UFS Weather Model Users Guide

Feb 23, 2021

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 v2.0 is the latest 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.

The UFS Weather Model is currently included in two UFS Application releases. These UFS Apps also contain pre- and post-processing components, a comprehensive build system, and workflows for configuration and execution of the application.

The UFS WM v2.0 is included as part of the UFS Short Range Weather App, and details can be found here.

The UFS WM v1.1 and v1.0 is included as part of the UFS Medium Range Weather App, and details can be found here.

The UFS WM v2 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: 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, the *cellular automata* stochastic scheme, and the use of different file formats for input and output. It is expected that the UFS WM supported capabilities will be expanded in future releases.

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 the UFS WM GitHub wiki 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 (FAQ) lists frequently asked questions and answers.

Finally, Chapters 6 and 7 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-GFDL/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/NOAA-EMC/fv3atm
│   ├── atmos_cubed_sphere               https://github.com/NOAA-GFDL/GFDL_atmos_
↪cubed_sphere
│   ├── ccpp
│   │   ├── framework                    https://github.com/NCAR/ccpp-framework
│   │   ├── physics                      https://github.com/NCAR/ccpp-physics
├── NEMS                                 https://github.com/NOAA-EMC/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 v2.0.0 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-v2.0.0
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 NCEPLIBS, NCEPLIBS-external and CMake

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 from the *ufs-weather-model* directory on hera:

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

Note that loading this module file will also set the CMake environment variables shown in Table 3.3.

Table 3.3: *CMake environment variables required to configure the build for the Weather Model*

EnvironmentVariable	Description	Hera Intel Value
CMAKE_C_COMPILER	Name of C compiler	mpiicc
CMAKE_CXX_COMPILER	Name of C++ compiler	mpiicpc
CMAKE_Fortran_COMPILER	Name of Fortran compiler	mpiifort
CMAKE_Platform	String containing platform and compiler name	hera.intel

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-v2.0.0/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 the CCPP_SUITES environment variable

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 set to `'FV3_GFS_v15p2'` in `build.sh`.

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 configures the build environment and runs the `make` command. This script also checks that all necessary environment variables have been set.

If any of the environment variables have not been set, the `build.sh` script will exit with a message similar to:

```
./build.sh: line 11: CMAKE_Platform: Please set the CMAKE_Platform environment␣
↪variable, e.g. [macosx.gnu|linux.gnu|linux.intel|hera.intel|...]
```

The WM can be built by running the following command 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, initial and boundary conditions, and model configuration files (such as namelists).

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

The static input files for global configurations are listed and described in Table 4.1. Similar files are used for a regional grid but are grid specific and generated by pre-processing utilities.

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 for global configurations are listed and described in Table 4.2. The input files for a limited area model (LAM) configuration including grid information and initial and lateral boundary conditions are listed and described in Table 4.3. Note that the regional grid is referred to as Tile 7 here, and are generated by several pre-processing utilities.

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

Filename	Description	Date-dependent
Cxx_grid.tile[1-6].nc	Cxx grid information for tiles 1-6, where 'xx' is the grid number	
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	✓

Table 4.3: *Regional input files containing grid information and initial and lateral boundary conditions for regional configurations*

Filename	Description	Date-dependent
Cxx_grid.tile7.nc	Cxx grid information for tile 7, where 'xx' is the grid number	
gfs_ctrl.nc	NCEP NGGPS tracers, ak, and bk	✓
gfs_bndy.tile7.HHH.nc	Lateral boundary conditions at hour HHH	✓
gfs_data.tile7.nc	Initial condition fields (ps, u, v, u, z, t, q, O3). May include spfo3, spfo, spf02 if multiple gases are used	✓
oro_data.tile7.nc	Model terrain (topographic/orographic information) for grid tile 7	
sfc_ctrl.nc	Control parameters for surface input: forecast hour, date, number of soil levels	
sfc_data.tile7.nc	Surface properties for grid tile 7	✓

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

While the *input.nml* file is also a configuration file used by the UFS Weather Model, it is described in Section 4.1.4. The run-time configuration of model output fields is controlled by the combination of *diag_table* and *model_configure*, and is described in detail in Section 4.2.

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 to 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 4.4.

Table 4.4: *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 4.5.

Table 4.5: *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 4.6. 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 4.6: *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.

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 4.7 shows the following parameters that can be set in *model_configure* at run-time.

Table 4.7: *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 4.8 shows the following parameters in *model_configure* that are not usually changed.

Table 4.8: *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 for the UFS Medium Range Weather App configuration: *suite_FV3_GFS_v15p2.xml* and *suite_FV3_GFS_v16beta.xml*.

There are two SDFs currently supported for the UFS Short Range Weather App configuration: *suite_FV3_GFS_v15p2.xml* and *suite_FV3_RRFS_v1alpha.xml*.

Detailed descriptions of the supported suites can be found with the CCPP v5.0.0 Scientific Documentation.

4.1.4 Namelist file `input.nml`

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 provides an in depth description of the namelist settings:

https://dtcenter.ucar.edu/GMTB/v5.0.0/sci_doc/

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

https://dtcenter.ucar.edu/GMTB/v5.0.0/sci_doc/CCPPsuite_nml_desp.html

The following link describes the stochastic physics namelist records:

https://stochastic-physics.readthedocs.io/en/ufs-v1.1.0/namelist_options.html

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

https://noaa-emc.github.io/FV3_Dycore_ufs-v1.1.0/html/index.html

The namelist section `&interpolator_nml` is not used in this release, and any modifications to it will have no effect on the model results.

fms_io_nml

The namelist section `&fms_io_nml` of `input.nml` contains variables that control reading and writing of restart data in netCDF format. There is a global switch to turn on/off the netCDF restart options in all of the modules that read or write these files. The two namelist variables that control the netCDF restart options are `fms_netcdf_override` and `fms_netcdf_restart`. The default values of both flags are .true., so by default, the behavior of the entire model is to use netCDF IO mode. To turn off netCDF restart, simply set `fms_netcdf_restart` to .false.. The namelist variables used in `&fms_io_nml` are described in Table 4.9.

Table 4.9: *Description of the &fms_io_nml namelist section.*

Variable Name	Description	Data Type	Default Value
fms_netcdf_override	If true, `fms_netcdf_restart` overrides the individual `do_netcdf_restart` value. If false, individual module settings has a precedence over the global setting, therefore `fms_netcdf_restart` is ignored.	logical	.true.
fms_netcdf_restart	If true, all modules using restart files will operate under netCDF mode. If false, all modules using restart files will operate under binary mode. This flag is effective only when `fms_netcdf_override` is .true. When `fms_netcdf_override` is .false., individual module setting takes over.	logical	.true.
threading_read	Can be 'single' or 'multi'	character(len=32)	'multi'
format	Format of restart data. Only netCDF format is supported in fms_io.	character(len=32)	'netcdf'
read_all_pe	Reading can be done either by all PEs (default) or by only the root PE.	logical	.true.
iospec_ieee32	If set, call mpp_open single 32-bit ieee file for reading.	character(len=64)	'-N ieee_32'
max_files_w	Maximum number of write files	integer	40
max_files_r	Maximum number of read files	integer	40
time_stamp_restart	If true, `time_stamp` will be added to the restart file name as a prefix.	logical	.true.
print_chksum	If true, print out chksum of fields that are read and written through save_restart/restore_state.	logical	.false.
show_open_namelist_file_warning	Flag to warn that open_namelist_file should not be called when INTERNAL_FILE_NML is defined.	logical	.false.
debug_mask_list	Set `debug_mask_list` to true to print out mask_list reading from mask_table.	logical	.false.
checksum_required	If true, compare checksums stored in the attribute of a field against the checksum after reading in the data.	logical	.true.

This release of the UFS Weather Model sets the following variables in the `&fms_io_nml` namelist:

```
&fms_io_nml
  checksum_required = .false.
  max_files_r = 100
  max_files_w = 100
/
```

namsfc

The namelist section `&namsfc` contains the filenames of the static datasets (i.e., *fix files*). Table 4.1 contains a brief description of the climatological information in these files. The variables used in `&namsfc` to set the filenames are described in Table 4.10.

Table 4.10: *List of common variables in the *namsfc* namelist section used to set the filenames of static datasets.**

Variable Name	File contains	Data Type	Default Value
fnglac	Climatological glacier data	character*500	'global_glacier.2x2.grb'
fnmxic	Climatological maximum ice extent	character*500	'global_maxice.2x2.grb'
fntsfc	Climatological surface temperature	character*500	'global_sstclim.2x2.grb'
fnsnoc	Climatological snow depth	character*500	'global_snoclim.1.875.grb'
fnzorc	Climatological surface roughness	character*500	'global_zorclim.1x1.grb'
fnalbc	Climatological snowfree albedo	character*500	'global_albedo4.1x1.grb'
fnalbc2	Four albedo fields for seasonal mean climatology	character*500	'global_albedo4.1x1.grb'
fnaisc	Climatological sea ice	character*500	'global_iceclim.2x2.grb'
fntg3c	Climatological deep soil temperature	character*500	'global_tg3clim.2.6x1.5.grb'
fnvegc	Climatological vegetation cover	character*500	'global_vegfrac.1x1.grb'
fnvetc	Climatological vegetation type	character*500	'global_vegtype.1x1.grb'
fnsotc	Climatological soil type	character*500	'global_soiltype.1x1.grb'
fnsmcc	Climatological soil moisture	character*500	'global_soilmcpc.1x1.grb'
fnmskh	High resolution land mask field	character*500	'global_slmask.t126.grb'
fnvmnc	Climatological minimum vegetation cover	character*500	'global_shdmin.0.144x0.144.grb'
fnvmxc	Climatological maximum vegetation cover	character*500	'global_shdmax.0.144x0.144.grb'
fnslpc	Climatological slope type	character*500	'global_slope.1x1.grb'
fnabsc	Climatological maximum snow albedo	character*500	'global_snoalb.1x1.grb'

A sample subset of this namelist is shown below:

```
&namsfc
  FNGLAC   = 'global_glacier.2x2.grb'
  FNMXIC   = 'global_maxice.2x2.grb'
  FNTSFC   = 'RTGSST.1982.2012.monthly.clim.grb'
  FNSNOC   = 'global_snoclim.1.875.grb'
  FNZORC   = 'igbp'
  FNALBC   = 'global_snowfree_albedo.bosu.t126.384.190.rg.grb'
  FNALBC2  = 'global_albedo4.1x1.grb'
  FNAISC   = 'CFSR.SEAICE.1982.2012.monthly.clim.grb'
  FNTG3C   = 'global_tg3clim.2.6x1.5.grb'
  FNVEGC   = 'global_vegfrac.0.144.decpercent.grb'
  FNVETC   = 'global_vegtype.igbp.t126.384.190.rg.grb'
  FNSOTC   = 'global_soiltype.statsgo.t126.384.190.rg.grb'
  FNSMCC   = 'global_soilmgldas.t126.384.190.grb'
  FNMSKH   = 'seaice_newland.grb'
  FNVMNC   = 'global_shdmin.0.144x0.144.grb'
  FNVMXC   = 'global_shdmax.0.144x0.144.grb'
  FNSLPC   = 'global_slope.1x1.grb'
  FNABSC   = 'global_mxsnoalb.uariz.t126.384.190.rg.grb'
/
```

Additional variables for the `&namsfc` namelist can be found in the `FV3/ccpp/physics/physics/sfcsub.F` file.

atmos_model_nml

The namelist section `&atmos_model_nml` contains information used by the atmosphere model. The variables used in `&atmos_model_nml` are described in Table 4.11.

Table 4.11: *List of common variables in the *atmos_model_nml* namelist section.*

Variable Name	Description	Data Type	Default Value
blocksize	Number of columns in each `block` sent to the physics. OpenMP threading is done over the number of blocks. For best performance this number should divide the number of grid cells per processor: `((npx-1)*(npy-1)/(layout\_x)*(layout\_y))`. A description of these variables is provided here.	integer	1
chksum_debug	If true, compute checksums for all variables passed into the GFS physics, before and after each physics timestep. This is very useful for reproducibility checking.	logical	.false.
dycore_only	If true, only the dynamical core (and not the GFS physics) is executed when running the model, essentially running the model as a solo dynamical core.	logical	.false.
debug	If true, turn on additional diagnostics for the atmospheric model.	logical	.false.
sync	If true, initialize timing identifiers.	logical	.false.
fdiag	Array with dimension `maxhr` = 4096 listing the diagnostic output times (in hours) for the GFS physics. This can either be a list of times after initialization, or an interval if only the first entry is nonzero. The default setting of 0 will result in no outputs.	real	0.
fhmax	The maximal forecast time for output.	real	384.0
fhmaxhf	The maximal forecast hour for high frequency output.	real	120.0
fhout	Output frequency during forecast time from 0 to `fhmax`, or from `fhmaxhf` to `fhmax` if `fhmaxf>0`.	real	3.0
fhouthf	The high frequency output frequency during the forecast time from 0 to `fhmaxhf` hour.	real	1.0
ccpp_suite	Name of the CCPP physics suite	character(len=256)	FV3_GFS_v15p2, set in `build.sh`
avg_max_length	Forecast interval (in seconds) determining when the maximum values of diagnostic fields in FV3 dynamics are computed.	real	3600.

A sample of this namelist is shown below:

```
&atmos_model_nml
  blocksize = 32
  chksum_debug = .false.
  dycore_only = .false.
  fdiag = 1
  fhmax = 384
  fhout = 3
  fhmaxhf = 120
  fhouthf = 1
```

(continues on next page)

(continued from previous page)

```
  ccpp_suite = 'FV3_GFS_v16beta'
/
```

The namelist section relating to the FMS diagnostic manager `&diag_manager_nml` is described in Section 4.3.1.

4.2 Output files

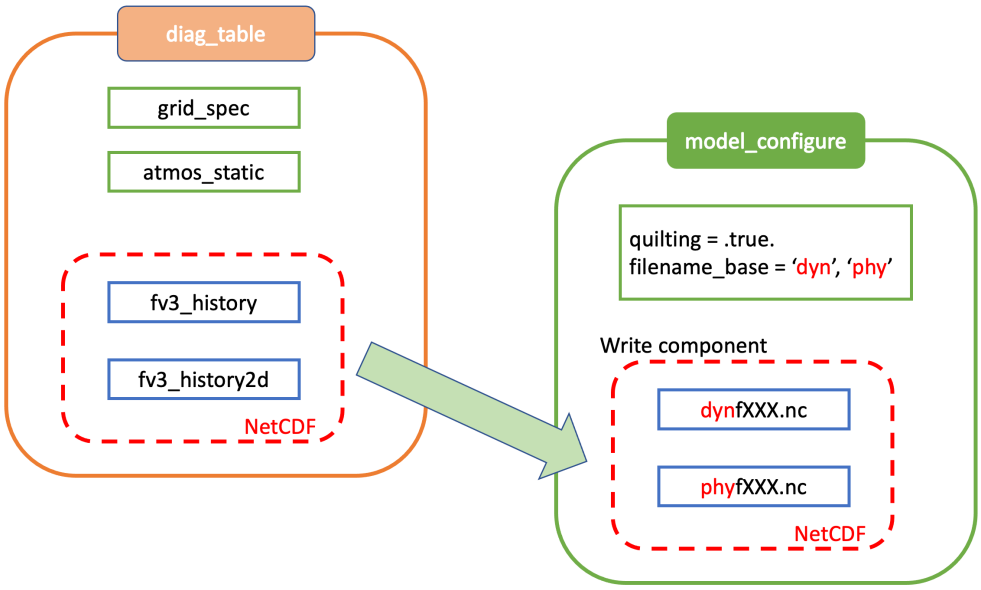
The output files generated when running *fv3.exe* are defined in the *diag_tabl* file. For the default global configuration, the following files are output (six files of each kind, corresponding to the six tiles of the model grid):

- *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 (format, projection, etc) may be overridden in the *model_configure* input file, see Section 4.1.3.

The regional configuration will generate similar output files, but the *tile[1-6]* is not included in the filename.

Two files (*model_configure* and *diag_table*) control the output that is generated by the UFS Weather Model. The output files that contain the model variables are written to a file as shown in the figure below. The format of these output files is selected in *model_configure* as NetCDF or nemsio. The information in these files may be remapped, augmented with derived variables, and converted to GRIB2 by the Unified Post Processor (UPP). Model variables are listed in the *diag_table* in two groupings, *fv3_history* and *fv3_history2d*, as described in Section 4.1.3. The names of the files that contain these model variables are specified in the *model_configure* file. When *quilting* is set to *.true.* for the write component, the variables listed in the groups *fv3_history* and *fv3_history2d* are converted into the two output files named in the *model_configure* file, e.g. *atmfHHH.* and *sfcfHHH.*. The bases of the file names (*atm* and *sfc*) are specified in the *model_configure* file, and *HHH* refers to the forecast hour.



Standard output files are *logfHHH* (one per forecast hour), 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 *PETnnn.ESMF_LogFile* (one per MPI task).

Additional output files include: nemsusage.xml, a timing log file; time_stamp.out, contains the model init time; *RESTART/*nc*, files needed for restart runs.

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 4.12. See `FMS/diag_manager/diag_manager.F90` for the complete list.

Table 4.12: *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.
/
```

4.4 Additional Information about the Write Component

The UFS Weather Model is built using the Earth System Modeling Framework (ESMF). As part of this framework, the output history files written by the model use an ESMF component, referred to as the *write component*. This model component is configured with settings in the model_configure file, as described in Section 4.1.3. By using the ESMF capabilities, the write component can generate output files in several different formats and several different map projections. For example, a Gaussian global grid in NEMSIO format, or a native grid in NetCDF format. The write component also runs on independent MPI tasks, and so the computational tasks can continue while the write component completes its work asynchronously. The configuration of write component tasks, balanced with compute tasks, is part of the tuning for each specific application of the model (HPC, write frequency, i/o speed, model domain, etc). For the global grid, if the write component is not selected (quilting=.false.), the FV3 code will write tiled output in the native projection in NetCDF format. The regional grid requires the use of the write component.

CHAPTER

FIVE

FAQ

5.1 How do I build and run a single test of the UFS Weather Model?

An efficient way to build and run the UFS Weather Model is to use the regression test (`rt.sh`). This script is widely used by model developers on Tier 1 and 2 platforms and is described in the UFS WM GitHub wiki. The advantages to this approach are:

- It does not require a workflow, pre- or post-processing steps.
- The batch submission script is generated.
- Any required input data is already available for machines used by the regression test.
- Once the `rt.sh` test completes, you will have a working copy in your run directory where you can make modifications to the namelist and other files, and then re-run the executable.

The steps are:

1. Clone the source code and all the submodules as described in Section 3.2, then go into the `tests` directory:

```
cd ufs-weather-model (or the top level where you checked out the code)
cd tests
```

2. Find a configure (`*.conf`) file that contains the machine and compiler you are using. For this example, the Intel compiler on Cheyenne is used. To create a custom configure file, two lines are needed: a `COMPILE` line and a `RUN` line. The `COMPILE` line should contain the name of the machine and compiler `cheyenne.intel` and the desired `SUITES` for the build. Choose a `RUN` line under this `COMPILE` command that uses the desired `SUITE`. For example:

```
COMPILE | 32BIT=Y CCPP=Y STATIC=Y SUITES=FV3_GFS_v15p2,FV3_GFS_v16beta,FV3_GFS_
↪v15p2_no_nsst,FV3_GFS_v16beta_no_nsst                    | standard    |␣
↪cheyenne.intel | fv3
RUN     | fv3_ccpp_gfs_v16beta
↪                                                         | standard    |
↪      | fv3         |
```

   Put these two lines into a file called `my_test.conf`. The parameters used in this run can be found in the `fv3_ccpp_gfs_v16beta` file in the `ufs-weather-model/tests/tests` directory.

   Note: These two lines are long and may not appear in entirety in your browser. Scroll to the right to see the entire line.

3. Modify the `rt.sh` script to put the output in a run directory where you have write permission:

```
if [[ $MACHINE_ID = cheyenne.* ]]; then stanza:
...
dprefix=/glade/scratch
```

This works for Cheyenne, since `$USER/FV3_RT` will be appended. Also check that `RTPWD` points to a diretory that exists:

```
if [[ $MACHINE_ID = cheyenne.* ]]; then
  RTPWD=${RTPWD:-$DISKNM/ufs-public-release-20200224/${COMPILER^^}}
```

4. Run the `rt.sh` script from the `tests` directory:

```
./rt.sh -k -l my_test.conf >& my_test.out &
```

Check `my_test.out` for build and run status, plus other standard output. Check `/glade/scratch/$USER/FV3_RT/rt_PID` for the model run, where `PID` is a process ID. The build will take about 10-15 minutes and the run will be fast, depending on how long it waits in the queue. A message `"REGRESSION TEST WAS SUCCESSFUL"` will be written to this file, along with other entertainment: `'Elapsed time: 00h:14m:12s. Have a nice day!'`.

5. When the build and run are complete, modify the namelist or `model_configure` files and re-run by submitting the `job_card` file:

```
qsub job_card
```

5.2 How do I change the length of the model run?

In your run directory, there is a file named `model_configure`. Change the variable `nhours_fcst` to the desired number of hours.

5.3 How do I select the file format for the model output (netCDF or NEMSIO)?

In your run directory, there is a file named `model_configure`. Change the variable `output_file` to `'netcdf'` or `'nemsio'`. The variable `output_file` is only valid when the write component is activated by setting `quilting` to .true. in the `model_configure` file.

5.4 How do I set the output history interval?

The interval at which output (history) files are written is controlled in two places, and depends on whether you are using the write component to generate your output files. Table 5.1 describes the relevant variables. If the write_component is used, then the variables listed as *model_configure* are required. It is however, also required that the settings in *input.nml* match those same settings in *model_configure*. If these settings are inconsistent, then unpredictable output files and intervals may occur!

Table 5.1: *Namelist variables used to control the output file frequency.*

Namelist variable	Location	Default Value	Description
fdiag	input.nml	0	Array with dimension `maxhr` = 4096 listing the diagnostic output times (in hours) for the GFS physics. This can either be a list of times after initialization, or an interval if only the first entry is nonzero. The default setting of 0 will result in no outputs.
fhmax	input.nml	384	The maximal forecast time for output.
fhmaxhf	input.nml	120	The maximal forecast hour for high frequency output.
fhout	input.nml	3	Output frequency during forecast time from 0 to `fhmax`, or from `fhmaxhf` to `fhmax` if `fhmaxf>0`.
fhouthf	input.nml	1	The high frequency output frequency during the forecast time from 0 to `fhmaxhf` hour.
nfhmax_hf	model_configure	0	forecast length of high history file
nfhout_hf	model_configure	1	high history file output frequency
nfhout	model_configure	3	history file output frequency

5.5 How do I set the total number of tasks for my job?

The total number of MPI tasks used by the UFS Weather Model is a combination of compute and quilt tasks, and can be calculated using the following relationship:

- total tasks = compute tasks + quilt tasks
- compute tasks = x layout * y layout * number of tiles
- quilt tasks = write_groups * write_tasks_per_group if quilting==.true.

The layout and tiles settings are in `input.nml`, and the quilt task settings are in `model_configure`

CHAPTER

SIX

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
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

SEVEN

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