CHARACTER(len=10), OPTIONAL	Time units for creating a new file: either years, months, days, minutes, hours, or seconds. NOTE: If the new_file_freq field is present, then this field must also be present.
start_time	CHARACTER(len=25), OPTIONAL	Time to start the file for the first time. The format of this string is the same as the global date. NOTE: The new_file_freq and the new_file_freq_units fields must be present to use this field.
file_duration	INTEGER, OPTIONAL	How long file should receive data after start time in file_duration_units. This optional field can only be used if the start_time field is present. If this field is absent, then the file duration will be equal to the frequency for creating new files. NOTE: The file_duration_units field must also be present if this field is present.
file_duration_units	CHARACTER(len=10), OPTIONAL	File duration units. Can be either years, months, days, minutes, hours, or seconds. NOTE: If the file_duration field is present, then this field must also be present.

Field Description

The field section of the diag_table specifies the fields to be output at run time. Only fields registered with `register_diag_field()`, which is an API in the FMS `diag_manager` routine, can be used in the *diag_table*.

Registration of diagnostic fields is done using the following syntax

```
diag_id = register_diag_field(module_name, diag_name, axes, ...)
```

in file `FV3/atmos_cubed_sphere/tools/fv_diagnostics.F90`. As an example, the sea level pressure is registered as:

```
id_slp = register_diag_field (trim(field), 'slp', axes(1:2), &   Time, 'sea-level␣
↪pressure', 'mb', missing_value=missing_value, range=slprange )
```

All data written out by `diag_manager` is controlled via the *diag_table*. A line in the field section of the *diag_table* file contains eight variables with the following format:

```
"module_name", "field_name", "output_name", "file_name", "time_sampling", "reduction_
↪method", "regional_section", packing
```

These field section entries are described in Table 4.4.

Table 4.4: *Description of the eight variables used to define the fields written to the output files.*

Field Entry	Variable Type	Description
module_name	CHARACTER(len=128)	Module that contains the field_name variable. (e.g. dynamic, gfs_phys, gfs_sfc)
field_name	CHARACTER(len=128)	The name of the variable as registered in the model.
output_name	CHARACTER(len=128)	Name of the field as written in file_name.
file_name	CHARACTER(len=128)	Name of the file where the field is to be written.
time_sampling	CHARACTER(len=50)	Currently not used. Please use the string "all".
reduction_method	CHARACTER(len=50)	The data reduction method to perform prior to writing data to disk. Current supported option is .false.. See `FMS/diag_manager/diag_table.F90` for more information.
regional_section	CHARACTER(len=50)	Bounds of the regional section to capture. Current supported option is "none". See `FMS/diag_manager/diag_table.F90` for more information.
packing	INTEGER	Fortran number KIND of the data written. Valid values: 1=double precision, 2=float, 4=packed 16-bit integers, 8=packed 1-byte (not tested).

Comments can be added to the diag_table using the hash symbol (#).

A brief example of the diag_table is shown below. "`...`" denote where lines have been removed.

```
20161003.00Z.C96.64bit.non-mono
2016 10 03 00 0 0

"grid_spec",     -1,  "months",   1, "days",  "time"
"atmos_4xdaily",  6,  "hours",    1, "days",  "time"
"atmos_static"   -1,  "hours",    1, "hours", "time"
"fv3_history",    0,  "hours",    1, "hours", "time"
"fv3_history2d",  0,  "hours",    1, "hours", "time"

#
#======================
# ATMOSPHERE DIAGNOSTICS
#======================
###
# grid_spec
###
 "dynamics", "grid_lon",  "grid_lon",  "grid_spec", "all", .false.,  "none", 2,
 "dynamics", "grid_lat",  "grid_lat",  "grid_spec", "all", .false.,  "none", 2,
 "dynamics", "grid_lont", "grid_lont", "grid_spec", "all", .false.,  "none", 2,
 "dynamics", "grid_latt", "grid_latt", "grid_spec", "all", .false.,  "none", 2,
 "dynamics", "area",      "area",      "grid_spec", "all", .false.,  "none", 2,
###
# 4x daily output
###
```

(continues on next page)

(continued from previous page)

```
 "dynamics",  "slp",       "slp",       "atmos_4xdaily", "all", .false.,  "none", 2
 "dynamics",  "vort850",   "vort850",   "atmos_4xdaily", "all", .false.,  "none", 2
 "dynamics",  "vort200",   "vort200",   "atmos_4xdaily", "all", .false.,  "none", 2
 "dynamics",  "us",        "us",        "atmos_4xdaily", "all", .false.,  "none", 2
 "dynamics",  "u1000",     "u1000",     "atmos_4xdaily", "all", .false.,  "none", 2
 "dynamics",  "u850",      "u850",      "atmos_4xdaily", "all", .false.,  "none", 2
 "dynamics",  "u700",      "u700",      "atmos_4xdaily", "all", .false.,  "none", 2
 "dynamics",  "u500",      "u500",      "atmos_4xdaily", "all", .false.,  "none", 2
 "dynamics",  "u200",      "u200",      "atmos_4xdaily", "all", .false.,  "none", 2
 "dynamics",  "u100",      "u100",      "atmos_4xdaily", "all", .false.,  "none", 2
 "dynamics",  "u50",       "u50",       "atmos_4xdaily", "all", .false.,  "none", 2
 "dynamics",  "u10",       "u10",       "atmos_4xdaily", "all", .false.,  "none", 2

...
###
# gfs static data
###
 "dynamics",  "pk",        "pk",        "atmos_static",  "all", .false.,  "none", 2
 "dynamics",  "bk",        "bk",        "atmos_static",  "all", .false.,  "none", 2
 "dynamics",  "hyam",      "hyam",      "atmos_static",  "all", .false.,  "none", 2
 "dynamics",  "hybm",      "hybm",       "atmos_static",  "all", .false.,  "none", 2
 "dynamics",  "zsurf",     "zsurf",      "atmos_static",  "all", .false.,  "none", 2
###
# FV3 variables needed for NGGPS evaluation
###
"gfs_dyn",    "ucomp",      "ugrd",     "fv3_history",    "all",  .false.,  "none", ␣
↪2
"gfs_dyn",    "vcomp",      "vgrd",     "fv3_history",    "all",  .false.,  "none", ␣
↪2
"gfs_dyn",    "sphum",      "spfh",     "fv3_history",    "all",  .false.,  "none", ␣
↪2
"gfs_dyn",    "temp",       "tmp",      "fv3_history",    "all",  .false.,  "none", ␣
↪2
...
"gfs_phys",  "ALBDO_ave",    "albdo_ave", "fv3_history2d", "all", .false., "none",  2
"gfs_phys",  "cnvprcp_ave",  "cprat_ave", "fv3_history2d", "all", .false., "none",  2
"gfs_phys",  "cnvprcpb_ave", "cpratb_ave","fv3_history2d", "all", .false., "none",  2
"gfs_phys",  "totprcp_ave",  "prate_ave", "fv3_history2d", "all", .false., "none",  2
...
"gfs_sfc",   "crain",   "crain",    "fv3_history2d",  "all",  .false.,  "none",  2
"gfs_sfc",   "tprcp",   "tprcp",    "fv3_history2d",  "all",  .false.,  "none",  2
"gfs_sfc",   "hgtsfc",  "orog",     "fv3_history2d",  "all",  .false.,  "none",  2
"gfs_sfc",   "weasd",   "weasd",    "fv3_history2d",  "all",  .false.,  "none",  2
"gfs_sfc",   "f10m",    "f10m",     "fv3_history2d",  "all",  .false.,  "none",  2
...
```

More information on the content of this file can be found in `FMS/diag_manager/diag_table.F90`.

Note: None of the lines in the *diag_table* can span multiple lines.

field_table file

The FMS field and tracer managers are used to manage tracers and specify tracer options. All tracers advected by the model must be registered in an ASCII table called *field_table*. The field table consists of entries in the following format:

The first line of an entry should consist of three quoted strings:

- The first quoted string will tell the field manager what type of field it is. The string "TRACER" is used to declare a field entry.
- The second quoted string will tell the field manager which model the field is being applied to. The supported type at present is "atmos_mod" for the atmosphere model.
- The third quoted string should be a unique tracer name that the model will recognize.

The second and following lines are called methods. These lines can consist of two or three quoted strings. The first string will be an identifier that the querying module will ask for. The second string will be a name that the querying module can use to set up values for the module. The third string, if present, can supply parameters to the calling module that can be parsed and used to further modify values.

An entry is ended with a forward slash (/) as the final character in a row. Comments can be inserted in the field table by having a hash symbol (#) as the first character in the line.

Below is an example of a field table entry for the tracer called "sphum":

```
# added by FRE: sphum must be present in atmos
# specific humidity for moist runs
 "TRACER", "atmos_mod", "sphum"
           "longname",      "specific humidity"
           "units",         "kg/kg"
           "profile_type", "fixed", "surface_value=3.e-6" /
```

In this case, methods applied to this tracer include setting the long name to "specific humidity", the units to "kg/kg". Finally a field named "profile_type" will be given a child field called "fixed", and that field will be given a field called "surface_value" with a real value of 3.E-6. The "profile_type" options are listed in Table 4.5. If the profile type is "fixed" then the tracer field values are set equal to the surface value. If the profile type is "profile" then the top/bottom of model and surface values are read and an exponential profile is calculated, with the profile being dependent on the number of levels in the component model.

Table 4.5: *Tracer profile setup from FMS/tracer_manager/tracer_manager.F90.*

Method Type	Method Name	Method Control
profile_type	fixed	surface_value = X
profile_type	profile	surface_value = X, top_value = Y (atmosphere)

For the case of

```
"profile_type","profile","surface_value = 1e-12, top_value = 1e-15"
```

in a 15 layer model this would return values of surf_value = 1e-12 and multiplier = 0.6309573, i.e 1e-15 = 1e-12*(0.6309573^15).

A method is a way to allow a component module to alter the parameters it needs for various tracers. In essence, this is a way to modify a default value. A namelist can supply default parameters for all tracers and a method, as supplied through the field table, will allow the user to modify the default parameters on an individual tracer basis. The lines in this file can be coded quite flexibly. Due to this flexibility, a number of restrictions are required. See FMS/field_manager/field_manager.F90 for more information.

input.nml file

The atmosphere model reads many parameters from a Fortran namelist file, named *input.nml*. This file contains several Fortran namelist records, some of which are always required, others of which are only used when selected physics options are chosen.

The following link describes the various physics-related namelist records:

https://dtcenter.org/GMTB/v4.0/sci_doc/CCPPsuite_nml_desp.html

The following link describes the stochastic physics namelist records

https://stochastic-physics.readthedocs.io/en/ufs-v1.0.0/namelist_options.html

The following link describes some of the other namelist records (dynamics, grid, etc):

https://www.gfdl.noaa.gov/wp-content/uploads/2017/09/fv3_namelist_Feb2017.pdf

The namelist section relating to the FMS diagnostic manager is described in the last section of this chapter.

model_configure file

This file contains settings and configurations for the NUOPC/ESMF main component, including the simulation start time, the processor layout/configuration, and the I/O selections. Table 4.6 shows the following parameters that can be set in *model_configure* at run-time.

Table 4.6: *Parameters that can be set in model_configure at run-time.*

Parameter	Meaning	Type	Default Value
print_esmf	flag for ESMF PET files	logical	.true.
PE_MEMBER01	total number of tasks for ensemble number 1	integer	150 (for c96 with quilt)
start_year	start year of model integration	integer	2019
start_month	start month of model integration	integer	09
start_day	start day of model integration	integer	12
start_hour	start hour of model integration	integer	00
start_minute	start minute of model integration	integer	0
start_second	start second of model integration	integer	0
nhours_fcst	total forecast length	integer	48
dt_atmos	atmosphere time step in second	integer	1800 (for C96)
output_1st_tstep_rst	output first time step history file after restart	logical	.false.
memuse_verbose	flag for printing out memory usage	logical	.false.
atmos_nthreads	number of threads for atmosphere	integer	4
restart_interval	frequency to output restart file	integer	0 (write restart file at the end of integration)
quilting	flag to turn on quilt	logical	.true.
write_groups	total number of groups	integer	2
write_tasks_per_group	total number of write tasks in each write group	integer	6
output_history	flag to output history files	logical	.true.
num_files	number of output files	integer	2
filename_base	file name base for the output files	character(255)	'atm' 'sfc'
output_grid	output grid	character(255)	gaussian_grid
output_file	output file format	character(255)	nemsio
imo	i-dimension for output grid	integer	384
jmo	j-dimension for output grid	integer	190
nfhout	history file output frequency	integer	3
nfhmax_hf	forecast length of high history file	integer	0 (0:no high frequency output)
nfhout_hf	high history file output frequency	integer	1
nsout	output frequency of number of time step	integer	-1 (negative: turn off the option, 1: output history file at every time step)

Table 4.7 shows the following parameters in *model_configure* that are not usually changed.

Table 4.7: *Parameters that are not usually changed in model_configure at run-time.*

Parameter	Meaning	Type	Default Value
total_member	total number of ensemble member	integer	1
RUN_CONTINUE	Flag for more than one NEMS run	logical	.false.
ENS_SPS	flag for the ensemble stochastic coupling flag	logical	.false.
calendar	type of calendar year	character(*)	'gregorian'
fhrot	forecast hour at restart for nems/earth grid component clock in coupled model	integer	0
cpl	flag for coupling with MOM6/CICE5	logical	.false.
write_dopost	flag to do post on write grid component	logical	.false.
ideflate	lossless compression level	integer	1 (0:no compression, range 1-9)
nbits	lossy compression level	integer	14 (0: lossless, range 1-32)
write_nemsioflip	flag to flip the vertical level for nemsio file	logical	.true.
write_fsyncflag	flag to check if a file is synced to disk	logical	.true.
iau_offset	IAU offset lengdth	integer	0

nems.configure file

This file contains information about the various NEMS components and their run sequence. In the current release, this is an atmosphere-only model, so this file is simple and does not need to be changed. A sample of the file contents is below:

```
EARTH_component_list: ATM
ATM_model:            fv3
runSeq::
  ATM
::
```

The SDF (Suite Definition File) file

There are two SDFs currently supported: *suite_FV3_GFS_v15p2.xml* and *suite_FV3_GFS_v16beta.xml*.

4.2 Output files

The following files are output when running *fv3.exe* in the default configuration (six files of each kind, corresponding to the six tiles of the model grid):

- *atmos_4xdaily.tile[1-6].nc*
- *atmos_static.tile[1-6].nc*
- *sfcfHHH.nc*
- *atmfHHH.nc*

- *grid_spec.tile[1-6].nc*

Note that the sfcf* and atmf* files are not output on the 6 tiles, but instead as a single global gaussian grid file. The specifications of the output files (type, projection, etc) may be overridden in the *model_configure* input file.

Standard output files are *logf???*, and out and err as specified by the job submission. ESMF may also produce log files (controlled by variable print_esmf in the *model_configure* file), called *PET???.ESMF_LogFile*.

4.3 Additional Information about the FMS Diagnostic Manager

The UFS Weather Model output is managed through the FMS (Flexible Modeling System) diagnostic manager (`FMS/diag_manager`) and is configured using the *diag_table* file. Data can be written at any number of sampling and/or averaging intervals specified at run-time. More information about the FMS diagnostic manager can be found at: https://data1.gfdl.noaa.gov/summer-school/Lectures/July16/03_Seth1_DiagManager.pdf

4.3.1 Diagnostic Manager namelist

The `diag_manager_nml` namelist contains values to control the behavior of the diagnostic manager. Some of the more common namelist options are described in Table 4.8. See `FMS/diag_manager/diag_manager.F90` for the complete list.

Table 4.8: *Namelist variables used to control the behavior of the diagnostic manager.*

Namelist variable	Type	Description	Default value
max_files	INTEGER	Maximum number of files allowed in diag_table	31
max_output_fields	INTEGER	Maximum number of output fields allowed in diag_table	300
max_input_fields	INTEGER	Maximum number of registered fields allowed	300
prepend_date	LOGICAL	Prepend the file start date to the output file. .TRUE. is only supported if the diag_manager_init routine is called with the optional time_init parameter.	.TRUE.
do_diag_field_log	LOGICAL	Write out all registered fields to a log file	.FALSE.
use_cmor	LOGICAL	Override the missing_value to the CMOR value of -1.0e20	.FALSE.
issue_oor_warnings	LOGICAL	Issue a warning if a value passed to diag_manager is outside the given range	.TRUE.
oor_warnings_fatal	LOGICAL	Treat out-of-range errors as FATAL	.FALSE.
debug_diag_manager	LOGICAL	Check if the diag table is set up correctly	.FALSE.

This release of the UFS Weather Model uses the following namelist:

```
&diag_manager_nml
  prepend_date = .false.
/
```

CHAPTER

FIVE

SDF AND NAMELIST SAMPLES AND BEST PRACTICES

The public release of the UFS MR Weather App includes four supported physics suites: GFS_v15p2, GFS_v15p2_no_nsst, GFS_v16beta, and GFS_v16beta_no_nsst. You will find the Suite Definition Files (SDFs) for these suites in

https://github.com/NOAA-EMC/fv3atm/tree/ufs-v1.0.0/ccpp/suites

(no other SDFs are available with this release). You will find the namelists for the C96 configuration here:

https://github.com/ufs-community/ufs-weather-model/tree/ufs-v1.0.0/parm/ccpp_v15p2_c96.nml.IN

and

https://github.com/ufs-community/ufs-weather-model/tree/ufs-v1.0.0/parm/ccpp_v16beta_c96.nml.IN

As noted in the file names, these namelists are for the operational (v15p2) and developmental (v16beta) GFS suites. Each of these namelists are relevant to the suites with and without the SST prediction scheme, that is, they are relevant for the suite that employs NSST and for the suite that employs the simple ocean model (*no_nsst*). The only difference in the namelist regarding how SST prediction is addressed is variable *nstf_name*. For more information about this variable and for information about namelist options for higher resolution configurations, please consult the CCPP v4 Scientific Documentation.

The four CCPP suites for the UFS MR Weather App release are supported in four grid resolutions: C96, C192, C384, and C768, with 64 vertical levels.

An in depth description of the namelist settings, SDFs, and parameterizations used in all supported suites can be found in the CCPP v4 Scientific Documentation. Note both suites do not use stochastic physics by default, but the stochastic physics can be activated following the instructions described in the stochastic physics v1 user's guide.

Both the SDF and the *input.nml* contain information about how to specify the physics suite. Some of this information is redundant, and the user must make sure they are compatible. The safest practice is to use the SDF and namelist provided for each suite, since those are supported configurations.

Changes to the SDF must be accompanied by corresponding changes to the namelist. While there is not a one-to-one correspondence between the namelist and the SDF, Table 5.1 shows some variables in the namelist that must match the SDF.

Table 5.1: *Variables related to PBL options*

Namelist option	Meaning	Possible Values	Default	Used with CCPP scheme	Recommentation
	PBL-related variables				
do_myjpbl	Flag to activate the MYJ PBL scheme	T	F	mypbl_wrapper	Set to F for GFSv15p2* and GFSv16beta*
do_myjsfc	Flag to activate the MYJ PBL surface layer scheme	T, F	F	myjsfc_wrapper	Set to F for GFSv15p2* and GFSv16beta*
do_mynnedmf	Flag to activate the MYNN-EDMF scheme	T, F	F	mynnedmf_wrapper	Set to F for GFSv15p2* and GFSv16beta*
do_ysu	Flag to activate the YSU PBL scheme	T, F	F	ysudif	Set to F for GFSv15p2* and GFSv16beta*
hybedmf	Flag to activate the K-based PBL scheme	T, F	F	hedmf	Set to T for GFSv15p2* and GFSv16beta*
isatedmf	Flag for version of scale-aware TKE-based EDMF scheme	0, 1	0	0=satmedmfvdif, 1=satmedmfvdifq	Set to 0 for GFSv15p2* and 1 for GFSv16beta*
ism	Flag to choose a land surface model to use	0, 1, 2	1	1=lsm_noah, 2=lsm_ruc	Set to 1 for GFSv15p2* and GFSv16beta*
satedmf	Flag to activate the scale-aware TKE-based EDMF scheme	T, F	F	satmedmfvdif or satmedmfvdifq	Set to T for GFSv15p2* and GFSv16beta*
shinhong	Flag to activate the Shin-Hong PBL parameterization	T, F	F	shinhongdif	Set to F for GFSv15p2* and GFSv16beta*
	Convection-releated flags				
cscnv	Flag to activate the Chikira-Sugiyama deep convection scheme	T, F	F	cs_conv	Set to F for GFSv15p2* and GFSv16beta*
do_aw	Flag to activate the Arakawa-Wu extension to the Chikira-Sugiyama deep convection scheme	T, F	F	cs_conv_aw_adj	Set to F for GFSv15p2* and GFSv16beta*
imfdeepcnv	Flag to choose a mass flux deep convective scheme	-1, 2, 3, 4	-1	-1=no deep convection*, 2=samfshalcnv, 3=cu_gf_driver, 4=cu_ntiedtke	Set to 2 for GFSv15p2* and GFSv16beta*
imfshalcvn	Flag to choose a mass flux shallow convective scheme	-1, 2, 3, 4	-1	-1=no deep convection*, 2=samfshalcnv, 3=cu_gf_driver, 4=cu_ntiedtke	Set to 2 for GFSv15p2* and GFSv16beta*

*Even when imfdeepcvn=-1, the Chikira-Sugiyama deep convection scheme may be specified using cscnv=T.

Other miscellaneous changes to the SDF that must be accompanied by corresponding changes in the namelist are listed in Table 5.2.

Table 5.2: *Miscellaneous namelist variables and their relation to the SDF*

Namelist option	Meaning	Possible Values	Default	Used with CCPP scheme	Recommendation
	Miscellaneous variables				
do_myjsfc	Flag to activate the MYJ PBL surface scheme	T, F	F	mynnsfc_wrapper	Set to F for GFSv15p2* and GFSv16beta*
do_shoc	Flag to activate the Simplified Higher-Order Closure (SHOC) parameterization	T, F	F	shoc	Set to F for GFSv15p2* and GFSv16beta*
do_ugwp**	Flag to activate the unified Gravity Wave Physics parameterization	T, F	F	cires_ugwp	Set to F for GFSv15p2* and GFSv16beta*
imp_physics	Flag to choose a microphysics scheme	8, 10, 11	99	8=mp_thompson, 10=m_micro, 11=gfdl_cloud_microphysics	Set to 11 for GFSv15p2* and GFSv16beta*
lsm	Flag to choose a land surface model to use	0, 1, 2	1	1=lsm_noah, 2=lsm_ruc	Set to 1 for GFSv15p2* and GFSv16beta*
lsoil	Number of soil layers	4, 9	4	4 for lsm_noah, 9 for lsm_ruc	Set to 4 for GFSv15p2* and GFSv16beta*
h2o_phys	Flag for stratosphere h2o scheme	T, F		h2ophys	Set to T for GFSv15p2* and GFSv16beta*
oz_phys_2015	Flag for new (2015) ozone physics	T, F		ozphys_2015	Set to T for GFSv15p2* and GFSv16beta*

**The CIRES Unified Gravity Wave Physics (cires_ugwp) scheme is used in GFSv15p2* and GFSv16beta* SDFs with do_ugwp=F in the namelist. In this setting, the cires_ugwp calls the operational GFS v15.2 orographic gravity wave drag (gwdps) scheme. When do_ugwp=T, the cires_ugwp scheme calls an experimental orographic gravity wave (gwdps_v0).

Note that some namelist variables are not available for use with CCPP.

- **do_deep**. In order to disable deep convection, it is necessary to remove the deep convection scheme from the SDF.
- **shal_cnv**. In order to disable shallow convection, it is necessary to remove the deep convection scheme from the SDF.
- **ldiag3d** and **ldiag_ugwp**. Must be F for CCPP runs.
- **gwd_opt**. Ignored in CCPP-supported suites.

When certain parameterizations are turned on, additional namelist options can be used (they are ignored otherwise). Some examples are shown in Table 5.3.

Table 5.3: *Enabled namelist variables*

Namelist setting	Enabled namelist variables
do_ugwp=T	All variables in namelist record &cires_ugwp_nml plus do_tofd. Additionally, if namelist variable cnvgwd=T and the third and fourth position of namelist array cdmbgwd are both 1, then the convective gravity wave drag that is part of cires_ugwp will be called. (Not supported with the UFS)
do_mynnedmf=T	bl_mynn_tkeadvect, bl_mynn_edmf, bl_mynn_edmf_mom (Not supported with the UFS)
imp_physics=99	psautco and prautco (Not supported with the UFS)
imp_physics=10	mg_* (Not supported with UFS)
imp_physics=11	All variables in namelist record gfdl_cloud_microphysics_nml and lgfdlmprad
satedmf=T	isatedmf

CHAPTER

SIX

CONTRIBUTING DEVELOPMENT

The ufs-weather-model repository contains the model code and external links needed to build the Unified Forecast System (UFS) atmosphere model and associated components, including the WaveWatch III model. This weather model is used in several of the UFS applications, including the medium-range weather application, the short-range weather application, and the sub-seasonal to seasonal application.

6.1 Making code changes using a forking workflow

If developers would like to make code changes, they need to make a personal fork, set up upstream remote (for merging with the original ufs-weather-model), and create a branch for ufs-weather-model and each of the subcomponent repositories they want to change. They can then make code changes, perform testing and commit the changes to the branch in their personal fork. It is suggested that they update their branch by merging the develop branch with the develop branch of the original repositories periodically to get the latest updates and bug fixes.

If developers would like to get their code committed back to the original repository, it is suggested to follow the steps below:

1. Create an issue in the authoritative repository. For example to commit code changes to fv3atm, please go to https://github.com/NOAA-EMC/fv3atm, under NOAA-EMC/fv3atm and find the "Issues" tab next to the "Code" tab. Click on "Issues" and a new page will appear. On the right side of the page, there is a green "New issue" button. Clicking on that will lead to a new issue page. Fill out the title, comments to describe the code changes, and also please provide personal fork and branch information. Lastly, click on the "Submit new issue" button, so that the new issue is created.

2. When the development is mature, tests have been conducted, and the developer is satisfied with the results, create a pull request to commit the code changes.

    - Merge developer's branch to the latest ufs-weather-model develop branch in authoritative repository. If changes are made in model sub-components, developers need to merge their branches to branches with the corresponding authoritative repository (or original repository for some components). For this, code management practices of the subcomponents need to be followed.

    - Regression tests associated with the ufs-weather-model are available on Tier 1 and Tier 2 platforms as described in https://github.com/ufs-community/ufs-weather-model/wiki/Regression-Test-Policy-for-Weather-Model-Platforms-and-Compilers. If the developer has access to these platforms, the developer should pass the regression test on at least one supported platform. If the developer does not have access to these platforms, this should be stated in the PR so the code manager(s) can conduct the tests.

    - For each component branch where developers make changes, developers need to go to their personal fork on GitHub and click on the "New pull request" button. When a new page "Compare changes" appears, developers will choose the branch in their fork with code changes to commit and the branch in upstream repository that the changes will be committed to. Also developers in the commit comment must add the

github issue title and number created in 1) in the comment box. The code differences between the two branches will be displayed. Developers can review the differences and click on "submit pull request" to make the pull request. After code changes are committed to the component repository, developers will make pull requests to ufs-weather-model repository.

3. When PRs are created, the creator must temporarily modify .gitmodules to point to his/her fork and branch if updates are required for submodules.
4. Merging code from PRs with submodules requires coordination with the person making the PRs. From the "innermost" nested PR up to the top-level PR, the PRs need to be merged as-is. After each merge, the person creating the PRs has to update his/her local code to check out the merged version, revert the change to .gitmodules, and push this to GitHub to update the PR. And so on and so forth.
5. Checking out the code ufs_release_1.0 should always be as follows:

```
git clone https://github.com/ufs-community/ufs-weather-model
cd ufs-weather-model
git checkout ufs_release_1.00
git submodule update --init --recursive
```

6. Checking out a PR with id ID for testing it should always be as follows:

```
git clone https://github.com/ufs-community/ufs-weather-model
cd ufs-weather-model
git fetch origin pull/ID/head:BRANCHNAME
git checkout BRANCHNAME
git submodule update --init --recursive
```

It is suggested that the developers inform all the related code managers as the hierarchy structure of the ufs-weather-model repository may require collaboration among the code managers.

6.2 Engaging in the code review process

When code managers receive a pull request to commit the code changes, it is recommended that they add at least two code reviewers to review the code and at least one of the reviewers has write permission. The reviewers will write comments about the code changes and give a recommendation as to whether the code changes can be committed. What kinds of code changes will be accepted in the repository is beyond the scope of this document; future ufs-weather-model code management documents may have a detailed answer for that.

Reviewers may suggest some code changes during the review process. Developers need to respond to these comments in order to get code changes committed. If developers make further changes to their branch, reviewers need to check the code changes again. When both reviewers give recommendation to commit the code, code managers will merge the changes into the repository.

6.3 Conducting regression tests

Only developers using Tier 1 and 2 platforms can run the ufs-weather-model regression tests. Other developers need to work with the code managers to assure completion of the regression tests.

To run regression test using rt.sh

rt.sh is a bash shell file to run the RT and has the following options:

```
Usage: ./rt.sh -c | -f | -s | -l <file> | -m | -k | -r | -e | -h
-c create new baseline results for <model>
-f run full suite of regression tests
-s run standard suite of regression tests
-l run test specified in <file>
-m compare against new baseline results
-k  keep run directory (automatically deleted otherwise if all tests pass)
-r use Rocoto workflow manager
-e use ecFlow workflow manager
-h display this help
```

```
% cd ufs-weather-model/tests
% ./rt.sh -f
```

This command can only be used on platforms that have been configured for regression testing (Tier 1 and Tier 2 platforms as described in https://github.com/ufs-community/ufs-weather-model/wiki/Regression-Test-Policy-for-Weather-Model-Platforms-and-Compilers). For information on testing the CCPP code, or using alternate computational platforms, see the following sections.

This command and all others below produce log output in ./tests/log_machine.compiler. These log files contain information on the location of the run directories that can be used as templates for the user. Each rt*.conf contains one or more compile commands preceding a number of tests.

Regression test log files (ufs-weather-model/tests/Compile_$(MACHINE_ID).log and ufs-weather-model/tests/RegressionTests_$(MACHINE_ID).log) will be updated.

If developers wish to contribute code that changes the results of the regression tests (because of updates to the physics, for example), it is useful to run rt.sh as described above to make sure that the test failures are as expected. It is then useful to establish a new personal baseline:

./rt.sh -l rt.conf -c # create own reg. test baseline

Once the personal baseline has been created, future runs of the RT should be compared against the personal baseline using the -m option.

./rt.sh -l rt.conf -m # compare against own baseline

To create new baseline:

```
% cd ufs-weather-model/tests
% ./rt.sh -f -c
```

An alternative/complementary regression test system is using NEMSCompsetRun, which focuses more on coupled model configurations than testing features of the standalone ufs-weather-model. To run regression test using NEMSCompsetRun:

```
% cd ufs-weather-model
% ./NEMS/NEMSCompsetRun -f
```

Regression test log files (ufs-weather-model/log/$MACHINE_ID/*) will be updated.

To create new baseline:

```
% cd ufs-weather-model
% ./NEMS/NEMSCompsetRun --baseline fv3 --platform=${PLATFORM}
```

The value of ${PLATFORM} can be found in ufs-weather-model/compsets/platforms.input.

Developers need to commit the regression test log files to their branch before making pull request.

CHAPTER

SEVEN

ACRONYMS

Acronyms	Explanation
AOML	NOAA's Atlantic Oceanographic and Meteorological Laboratory
API	Application Programming Interface
b4b	Bit-for-bit
CCPP	Common Community Physics Package
dycore	Dynamical core
EDMF	Eddy-Diffusivity Mass Flux
EMC	Environmental Modeling Center
ESMF	The Earth System Modeling Framework
ESRL	NOAA Earth System Research Laboratories
FMS	Flexible Modeling System
FV3	Finite-Volume Cubed Sphere
GFDL	NOAA Geophysical Fluid Dynamics Laboratory
GFS	Global Forecast System
GSD	Global Systems Division
HTML	Hypertext Markup Language
LSM	Land Surface Model
MPI	Message Passing Interface
NCAR	National Center for Atmospheric Research
NCEP	National Centers for Environmental Predicction
NEMS	NOAA Environmental Modeling System
NOAA	National Oceanic and Atmospheric Administration
NSSL	National Severe Storms Laboratory
PBL	Planetary Boundary Layer
PR	Pull request
RRTMG	Rapid Radiative Transfer Model for Global Circulation Models
RT	Regression test
SAS	Simplified Arakawa-Schubert
SDF	Suite Definition File
sfc	Surface
SHUM	Perturbed boundary layer specific humidity
SKEB	Stochastic Kinetic Energy Backscatter
SPPT	Stochastically Perturbed Physics Tendencies
TKE	Turbulent Kinetic Energy
UFS	Unified Forecast System
WM	Weather Model

CHAPTER

EIGHT

GLOSSARY

CCPP Model agnostic, vetted, collection of codes containing atmospheric physical parameterizations and suites for use in NWP along with a framework that connects the physics to host models

CCPP-Framework The infrastructure that connects physics schemes with a host model; also refers to a software repository of the same name

CCPP-Physics The pool of CCPP-compliant physics schemes; also refers to a software repository of the same name

FMS The Flexible Modeling System (FMS) is a software framework for supporting the efficient development, construction, execution, and scientific interpretation of atmospheric, oceanic, and climate system models.

NEMS The NOAA Environmental Modeling System - a software infrastructure that supports NCEP/EMC's forecast products.

NUOPC The National Unified Operational Prediction Capability is a consortium of Navy, NOAA, and Air Force modelers and their research partners. It aims to advance the weather modeling systems used by meteorologists, mission planners, and decision makers. NUOPC partners are working toward a common model architecture - a standard way of building models - in order to make it easier to collaboratively build modeling systems.

Parameterization or physics scheme The representation, in a dynamic model, of physical effects in terms of admittedly oversimplified parameters, rather than realistically requiring such effects to be consequences of the dynamics of the system (AMS Glossary)

Suite Definition File (SDF) 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

Suite A collection of primary physics schemes and interstitial schemes that are known to work well together

UFS A Unified Forecast System (UFS) is a community-based, coupled comprehensive Earth system modeling system. The UFS numerical applications span local to global domains and predictive time scales from sub-hourly analyses to seasonal predictions. It is designed to support the Weather Enterprise and to be the source system for NOAA's operational numerical weather prediction applications

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 be an atmospheric model coupled with one or more additional components, such as a wave or ocean model.

INDEX