UFS Weather Model Users Guide

Feb 23, 2024

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). In addition to its use in NOAA's operational forecast systems, the UFS WM is the atmospheric model used in public UFS application releases, such as the Short-Range Weather (SRW) Application v2.2.0 release. These releases represent a snapshot of a continuously evolving system undergoing open development. More information about the UFS can be found on the UFS Community Portal at https://ufscommunity.org/ and on the Earth Prediction Innovation Center (EPIC) website at https://epic.noaa.gov/get-code/ufs-weather-model/.

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 is the computational part of an atmospheric model that solves the equations of fluid motion.
- 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. It is used for functions such as parallelization.
- The Common-Community Physics Package (*CCPP*), provides a framework and library of physics schemes, or *parameterizations*, that support interoperable atmospheric physics. Atmospheric physics is a set of numerical methods approximating the effects of small-scale processes such as clouds, turbulence, radiation, and their interactions.
- Stochastic physics schemes apply randomized perturbations to the physical tendencies, or physical parameters, of a model in order to compensate for model uncertainty. They include the Stochastic Kinetic Backscatter Scheme (SKEBS), the Stochastically Perturbed Parameterization Tendencies (SPPT) scheme, the perturbed boundary layer humidity (SHUM) scheme, the Stochastically Perturbed Parameterizations (SPP) scheme, Land Surface Model SPP (LSM-SPP), and the cellular automata method (Bengtsson *et al.* [BDT+20]).
- The libraries needed to build the system, which are bundled together via spack-stack and include:
  - 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: The UFS Short-Range Weather (*SRW*) Application v2.2.0 release (October 2023) and the UFS Medium Range Weather Application (*MRW*) v1.1.0 release (October 2020). These UFS Apps also contain pre- and post-processing components, a comprehensive build system, and workflows for configuration and execution of the application. The SRW App v2.2.0 documentation and details can be found here. The MRW App v1.1.0 documentation and details can be found here.

The UFS WM code is portable and can be used with Linux or Mac operating systems and with Intel or GNU compilers. It has been tested on a variety of platforms widely used by atmospheric scientists, such as the NOAA Research Hera system, the National Center for Atmospheric Research (*NCAR*) Derecho 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 (Data: Input, Model Configuration, and Output Files) lists the model inputs and outputs and has a description of the key files.
- Chapter 6 (Configuration Parameters) lists the purpose and valid values for various configuration parameters.
- Chapter 7 (Automated Testing) describes UFS WM automated testing options.
- Chapter 8 (FAQ) lists frequently asked questions and answers.

Finally, Chapters 9 and 10 contain a list of acronyms and a glossary, respectively.

CHAPTER

TWO

TECHNICAL OVERVIEW

2.1 Supported Platforms and Compilers for Running the UFS Weather Model

Four levels of support have been defined for *UFS* applications, and the UFS Weather Model (*WM*) operates under this paradigm:

- **Level 1** *(Preconfigured)*: Prerequisite software libraries are pre-built and available in a central location; code builds and runs; full testing of model.
- **Level 2** *(Configurable)*: Prerequisite libraries are not available in a centralized location but are expected to install successfully; code builds and runs; full testing of model.
- **Level 3** *(Limited-test platforms)*: Libraries and code build on these systems, but there is limited testing with running the model.
- **Level 4** *(Build-only platforms)*: Libraries and code build, but running the model is not tested.

2.1.1 Level 1 Systems

Preconfigured (Level 1) systems for the UFS WM already have the required external libraries available in a central location via *spack-stack*. The WM is expected to build and run out-of-the-box on these systems, and users can download the WM code without first installing prerequisite software. Additionally, regression test data is already available on these systems. In general, users must have access to these Level 1 systems in order to use them.

Currently, Level 1 (or Tier-1) platforms for regression testing are:

- WCOSS2 (Intel)
- Gaea (Intel)
- Hera (Intel/GNU compilers)
- Jet (Intel)
- Orion (Intel)
- Hercules (Intel/GNU compilers)
- AWS Docker container (Intel)

More information is available in the UFS WM wiki.

2.1.2 Level 2-4 Systems

On non-Level 1 platforms, users must install the required libraries before building the UFS WM. Additionally, users must stage the required data in order to run regression tests. Once the prerequisite libraries are installed, and the data has been staged, the WM should build and run successfully. However, users may need to perform additional troubleshooting on Level 3 or 4 systems since little or no testing is conducted on these systems.

Currently, Level 2 platforms for regression testing are:

- S4 (Intel)

2.2 UFS Weather Model Hierarchical Repository Structure

The UFS *WM* repository supports the *UFS* short- and medium-range weather applications (*SRW* / *MRW* Apps). The WM repository contains atmosphere, ocean, sea ice, land, and wave components, as well as some infrastructure components. Each of these subcomponents has its own repository. All the repositories are currently located in GitHub with public access to the broader community. Table 2.1 describes the list of repositories that comprise the UFS WM.

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
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 (FV3)	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-PSL/stochastic_physics
Modular Ocean Model (*MOM6*)	https://github.com/NOAA-EMC/MOM6
HYbrid Coordinate Ocean Model (*HYCOM*)	https://github.com/NOAA-EMC/HYCOM-src
Los Alamos sea ice model (*CICE6*)	https://github.com/NOAA-EMC/CICE
NOAA/NCEP WAVEWATCH III Model (*WW3*)	https://github.com/NOAA-EMC/WW3
The Goddard Chemistry Aerosol Radiation and Transport (*GOCART*)	https://github.com/GEOS-ESM/GOCART
NUOPC Community Mediator for Earth Prediction Systems (*CMEPS*)	https://github.com/NOAA-EMC/CMEPS
Community Data Models for Earth Prediction Systems (*CDEPS*)	https://github.com/NOAA-EMC/CDEPS
Air Quality Model (*AQM*)	https://github.com/NOAA-EMC/AQM
Noah-MP Land Surface Model (Noah-MP)	https://github.com/NOAA-EMC/noahmp

In the table, the left-hand column contains a description of each repository, and the right-hand column shows the GitHub location of the authoritative component repositories. The UFS WM currently uses Git submodules to manage these subcomponents.

2.3 Directory Structure

The umbrella repository for the UFS WM 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 basic directory structure will be similar to the example below. Files and some directories have been removed for brevity. Directories in parentheses will appear only after a recursive clone or submodule update (`git submodule update --init --recursive`).

```
ufs-weather-model
 ├── AQM
 │     └── (src)
 │         ├── (model)
 │            └── (CMAQ)                      -------- EPA Air Quality Model
 ├── build.sh                                 -------- script for building the WM
 ├── CDEPS-interface
 │     └── CDEPS
 │         ├── (datm)                         -------- CDEPS DATM
 │         └── (docn)                         -------- CDEPS DOCN
 ├── CICE-interface
 │    └── CICE                                -------- CICE6 sea ice model
 │        ├── (icepack)                       -------- Sea ice column physics
 │        └── (cicecore/drivers/nuopc/cmeps)  -------- NUOPC CICE6 cap
 ├── cmake                                    -------- cmake configuration files
 ├── CMakeLists.txt
 ├── CMakeModules
 ├── CMEPS-interface
 │    └── CMEPS
 │         └── (cesm)                         -------- CMEPS CESM
 ├── doc                                      -------- User Guide files
 ├── driver
 ├── FV3                                      -------- UFSAtm atmosphere model
 │   ├── (atmos_cubed_sphere)                 -------- FV3 dynamical 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
 │   ├── (io)                                 -------- UFSAtm write grid comp code
 │   └── (stochastic_physics)                 -------- Wrapper for stochastic physics
 ├── GOCART
 │    └── (ESMF)                              -------- GOCART model
 ├── HYCOM-interface
 │    └── HYCOM                               -------- HYCOM ocean model
 │        └── (NUOPC)                         -------- NUOPC HYCOM cap
 ├── LICENSE.md
 ├── modulefiles                              -------- system module files for supported␣
```

(continues on next page)

(continued from previous page)

```
↪HPC systems
├── MOM6-interface
│     └── MOM6
│           ├── (src)                              -------- MOM6 ocean model
│           └── (config_source/drivers/nuopc_cap)  -------- NUOPC MOM6 cap
├── NOAHMP-interface
│     └── noahmp
│           ├── (cmake)                            -------- Noah-MP land model
│           ├── (drivers/nuopc)                    -------- NUOPC Noah-MP cap
│           ├── (parameters)
│           └── (src)
├── README.md
├── stochastic_physics                             -------- stochastic physics pattern␣
↪generator
├── tests                                          -------- regression test infrastructure
│    └── parm
│    └── tests
│    └── fv3_conf
└── WW3
      └── (model)                                  -------- WW3 model
            └── (src)                              -------- NUOPC WW3 caps
```

CHAPTER

THREE

BUILDING AND RUNNING THE UFS WEATHER MODEL

3.1 Supported Platforms & Compilers

Before running the Weather Model (*WM*), users should determine which of the *levels of support* is applicable to their system. Generally, Level 1 & 2 systems are restricted to those with access through NOAA and its affiliates. These systems are named (e.g., Hera, Orion, Derecho). Level 3 & 4 systems include certain personal computers or non-NOAA-affiliated HPC systems. The prerequisite software libraries for building the WM already exist in a centralized location on Level 1/preconfigured systems, so users may skip directly to *getting the data* and downloading the code. On other systems, users will need to build the prerequisite libraries using *spack-stack* or *HPC-Stack*.

3.2 Prerequisite Libraries

The UFS WM requires a number of libraries. The WM uses two categories of libraries, which are available as a bundle via *spack-stack* or *HPC-Stack*:

1. *NCEP* 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.
2. Third-party libraries (*NCEPLIBS-external*): These are libraries that were developed externally to the UFS Weather Model. They are general software packages that are also used by other community models. Building these libraries is optional if users can point to existing builds of these libraries on their system instead.

Note: Currently, spack-stack is the software stack validated by the UFS WM for running *regression tests*. Spack-stack is a Spack-based method for installing UFS prerequisite software libraries. UFS applications and components are also shifting to spack-stack from HPC-Stack but are at various stages of this transition. Although users can still build and use HPC-Stack, the UFS WM no longer uses HPC-Stack for validation, and support for this option is being deprecated.

3.2.1 Common Modules

As of May 19, 2023, the UFS WM Regression Tests (*RTs*) on Level 1 systems use the following common modules:

```
bacio/2.4.1
crtm/2.4.0
esmf/8.3.0b09
fms/2022.04
g2/3.4.5
```

(continues on next page)

(continued from previous page)

```
g2tmpl/1.10.2
gftl-shared/v1.5.0
hdf5/1.10.6
ip/3.3.3
jasper/2.0.25
libpng/1.6.37
mapl/2.22.0-esmf-8.3.0b09
netcdf/4.7.4
pio/2.5.7
sp/2.3.3
w3emc/2.9.2
zlib/1.2.11
```

The most updated list of common modules can be viewed in `ufs_common.lua` here.

Attention: Documentation is available for installing spack-stack and HPC-Stack, respectively. One of these software stacks (or the libraries they contain) must be installed before running the UFS Weather Model.

3.3 Get Data

The WM RTs require input files to run. These include static datasets, files that depend on grid resolution and initial/boundary conditions, and model configuration files. On Level 1 and 2 systems, the data required to run the WM RTs are already available in the following locations:

Table 3.1: Data Locations for Level 1 & 2 Systems

Machine	File location
Derecho	/glade/derecho/scratch/epicufsrt/ufs-weather-model/RT
Gaea	/lustre/f2/pdata/ncep_shared/emc.nemspara/RT
Hera	/scratch1/NCEPDEV/nems/emc.nemspara/RT
Jet	/mnt/lfs4/HFIP/hfv3gfs/role.epic/RT
Orion	/work/noaa/nems/emc.nemspara/RT
S4	/data/prod/emc.nemspara/RT
WCOSS2	/lfs/h2/emc/nems/noscrub/emc.nems/RT

For Level 3-4 systems, the data must be added to the user's system. Publicly available RT data is available in the UFS WM Data Bucket. Data for running RTs off of the develop branch is available for the most recent 60 days. To view the data, users can visit https://noaa-ufs-regtests-pds.s3.amazonaws.com/index.html. To download data, users must select the data they want from the bucket and either download it in their browser or via a `wget` command. For example, to get the data for `control_p8` (specifically the May 17, 2023 `develop` branch version of the WM), run:

```
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪atmf000.nc
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪atmf021.nc
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪atmf024.nc
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
```

(continues on next page)

(continued from previous page)

```
↪GFSFLX.GrbF00
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪GFSFLX.GrbF21
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪GFSFLX.GrbF24
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪GFSPRS.GrbF00
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪GFSPRS.GrbF21
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪GFSPRS.GrbF24
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪sfcf000.nc
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪sfcf021.nc
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪sfcf024.nc
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪RESTART/20210323.060000.coupler.res
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪RESTART/20210323.060000.fv_core.res.nc
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪RESTART/20210323.060000.fv_core.res.tile1.nc
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪RESTART/20210323.060000.fv_core.res.tile2.nc
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪RESTART/20210323.060000.fv_core.res.tile3.nc
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪RESTART/20210323.060000.fv_core.res.tile4.nc
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪RESTART/20210323.060000.fv_core.res.tile5.nc
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪RESTART/20210323.060000.fv_core.res.tile6.nc
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪RESTART/20210323.060000.fv_srf_wnd.res.tile1.nc
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪RESTART/20210323.060000.fv_srf_wnd.res.tile2.nc
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪RESTART/20210323.060000.fv_srf_wnd.res.tile3.nc
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪RESTART/20210323.060000.fv_srf_wnd.res.tile4.nc
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪RESTART/20210323.060000.fv_srf_wnd.res.tile5.nc
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪RESTART/20210323.060000.fv_srf_wnd.res.tile6.nc
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪RESTART/20210323.060000.fv_tracer.res.tile1.nc
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪RESTART/20210323.060000.fv_tracer.res.tile2.nc
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪RESTART/20210323.060000.fv_tracer.res.tile3.nc
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
```

(continues on next page)

(continued from previous page)

```
↪RESTART/20210323.060000.fv_tracer.res.tile4.nc
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪RESTART/20210323.060000.fv_tracer.res.tile5.nc
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪RESTART/20210323.060000.fv_tracer.res.tile6.nc
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪RESTART/20210323.060000.phy_data.tile1.nc
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪RESTART/20210323.060000.phy_data.tile2.nc
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪RESTART/20210323.060000.phy_data.tile3.nc
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪RESTART/20210323.060000.phy_data.tile4.nc
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪RESTART/20210323.060000.phy_data.tile5.nc
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪RESTART/20210323.060000.phy_data.tile6.nc
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪RESTART/20210323.060000.sfc_data.tile1.nc
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪RESTART/20210323.060000.sfc_data.tile2.nc
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪RESTART/20210323.060000.sfc_data.tile3.nc
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪RESTART/20210323.060000.sfc_data.tile4.nc
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪RESTART/20210323.060000.sfc_data.tile5.nc
wget https://noaa-ufs-regtests-pds.s3.amazonaws.com/develop-20230517/INTEL/control_p8/
↪RESTART/20210323.060000.sfc_data.tile6.nc
```

Detailed information on input files can be found in Chapter 4.

3.4 Downloading the Weather Model Code

To clone the develop branch of the `ufs-weather-model` repository and update its submodules, execute the following commands:

```
git clone --recursive https://github.com/ufs-community/ufs-weather-model.git ufs-weather-
↪model
cd ufs-weather-model
```

Compiling the model will take place within the `ufs-weather-model` directory created by this command.

3.5 Building the Weather Model

3.5.1 Loading the Required Modules

The process for loading modules is fairly straightforward on NOAA *Level 1 Systems*. Users may need to make adjustments when running on other systems.

On NOAA Level 1 & 2 Systems

Modulefiles for *preconfigured platforms* are located in `modulefiles/ufs_<platform>.<compiler>`. For example, to load the modules from the `ufs-weather-model` directory on Hera:

```
module use modulefiles
module load ufs_hera.intel
```

Note that loading this module file will also set the CMake environment variables shown in Table 3.2.

Table 3.2: *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

On Other Systems

If you are not running on one of the pre-configured platforms, you will need to set the environment variables manually. For example, in a bash shell, a command in the following form will set the C compiler environment variable:

```
export CMAKE_C_COMPILER=</path/to/C/compiler>
```

3.5.2 Setting the `CMAKE_FLAGS` and `CCPP_SUITES` Environment Variables

The UFS Weather Model can be built in one of several configurations (see Table 4.1 for common options). The `CMAKE_FLAGS` environment variable specifies which configuration to build using the `-DAPP` and `-DCCPP_SUITES` variables. Users set which components to build using `-DAPP`. Users select the *CCPP* suite(s) by setting the `CCPP_SUITES` environment variable at build time in order to have one or more CCPP physics suites available at runtime. Multiple suites can be set. Additional variables, such as `-D32BIT=ON`, can be set if the user chooses. These options are documented in Section 6.1.3. The following examples assume a bash shell.

ATM Configurations

Standalone ATM

For the `ufs-weather-model ATM` configuration (standalone *ATM*):

```
export CMAKE_FLAGS="-DAPP=ATM -DCCPP_SUITES=FV3_GFS_v16"
```

ATMW

For the `ufs-weather-model ATMW` configuration (standalone ATM coupled to *WW3*):

```
export CMAKE_FLAGS="-DAPP=ATMW -DCCPP_SUITES=FV3_GFS_v16"
```

ATMAERO

For the `ufs-weather-model ATMAERO` configuration (standalone ATM coupled to *GOCART*):

```
export CMAKE_FLAGS="-DAPP=ATMAERO -DCCPP_SUITES=FV3_GFS_v17_p8"
```

ATMAQ

For the `ufs-weather-model ATMAQ` configuration (standalone ATM coupled to *CMAQ*):

```
export CMAKE_FLAGS="-DAPP=ATMAQ -DCCPP_SUITES=FV3_GFS_v15p2"
```

ATML

For the `ufs-weather-model ATML` configuration (standalone ATM coupled to *LND*):

```
export CMAKE_FLAGS="-DAPP=ATML -DCCPP_SUITES=FV3_GFS_v17_p8"
```

S2S Configurations

S2S

For the `ufs-weather-model S2S` configuration (coupled atm/ice/ocean):

```
export CMAKE_FLAGS="-DAPP=S2S -DCCPP_SUITES=FV3_GFS_v17_coupled_p8"
```

To turn on debugging flags, add `-DDEBUG=ON` flag after `-DAPP=S2S`. Users can allow verbose build messages by running:

```
export BUILD_VERBOSE=1
```

To receive atmosphere-ocean fluxes from the CMEPS *mediator*, add the argument `-DCMEPS_AOFLUX=ON`. For example:

```
export CMAKE_FLAGS="-DAPP=S2S -DCCPP_SUITES=FV3_GFS_v17_coupled_p8_sfcocn -DCMEPS_
↪AOFLUX=ON"
```

S2SA

For the `ufs-weather-model S2SA` configuration (atm/ice/ocean/aerosols):

```
export CMAKE_FLAGS="-DAPP=S2SA -DCCPP_SUITES=FV3_GFS_2017_coupled,FV3_GFS_v15p2_coupled,
↪FV3_GFS_v16_coupled,FV3_GFS_v16_coupled_noahmp"
```

S2SW

For the `ufs-weather-model` `S2SW` configuration (atm/ice/ocean/wave):

```
export CMAKE_FLAGS="-DAPP=S2SW -DCCPP_SUITES=FV3_GFS_v17_coupled_p8"
```

S2SWA

For the `ufs-weather-model` `S2SWA` configuration (atm/ice/ocean/wave/aerosols):

```
export CMAKE_FLAGS="-DAPP=S2SWA -DCCPP_SUITES=FV3_GFS_v17_coupled_p8,FV3_GFS_cpld_
↪rasmgshocnsstnoahmp_ugwp"
```

NG-GODAS Configuration

For the `ufs-weather-model` `NG-GODAS` configuration (atm/ocean/ice/data assimilation):

```
export CMAKE_FLAGS="-DAPP=NG-GODAS"
```

HAFS Configurations

HAFS

For the `ufs-weather-model` `HAFS` configuration (atm/ocean) in 32 bit:

```
export CMAKE_FLAGS="-DAPP=HAFS -D32BIT=ON -DCCPP_SUITES=FV3_HAFS_v0_gfdlmp_tedmf_nonsst,
↪FV3_HAFS_v0_gfdlmp_tedmf"
```

HAFSW

For the `ufs-weather-model` `HAFSW` configuration (atm/ocean/wave) in 32-bit with moving nest:

```
export CMAKE_FLAGS="-DAPP=HAFSW -D32BIT=ON -DMOVING_NEST=ON -DCCPP_SUITES=FV3_HAFS_v0_
↪gfdlmp_tedmf,FV3_HAFS_v0_gfdlmp_tedmf_nonsst,FV3_HAFS_v0_thompson_tedmf_gfdlsf"
```

HAFS-ALL

For the `ufs-weather-model` `HAFS-ALL` configuration (data/atm/ocean/wave) in 32 bit:

```
export CMAKE_FLAGS="-DAPP=HAFS-ALL -D32BIT=ON -DCCPP_SUITES=FV3_HAFS_v0_gfdlmp_tedmf,FV3_
↪HAFS_v0_gfdlmp_tedmf_nonsst"
```

LND Configuration

LND

For the `ufs-weather-model` `LND` configuration (datm/land):

```
export CMAKE_FLAGS="-DAPP=LND"
```

3.5.3 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, users should see the executable, named `ufs_model`, in the `ufs-weather-model/build/` directory. If users prefer to build in a different directory, specify the `BUILD_DIR` environment variable. For example: `export BUILD_DIR=test_cpld` will build in the `ufs-weather-model/test_cpld` directory instead.

Expert help is available through GitHub Discussions. Users may post questions there for help with difficulties related to the UFS WM.

3.6 Running the Model

Attention: Although the following discussions are general, users may not be able to execute the script successfully "as is" unless they are on a Tier-1 platform.

3.6.1 Using the Regression Test Script

Users can run a number of preconfigured regression test cases from the `rt.conf` file using the regression test script `rt.sh` in the `tests` directory. `rt.sh` is the top-level script that calls lower-level scripts to build specified WM configurations, set up environments, and run tests. Users must edit the `rt.conf` file to indicate which tests/configurations to run.

The `rt.conf` File

Each line in the PSV (Pipe-separated values) file, `rt.conf`, contains four columns of information. The first column specifies whether to build a test (`COMPILE`) or run a test (`RUN`). The second column specifies either configuration information for building a test or the name of a test to run. Thus, the second column in a `COMPILE` line will list the application to build (e.g., `-DAPP=S2S`), the CCPP suite to use (e.g., `-DCCPP_SUITES=FV3_GFS_2017_coupled`), and additional build options (e.g., `-DDEBUG=ON`) as needed. On a `RUN` line, the second column will contain a test name (e.g., `control_p8`). The test name should match the name of one of the test files in the `tests/tests` directory or, if the user is adding a new test, the name of the new test file. The third column of `rt.conf` relates to the platform; if blank, the test can run on any WM Tier-1 platform. The fourth column deals with baseline creation (see information on `-c` option *below* for more), and `fv3` means that the test will be included during baseline creation.

The order of lines in `rt.conf` matters since `rt.sh` processes them sequentially; a `RUN` line should be preceeded by a `COMPILE` line that builds the model used in the test. The following `rt.conf` file excerpt builds the standalone ATM model with GFS_v16 physics in 32-bit mode and then runs the `control` test:

```
COMPILE | -DAPP=ATM -DCCPP_SUITES=FV3_GFS_v16 -D32BIT=ON | | fv3
RUN     | control                                       | | fv3
```

The `rt.conf` file includes a large number of tests. If the user wants to run only specific tests, s/he can either (1) comment out the tests to be skipped (using the # prefix) or (2) create a new file (e.g., `my_rt.conf`), add the tests, and execute `./rt.sh -l my_rt.conf`.

On NOAA RDHPCS

On Tier-1 platforms, users can run regression tests by editing the `rt.conf` file and executing:

```
./rt.sh -l rt.conf
```

Users may need to add additional command line arguments or change information in the `rt.sh` file as well. This information is provided in Section 3.6.1 below.

On Other Systems

Users on non-NOAA systems will need to make adjustments to several files in the `tests` directory before running `rt.sh`, including:

- `rt.sh`
- `run_test.sh`
- `detect_machine.sh`
- `default_vars.sh`
- `fv3_conf/fv3_slurm.IN_*`
- `fv3_conf/compile_slurm.IN_*`
- `compile.sh`
- `module-setup.sh`

The `rt.sh` File

This section contains additional information on command line options and troubleshooting for the `rt.sh` file.

Optional Arguments

To display detailed information on how to use `rt.sh`, users can simply run `./rt.sh`, which will output the following options:

```
./rt.sh -c | -e | -h | -k | -w | -d | -l <file> | -m | -n <name> | -r
   -c  create new baseline results
   -e  use ecFlow workflow manager
   -h  display this help
   -k  keep run directory after rt.sh is completed
   -l  runs test specified in <file>
   -m  compare against new baseline results
   -n  run single test <name>
```

(continues on next page)

(continued from previous page)

```
   -r  use Rocoto workflow manager
   -w  for weekly_test, skip comparing baseline results
   -d  delete run direcotries that are not used by other tests
```

When running a large number (10's or 100's) of tests, the `-e` or `-r` options can significantly decrease testing time by using a workflow manager (ecFlow or Rocoto, respectively) to queue the jobs according to dependencies and run them concurrently. The `-n` option can be used to run a single test; for example, `./rt.sh -n control` will build the ATM model and run the `control` test. The `-c` option is used to create a baseline. New baselines are needed when code changes lead to result changes and therefore deviate from existing baselines on a bit-for-bit basis.

To run `rt.sh` using a custom configuration file and the Rocoto workflow manager, create the configuration file (e.g. `my_rt.conf`) based on the desired tests in `rt.conf`, and run:

```
./rt.sh -r -l my_rt.conf
```

adding additional arguments as desired.

To run a single test, users can try the following command instead of creating a `my_rt.conf` file:

```
./rt.sh -r -k -n control_p8
```

Troubleshooting

Users may need to adjust certain information in the `rt.sh` file, such as the *Machine* and *Account* variables (`$MACHINE_ID` and `$ACCNR`), for the tests to run correctly. If there is a problem with these or other variables (e.g., file paths), the output should indicate where:

```
+ echo 'Machine: ' hera.intel '    Account: ' nems
Machine:  hera.intel     Account:  nems
+ mkdir -p /scratch1/NCEPDEV/stmp4/First.Last
mkdir: cannot create directory ‘/scratch1/NCEPDEV/stmp4/First.Last’: Permission denied
++ echo 'rt.sh error on line 370'
rt.sh error on line 370
```

Then, users can adjust the information in `rt.sh` accordingly.

Log Files

The regression test generates a number of log files. The summary log file `RegressionTests_<machine>.<compiler>.log` in the `tests` directory compares the results of the test against the baseline for a given platform and reports the outcome:

- `'Missing file'` results when the expected files from the simulation are not found and typically occurs when the simulation did not run to completion;
- `'OK'` means that the simulation results are bit-for-bit identical to those of the baseline;
- `'NOT OK'` when the results are **not** bit-for-bit identical; and
- `'Missing baseline'` when there is no baseline data to compare against.

More detailed log files are located in the `tests/log_<machine>.<compiler>/` directory. The run directory path, which corresponds to the value of `RUNDIR` in the `run_<test-name>` file, is particularly useful. `$RUNDIR` is a self-contained (i.e., sandboxed) directory with the executable file, initial conditions, model configuration files, environment setup scripts and a batch job submission script. The user can run the test by navigating into $RUNDIR and invoking the command:

```
sbatch job_card
```

This can be particularly useful for debugging and testing code changes. Note that $RUNDIR is automatically deleted at the end of a successful regression test; specifying the -k option retains the $RUNDIR, e.g. ./rt.sh -l rt.conf -k.

Inside the $RUNDIR directory are a number of model configuration files (input.nml, model_configure, nems.configure) and other application dependent files (e.g., ice_in for the Subseasonal-to-Seasonal Application). These model configuration files are generated by rt.sh from the template files in the tests/parm directory. Specific values used to fill in the template files are test-dependent and are set in two stages. First, default values are specified in tests/default_vars.sh, and the default values are overriden if necessary by values specified in a test file tests/tests/<test-name>. For example, the variable DT_ATMOS is initially assigned 1800 in the function export_fv3 of the script default_vars.sh, but the test file tests/tests/control overrides this setting by reassigning 720 to the variable.

The files fv3_run and job_card also reside in the $RUNDIR directory. These files are generated from the template files in the tests/fv3_conf directory. job_card is a platform-specific batch job submission script, while fv3_run prepares the initial conditions for the test by copying relevant data from the input data directory of a given platform to the $RUNDIR directory. Table 3.3 summarizes the subdirectories discussed above.

Table 3.3: *Regression Test Subdirectories*

Name	Description
tests/	Regression test root directory. Contains rt-related scripts and the summary log file
tests/tests/	Contains specific test files
tests/parm/	Contains templates for model configuration files
tests/fv3_conf/	Contains templates for setting up initial conditions and a batch job
tests/log_*/	Contains fine-grained log files

Creating a New Test

When a developer needs to create a new test for his/her implementation, the first step would be to identify a test in the tests/tests directory that can be used as a basis and to examine the variables defined in the test file. As mentioned above, some of the variables may be overrides for those defined in default_vars.sh. Others may be new variables that are needed specifically for that test. Default variables and their values are defined in the export_fv3 function of the default_vars.sh script for ATM configurations, the export_cpl function for S2S configurations, and the export_datm function for the NG-GODAS configuration. Also, the names of template files for model configuration and initial conditions can be identified via variables INPUT_NML, NEMS_CONFIGURE and FV3_RUN by running grep -n INPUT_NML * inside the tests and tests/tests directories.

3.6.2 Using the Operational Requirement Test Script

The operational requirement test script opnReqTest in the tests directory can be used to run tests in place of rt.sh. Given the name of a test, opnReqTest carries out a suite of test cases. Each test case addresses an aspect of the requirements that new operational implementations must satisfy. These requirements are shown in Table 3.4. For the following discussions on opnReqTest, the user should note the distinction between 'test name' and 'test case'. Examples of test names are control, cpld_control and regional_control which are all found in the tests/tests directory, whereas test case refers to any one of the operational requirements: thr, mpi, dcp, rst, bit and dbg.

Table 3.4: *Operational Requirements*

Case	Description
thr	Varying the number of threads produces the same results
mpi	Varying the number of MPI tasks produces the same results
dcp	Varying the decomposition (i.e. tile layout of FV3) produces the same results
rst	Restarting produces the same results
bit	Model can be compiled in double/single precision and run to completion
dbg	Model can be compiled and run to completion in debug mode

The operational requirement testing uses the same testing framework as the regression tests, so it is recommened that the user first read Section 3.6.1. All the files in the subdirectories shown in Table 3.3 are relevant to the operational requirement test. The only difference is that the `opnReqTest` script replaces `rt.sh`. The `tests/opnReqTests` directory contains opnReqTest-specific lower-level scripts used to set up run configurations.

On Tier-1 platforms, tests can be run by invoking

```
./opnReqTest -n <test-name>
```

For example, `./opnReqTest -n control` performs all six test cases listed in Table 3.4 for the `control` test. At the end of the run, a log file `OpnReqTests_<machine>.<compiler>.log` is generated in the `tests` directory, which informs the user whether each test case passed or failed. The user can choose to run a specific test case by invoking

```
./opnReqTest -n <test-name> -c <test-case>
```

where `<test-case>` is one or more comma-separated values selected from `thr`, `mpi`, `dcp`, `rst`, `bit`, `dbg`. For example, `./opnReqTest -n control -c thr,rst` runs the `control` test and checks the reproducibility of threading and restart.

The user can see different command line options available to `opnReqTest` by executing `./opnReqTest -h`, which produces the following results:

```
Usage: opnReqTest -n <test-name> [ -c <test-case> ] [-b] [-d] [-e] [-k] [-h] [-x] [-z]

   -n  specify <test-name>

   -c  specify <test-case>
         defaults to all test-cases: thr,mpi,dcp,rst,bit,dbg,fhz
         comma-separated list of any combination of std,thr,mpi,dcp,rst,bit,dbg,fhz

   -b  test reproducibility for bit; compare against baseline
   -d  test reproducibility for dbg; compare against baseline
   -s  test reproducibility for std; compare against baseline
   -e  use ecFlow workflow manager
   -k  keep run directory
   -h  display this help and exit
   -x  skip compile
   -z  skip run
```

Frequently used options are `-e` to use the ecFlow workflow manager, and `-k` to keep the `$RUNDIR`. Not that the Rocoto workflow manager is not used operationally and therefore is not an option.

As discussed in Section 3.6.1, the variables and values used to configure model parameters and to set up initial conditions in the `$RUNDIR` directory are set up in two stages. First, `tests/default_vars.sh` define default values; then a specific test file in the `tests/tests` subdirectory either overrides the default values or creates new variables if required by the test. The regression test treats the different test cases shown in Table 3.4 as different tests. Therefore, each test case requires a test file in the `tests/tests` subdirectory. Examples include `control_2threads`, `control_decomp`, `control_restart` and `control_debug`, which are just variations of the `control` test to check various reproducibilities. There are two potential issues with this approach. First, if several different variations of a given test were created and included in the `rt.conf` file, there would be too many tests to run. Second, if a new test is added by the user, s/he will also have to create these variations. The idea behind the operational requirement test is to automatically configure and run these variations, or test cases, given a test file. For example, `./opnReqTest -n control` will run all six test cases in Table 3.4 based on a single `control` test file. Similarly, if the user adds a new test `new_test`, then `./opnReqTest -n new_test` will run all test cases. This is done by the operational requirement test script `opnReqTest` by adding a third stage of variable overrides. The related scripts can be found in the `tests/opnReqTests` directory.

CHAPTER

FOUR

DATA: INPUT, MODEL CONFIGURATION, AND OUTPUT FILES

The UFS Weather Model can be run in one of several configurations (sometimes referred to as “applications”), from a single-component atmospheric model to a fully coupled model with multiple earth system components (e.g., atmosphere, ocean, sea-ice and mediator). Currently, supported configurations include:

Table 4.1: *Supported ufs-weather-model applications*

Configuration Name	Description
ATM	Standalone Atmospheric Model (*ATM*)
ATMW	*ATM* coupled to *WW3*
ATMAERO	*ATM* coupled to *GOCART*
ATMAQ	*ATM* coupled to *CMAQ*
ATML	*ATM* coupled to *LND*
S2S	Coupled *ATM* - *MOM6* - *CICE6* - *CMEPS*
S2SA	Coupled *ATM* - *MOM6* - *CICE6* - *GOCART* - *CMEPS*
S2SW	Coupled *ATM* - *MOM6* - *CICE6* - *WW3* - *CMEPS*
S2SWA	Coupled *ATM* - *MOM6* - *CICE6* - *GOCART* - *WW3* - *CMEPS*
NG-GODAS	Coupled *CDEPS* - *DATM* - *MOM6* - *CICE6* - *CMEPS*
LND	Coupled *CDEPS* - *DATM* - *LND* -*CMEPS*
HAFS	Coupled *ATM* - *HYCOM* - *CMEPS*
HAFSW	Coupled *ATM* - *HYCOM* - *WW3* - *CMEPS*
HAFS-ALL	Coupled *CDEPS* - *ATM* - *HYCOM* - *WW3* - *CMEPS*

This chapter describes the input and output files needed for executing the model in the various supported configurations (see Table 4.1). Each of the component models for a given configuration requires specific input files, and each component model outputs a particular set of files. Each configuration requires a set of model configuration files, as well. This chapter describes the input and output files involved with each component model. It also discusses the various configuration files involved in running the model. Users will need to view the input file requirements for each component model involved in the configuration they are running. For example, users running the *S2S* configuration would need to gather input data required for the *ATM*, *MOM6*, and *CICE6* component models. Then, they would need to alter certain model configuration files to reflect the `ufs-weather-model` configuration that they plan to run.

4.1 Input files

There are three types of files needed to execute a run:

1. Static datasets (*fix* files containing climatological information)
2. Files that depend on grid resolution and initial/boundary conditions
3. Model configuration files (such as namelists)

Information on the first two types of file appears in detail below for each component model. Information on Model Configuration files can be viewed in Section 4.2.

4.1.1 ATM

Static Datasets (i.e., *fix files*)

The static input files for global configurations are listed and described in Table 4.2. Similar files are used for a regional grid but are grid-specific and generated by pre-processing utilities (e.g., UFS_UTILS).

Table 4.2: *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

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 4.3. 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 4.4. Note that the regional grid is referred to as Tile 7 here, and it is generated by several pre-processing utilities.

Table 4.3: *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 4.4: *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	✓

4.1.2 MOM6

Static Datasets (i.e., *fix files*)

The static input files for global configurations are listed and described in Table 4.5.

Table 4.5: *Fix files containing climatological information*

Filename	Description	Used in resolution
runoff.daitren.clim.1440x1080.v20180328.	climatological runoff	0.25
runoff.daitren.clim.720x576.v20180328.nc	climatological runoff	0.50
seawifs-clim-1997-2010.1440x1080.v20180328.nc	climatological chlorophyll concentration in sea water	0.25
seawifs-clim-1997-2010.720x576.v20180328.nc	climatological chlorophyll concentration in sea water	0.50
seawifs_1998-2006_smoothed_2X.nc	climatological chlorophyll concentration in sea water	1.00
tidal_amplitude.v20140616.nc	climatological tide amplitude	0.25
tidal_amplitude.nc	climatological tide amplitude	0.50, 1.00
geothermal_davies2013_v1.nc	climatological geothermal heat flow	0.50, 0.25
KH_background_2d.nc	climatological 2-d background harmonic viscosities	1.00

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 4.6.

Table 4.6: *Input files containing grid information and initial conditions for global configurations*

Filename	Description	Valid RES options	Date-dependent
ocean_hgrid.nc	horizonal grid information	1.00, 0.50, 0.25	
ocean_mosaic.nc	specify horizonal starting and ending points index	1.00, 0.50, 0.25	
ocean_topog.nc	ocean topography	1.00, 0.50, 0.25	
ocean_mask.nc	lans/sea mask	1.00, 0.50, 0.25	
hycom1_75_800m.nc	vertical coordinate level thickness	1.00, 0.50, 0.25	
layer_coord.nc	vertical layer target potential density	1.00, 0.50, 0.25	
All_edits.nc	specify grid points where topography are manually modified to adjust throughflow strength for narrow channels	0.25	
topo_edits_011818.	specify grid points where topography are manually modified to adjust throughflow strength for narrow channels	1.00	
MOM_channels_gl	specifies restricted channel widths	0.50, 0.25	
MOM_channel_SPI	specifies restricted channel widths	1.00	
interpolate_zgrid_40L.nc	specify target depth for output	1.00, 0.50, 0.25	
MOM.res*nc	ocean initial conditions (from CPC ocean DA)	0.25	✓
MOM6_IC_TS.nc	ocean temperature and salinity initial conditions (from CFSR)	1.00, 0.50, 0.25	✓

4.1.3 HYCOM

Static Datasets (i.e., *fix files*)

Static input files have been created for several regional domains. These domains are listed and described in Table 4.7.

Table 4.7: *The following table describes each domain identifier.*

Identifier	Description
hat10	Hurricane North Atlantic (1/12 degree)
hep20	Hurricane Eastern North Pacific (1/12 degree)
hwp30	Hurricane Western North Pacific (1/12 degree)
hcp70	Hurricane Central North Pacific (1/12 degree)

Static input files are listed and described in Table 4.8. Several datasets contain both dot-a (.a) and dot-b (.b) files. Dot-a files contain data written as 32-bit IEEE real values (idm*jdm) and dot-b files contain plain text metadata for each field in the dot-a file.

Table 4.8: *Fix files containing climatological information*

Filename	Description	Domain
blkdat.input	Model input parameters	
patch.input	Tile description	
ports.input	Open boundary cells	
forcing.chl.(a,b)	Chlorophyll (monthly climatology)	hat10, hep20, hwp30, hcp70
forcing.rivers.(a,b)	River discharge (monthly climatology)	hat10, hep20, hwp30, hcp70
iso.sigma.(a,b)	Fixed sigma thickness	hat10, hep20, hwp30, hcp70
regional.depth.(a,b)	Total depth of ocean	hat10, hep20, hwp30, hcp70
regional.grid.(a,b)	Grid information for HYCOM "C" grid	hat10, hep20, hwp30, hcp70
relax.rmu.(a,b)	Open boundary nudging value	hat10, hep20, hwp30, hcp70
relax.ssh.(a,b)	Surface height nudging value (monthly climatology)	hat10, hep20, hwp30, hcp70
tbaric.(a,b)	Thermobaricity correction	hat10, hep20, hwp30, hcp70
thkdf4.(a,b)	Diffusion velocity (m/s) for Laplacian thickness diffusivity	hat10, hep20, hwp30, hcp70
veldf2.(a,b)	Diffusion velocity (m/s) for biharmonic momentum dissipation	hat10, hep20, hwp30, hcp70
veldf4.(a,b)	Diffusion velocity (m/s) for Laplacian momentum dissipation	hat10, hep20, hwp30, hcp70

Grid Description and Initial Condition Files

The input files containing time dependent configuration and forcing data are listed and described in Table 4.9. These files are generated for specific regional domains (see Table 4.7) during ocean prep. When uncoupled, the the forcing data drives the ocean model. When coupled, the forcing data is used to fill in unmapped grid cells. Several datasets contain both dot-a (.a) and dot-b (.b) files. Dot-a files contain data written as 32-bit IEEE real values (idm*jdm) and dot-b files contain plain text metadata for each field in the dot-a file.

Table 4.9: *Input files containing grid information, initial conditions, and forcing data for regional configurations.*

Filename	Description	Domain	Date-dependent
limits	Model begin and end time (since HYCOM epoch)		✓
forcing.airtmp.(a,b)	GFS forcing data for 2m air temperature	hat10, hep20, hwp30, hcp70	✓
forcing.mslprs.(a,b)	GFS forcing data for mean sea level pressure (symlink)	hat10, hep20, hwp30, hcp70	✓
forcing.precip.(a,b)	GFS forcing data for precipitation rate	hat10, hep20, hwp30, hcp70	✓
forcing.presur.(a,b)	GFS forcing data for mean sea level pressure	hat10, hep20, hwp30, hcp70	✓
forcing.radflx.(a,b)	GFS forcing data for total radiation flux	hat10, hep20, hwp30, hcp70	✓
forcing.shwflx.(a,b)	GFS forcing data for net downward shortwave radiation flux	hat10, hep20, hwp30, hcp70	✓
forcing.surtmp.(a,b)	GFS forcing data for surface temperature	hat10, hep20, hwp30, hcp70	✓
forcing.tauewd.(a,b)	GFS forcing data for eastward momentum flux	hat10, hep20, hwp30, hcp70	✓
forcing.taunwd.(a,b)	GFS forcing data for northward momentum flux	hat10, hep20, hwp30, hcp70	✓
forcing.vapmix.(a,b)	GFS forcing data for 2m vapor mixing ratio	hat10, hep20, hwp30, hcp70	✓
forcing.wndspd.(a,b)	GFS forcing data for 10m wind speed	hat10, hep20, hwp30, hcp70	✓
restart_in.(a,b)	Restart file for ocean state variables	hat10, hep20, hwp30, hcp70	✓

4.1.4 CICE6

Static Datasets (i.e., *fix files*)

No fix files are required for CICE6.

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 4.10.

Table 4.10: *Input files containing grid information and initial conditions for global configurations*

Filename	Description	Valid RES options	Date-dependent
cice_model_RES.res_YYYYMMD	cice model IC or restart file	1.00, 0.50, 0.25	✓
grid_cice_NEMS_mxRES.nc	cice model grid at resolution RES	100, 050, 025	
kmtu_cice_NEMS_mxRES.nc	cice model land mask at resolution RES	100, 050, 025	

4.1.5 WW3

Static Datasets (i.e., *fix files*)

No fix files are required for WW3.

Grid Description and Initial Condition Files

The files for global configurations are listed and described in Table 4.11 for GFSv16 setup and Table 4.12 for single grid configurations. The model definitions for wave grid(s) including spectral and directional resolutions, time steps, numerical scheme and parallelization algorithm, the physics parameters, boundary conditions and grid definitions are stored in binary mod_def files. The aforementioned parameters are defined in ww3_grid.inp.<grd> and the ww3_grid executables generates the binary mod_def.<grd> files.

The WW3 version number in mod_def.<grd> files must be consistent with version of the code in ufs-weather-model. createmoddefs/creategridfiles.sh can be used in order to generate the mod_def.<grd> files, using ww3_grid.inp.<grd>, using the WW3 version in ufs-weather-model. In order to do it, the path to the location of the ufs-weather-model (UFSMODELDIR), the path to generated mod_def.<grd> outputs (OUTDIR), the path to input ww3_grid.inp.<grd> files (SRCDIR) and the path to the working directory for log files (WORKDIR) should be defined.

Table 4.11: *Input files containing grid information and conservative remapping for global configurations (GFSv16 Wave)*

Filename	Description	Spatial Resolution	nFreq	nDir
mod_def.aoc_9km	Antarctic Ocean PolarStereo [50N 90N]	9km	50	36
mod_def.gnh_10m	Global mid core [15S 52N]	10 min	50	36
mod_def.gsh_15m	southern ocean [79.5S 10.5S]	15 min	50	36
mod_def.glo_15mxt	Global 1/4 extended grid [90S 90S]	15 min	36	24
mod_def.points	GFSv16-wave spectral grid point output	na	na	na
rmp_src_to_dst_conserv_002_0	Conservative remapping gsh_15m to gnh_10m	na	na	na
rmp_src_to_dst_conserv_003_0	Conservative remapping aoc_9km to gnh_10m	na	na	na

Table 4.12: *Input grid information for single global/regional configurations*

Filename	Description	Spatial Resolution	nFreq	nDir
mod_def.ant_9km	Regional polar stereo antarctic grid [90S 50S]	9km	36	24
mod_def.glo_10m	Global grid [80S 80N]	10 min	36	24
mod_def.glo_30m	Global grid [80S 80N]	30 min	36	36
mod_def.glo_1deg	Global grid [85S 85N]	1 degree	25	24
mod_def.glo_2deg	Global grid [85S 85N]	2 degree	20	18
mod_def.glo_5deg	Global grid [85S 85N]	5 degree	18	12
mod_def.glo_gwes_30m	Global NAWES 30 min wave grid [80S 80N]	30 min	36	36
mod_def.natl_6m	Regional North Atlantic Basin [1.5N 45.5N; 98W 8W]	6 min	50	36

Coupled regional configurations require forcing files to fill regions that cannot be interpolated from the atmospheric component. For a list of forcing files used to fill unmapped data points see Table 4.13.

Table 4.13: *Forcing information for single regional configurations*

Filename	Description	Resolution
wind.natl_6m	Interpolated wind data from GFS	6 min

The model driver input (ww3_multi.inp) includes the input, model and output grids definition, the starting and ending times for the entire model run and output types and intervals. The ww3_multi.inp.IN template is located under tests/parm/ directory. The inputs are described hereinafter:

Table 4.14: *Model driver input*

NMGRIDS	Number of wave model grids
NFGRIDS	Number of grids defining input fields
FUNIPNT	Flag for using unified point output file.
IOSRV	Output server type
FPNTPROC	Flag for dedicated process for unified point output
FGRDPROC	Flag for grids sharing dedicated output processes

If there are input data grids defined (`NFGRIDS > 0`) then these grids are defined first (`CPLILINE`, `WINDLINE`, `ICELINE`, `CURRLINE`). These grids are defined as if they are wave model grids using the file `mod_def.<grd>`. Each grid is defined on a separate input line with `<grd>`, with nine input flags identifying $ the presence of 1) water levels 2) currents 3) winds 4) ice $ 5) momentum 6) air density and 7-9) assimilation data.

The `UNIPOINTS` defines the name of this grid for all point output, which gathers the output spectral grid in a unified point output file.

The `WW3GRIDLINE` defines actual wave model grids using 13 parameters to be read from a single line in the file for each. It includes (1) its own input grid `mod_def.<grd>`, (2-10) forcing grid ids, (3) rank number, (12) group number and (13-14) fraction of communicator (processes) used for this grid.

`RUN_BEG` and `RUN_END` define the starting and end times, `FLAGMASKCOMP` and `FLAGMASKOUT` are flags for masking at printout time (default F F), followed by the gridded and point outputs start time (`OUT_BEG`), interval (`DTFLD` and `DTPNT`) and end time (`OUT_END`). The restart outputs start time, interval and end time are define by `RST_BEG`, `DTRST`, `RST_END` respectively.

The `OUTPARS_WAV` defines gridded output fields. The `GOFILETYPE`, `POFILETYPE` and `RSTTYPE` are gridded, point and restart output types respectively.

No initial condition files are required for WW3.

Mesh Generation

For coupled applications using the CMEPS mediator, an ESMF Mesh file describing the WW3 domain is required. For regional and sub-global domains, the mesh can be created using a two-step procedure.

1. Generate a SCRIP format file for the domain
2. Generate the ESMF Mesh.

In each case, the SCRIP file needs to be checked that it contains the right start and end latitudes and longitudes to match the `mod_def` file being used.

For the HAFS regional domain, the following commands can be used:

```
ncremap -g hafswav.SCRIP.nc -G latlon=441,901#snwe=1.45,45.55,-98.05,-7.95#lat_typ=uni
↪#lat_drc=s2n
ESMF_Scrip2Unstruct hafswav.SCRIP.nc mesh.hafs.nc 0
```

For the sub-global 1-deg domain extending from latitude 85.0S:

```
ncremap -g glo_1deg.SCRIP.nc -G latlon=171,360#snwe=-85.5,85.5,-0.5,359.5#lat_typ=uni
↪#lat_drc=s2n
ESMF_Scrip2Unstruct glo_1deg.SCRIP.nc mesh.glo_1deg.nc 0
```

For the sub-global 1/2-deg domain extending from latitude 80.0S:

```
ncremap -g gwes_30m.SCRIP.nc -G latlon=321,720#snwe=-80.25,80.25,-0.25,359.75#lat_typ=uni
↪#lat_drc=s2n
ESMF_Scrip2Unstruct gwes_30m.SCRIP.nc mesh.gwes_30m.nc 0
```

For the tripole grid, the mesh file is generated as part of the `cpld_gridgen` utility in UFS_UTILS.

4.1.6 CDEPS

Static Datasets (i.e., *fix files*)

No fix files are required for CDEPS.

Grid Description and Initial Condition Files

The input files containing grid information and the time-varying forcing files for global configurations are listed and described in Table 4.15 and Table 4.16.

Data Atmosphere

Table 4.15: *Input files containing grid information and forcing files for global configurations*

Filename	Description	Date-dependent
cfsr_mesh.nc	ESMF mesh file for CFSR data source	
gefs_mesh.nc	ESMF mesh file for GEFS data source	
TL639_200618_ESMFmesh.nc	ESMF mesh file for ERA5 data source	
cfsr.YYYYMMM.nc	CFSR forcing file for year YYYY and month MM	✓
gefs.YYYYMMM.nc	GEFS forcing file for year YYYY and month MM	✓
ERA5.TL639.YYYY.MM.nc	ERA5 forcing file for year YYYY and month MM	✓

Note: Users can find atmospheric forcing files for use with the land (*LND*) component in the Land Data Assimilation (DA) data bucket. These files provide atmospheric forcing data related to precipitation, solar radiation, longwave radiation, temperature, pressure, winds, humidity, topography, and mesh data. Forcing files for the land component configuration come from the Global Soil Wetness Project Phase 3 (GSWP3) dataset.

```
clmforc.GSWP3.c2011.0.5x0.5.Prec.1999-12.nc
clmforc.GSWP3.c2011.0.5x0.5.Prec.2000-01.nc
clmforc.GSWP3.c2011.0.5x0.5.Solr.1999-12.nc
clmforc.GSWP3.c2011.0.5x0.5.Solr.2000-01.nc
clmforc.GSWP3.c2011.0.5x0.5.TPQWL.1999-12.nc
clmforc.GSWP3.c2011.0.5x0.5.TPQWL.2000-01.nc
clmforc.GSWP3.c2011.0.5x0.5.TPQWL.SCRIP.210520_ESMFmesh.nc
fv1.9x2.5_141008_ESMFmesh.nc
topodata_0.9x1.25_USGS_070110_stream_c151201.nc
topodata_0.9x1.SCRIP.210520_ESMFmesh.nc
```

See the Land DA User's Guide or the *WM LND Input* section of this page for more information on files used in land configurations of the UFS WM.

Data Ocean

Table 4.16: *Input files containing grid information and forcing files for global configurations*

Filename	Description	Date-dependent
TX025_210327_ESMFmesh_py.nc	ESMF mesh file for OISST data source	
sst.day.mean.YYYY.nc	OISST forcing file for year YYYY	✓

Table 4.17: *Input files containing grid information and forcing files for regional configurations*

Filename	Description	Date-dependent
hat10_210129_ESMFmesh_py.nc	ESMF mesh file for MOM6 data source	
GHRSST_mesh.nc	ESMF mesh file for GHRSST data source	
hycom_YYYYMM_surf_nolev.nc	MOM6 forcing file for year YYYY and month MM	✓
ghrsst_YYYYMMDD.nc	GHRSST forcing file for year YYYY, month MM and day DD	✓

4.1.7 GOCART

Static Datasets (i.e., *fix files*)

The static input files for GOCART configurations are listed and described in Table 4.18.

Table 4.18: *GOCART run control files*

Filename	Description
AERO.rc	Atmospheric Model Configuration Parameters
AERO_ExtData.rc	Model Inputs related to aerosol emissions
AERO_HISTORY.rc	Create History List for Output
AGCM.rc	Atmospheric Model Configuration Parameters
CA2G_instance_CA.bc.rc	Resource file for Black Carbon parameters
CA2G_instance_CA.br.rc	Resource file for Brown Carbon parameters
CA2G_instance_CA.oc.rc	Resource file for Organic Carbon parameters
CAP.rc	Meteorological fields imported from atmospheric model (CAP_imports) & Prognostic Tracers Table (CAP_exports)
DU2G_instance_DU.rc	Resource file for Dust parameters
GOCART2G_GridComp.rc	The basic properties of the GOCART2G Grid Components
NI2G_instance_NI.rc	Resource file for Nitrate parameters
SS2G_instance_SS.rc	Resource file for Sea Salt parameters
SU2G_instance_SU.rc	Resource file for Sulfur parameters

GOCART inputs defined in `AERO_ExtData` are listed and described in Table 4.19.

Table 4.19: *GOCART inputs defined in AERO_ExtData.rc*

Filename	Description
ExtData/dust	FENGSHA input files
ExtData/QFED	QFED biomass burning emissions
ExtData/CEDS	Anthropogenic emissions
ExtData/MERRA2	DMS concentration
ExtData/PIESA/sfc	Aviation emissions
ExtData/PIESA/L127	H2O2, OH and NO3 mixing ratios
ExtData/MEGAN_OFFLINE_BVOC	VOCs MEGAN biogenic emissions
ExtData/monochromatic	Aerosol monochromatic optics files
ExtData/optics	Aerosol radiation bands optic files for RRTMG
ExtData/volcanic	SO2 volcanic pointwise sources

The static input files when using climatology (MERRA2) are listed and described in Table 4.20.

Table 4.20: *Inputs when using climatology (MERRA2)*

Filename	Description
merra2.aerclim.2003-2014.m$(month).nc	MERRA2 aerosol climatology mixing ratio
Optics_BC.dat	BC optical look-up table for MERRA2
Optics_DU.dat	DUST optical look-up table for MERRA2
Optics_OC.dat	OC optical look-up table for MERRA2
Optics_SS.dat	Sea Salt optical look-up table for MERRA2
Optics_SU.dat	Sulfate optical look-up table for MERRA2

Grid Description and Initial Condition Files

Running GOCART in UFS does not require aerosol initial conditions, as aerosol models can always start from scratch (cold start). However, this approach does require more than two weeks of model spin-up to obtain reasonable aerosol simulation results. Therefore, the most popular method is to take previous aerosol simulation results. The result is not necessarily from the same model; it could be from a climatology result, such as MERRA2, or from a different model but with the same aerosol species and bin/size distribution.

The aerosol initial input currently read by GOCART is the same format as the UFSAtm initial input data format of `gfs_data_tile[1-6].nc` in Table 4.3, so the aerosol initial conditions should be combined with the meteorological initial conditions as one initial input file. There are many tools available for this purpose. The UFS_UTILS preprocessing utilities provide a solution for this within the Global Workflow.

4.1.8 AQM (CMAQ)

Static Datasets (i.e., *fix files*)

The static input files for AQM configurations are listed and described in Table 4.21.

Table 4.21: *AQM run control files*

Filename	Description
AQM.rc	NOAA Air Quality Model Parameters

AQM inputs defined in `aqm.rc` are listed and described in Table 4.22.

Table 4.22: *AQM inputs defined in aqm.rc*

Filename	Description
AE_cb6r3_ae6_aq.nml	AE Matrix NML
GC_cb6r3_ae6_aq.nml	GC Matrix NML
NR_cb6r3_ae6_aq.nml	NR Matrix NML
Species_Table_TR_0.nml	TR Matrix NML
CSQY_DATA_cb6r3_ae6_aq	CSQY Data
PHOT_OPTICS.dat	Optics Data
omi_cmaq_2015_361X179.dat	OMI data
NEXUS/NEXUS_Expt.nc	Emissions File
BEIS_RRFScmaq_C775.ncf	Biogenic File
gspro_biogenics_1mar2017.txt	Biogenic Speciation File
Hourly_Emissions_regrid_rrfs_13km_20190801_t	File Emissions File

4.1.9 LND

LND component datasets are available from the Land Data Assimilation (DA) System data bucket and can be retrieved using a `wget` command:

```
wget https://noaa-ufs-land-da-pds.s3.amazonaws.com/current_land_da_release_data/v1.2.0/
↪Landdav1.2.0_input_data.tar.gz
tar xvfz Landdav1.2.0_input_data.tar.gz
```

These files will be untarred into an `inputs` directory if the user does not specify a different name. They include data for Dec. 21, 2019. Table 4.23 describes the file types. In each file name, `YYYY` refers to a valid 4-digit year, `MM` refers to a valid 2-digit month, and `DD` refers to a valid 2-digit day of the month.

Table 4.23: *LND input files*

Filename(s)	Description	File Type
ufs-land_C96_init_fields.tile*.nc	Initial conditions files for each tile; the files include the initial state variables that are required for the UFS land snow DA to begin a cycling run. * stands for the grid tile number [1-6].	Initial conditions
C96.maximum_snow_albedo.tile* C96.slope_type.tile*.nc C96.soil_type.tile*.nc C96.soil_color.tile*.nc C96.substrate_temperature.tile*.n C96.vegetation_greenness.tile*.nc C96.vegetation_type.tile*.nc oro_C96.mx100.tile*.nc	Tiled static files that contain information on maximum snow albedo, slope type, soil color and type, substrate temperature, vegetation greenness and type, and orography (grid and land mask information). * stands for the grid tile number [1-6].	Static/fixed files
grid_spec.nc	Contains information on the mosaic grid	Grid
C96_grid.tile*.nc	C96 grid information for tiles 1-6, where * is the grid tile number [1-6].	Grid
C96_oro_data.tile*.nc / oro_C96.mx100.tileN.nc	Orography files that contain grid and land mask information, where * is the grid tile number [1-6]. `mx100` refers to the ocean resolution (100=1º).	Grid
See *CDEPS* for information on atmospheric forcing files.	Atmospheric forcing	CDEPS
ghcn_snwd_ioda_YYYYMMDD.	GHCN snow depth data assimilation files	DA
ufs_land_restart.YYYY-MM-DD_HH-mm-SS.nc	Restart file	Restart

Static Datasets (i.e., *fix files*)

The static files (listed in Table 4.23) include specific information on location, time, soil layers, and fixed (invariant) experiment parameters that are required for the land component to run. The data must be provided in *netCDF* format.

The following static files are available in the `inputs/UFS_WM/FV3_fix_tiled/C96/` data directory (downloaded *above*):

```
C96.maximum_snow_albedo.tile*.nc
C96.slope_type.tile*.nc
C96.soil_type.tile*.nc
C96.soil_color.tile*.nc
C96.substrate_temperature.tile*.nc
C96.vegetation_greenness.tile*.nc
C96.vegetation_type.tile*.nc
oro_C96.mx100.tile*.nc
```

where * refers to the tile number (1-6). Details on the configuration variables included in this file are available in the Land DA documentation.

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 4.23.

The initial conditions file includes the initial state variables that are required for the UFS land snow DA to begin a cycling run. The data must be provided in *netCDF* format. The initial conditions file is available in the `inputs` data directory (downloaded *above*) at the following path:

```
inputs/UFS_WM/NOAHMP_IC/ufs-land_C96_init_fields.tile*.nc
```

Grid files are available in the `inputs/UFS_WM/FV3_input_data/INPUT` directory:

```
C96_grid.tile*.nc
grid_spec.nc     # aka C96.mosaic.nc
```

The `C96_grid.tile*.nc` files contain grid information for tiles 1-6 at C96 grid resolution. The `grid_spec.nc` file contains information on the mosaic grid.

Additional Files

The LND component uses atmospheric forcing files, data assimilation files, and restart files, which are also listed in Table 4.23.

4.2 Model configuration files

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

- `diag_table`
- `field_table`
- `model_configure`
- `ufs.configure`
- `suite_[suite_name].xml` (used only at build time)
- `datm.streams` (used by CDEPS)
- `datm_in` (used by CDEPS)
- `blkdat.input` (used by HYCOM)

While the `input.nml` file is also a configuration file used by the UFS Weather Model, it is described in Section 4.2.9. 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 4.3.

4.2.1 `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 4.24.

Table 4.24: *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 4.25.

Table 4.25: *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. `"..."` denotes 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,
```

(continues on next page)

(continued from previous page)

```
 "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
###
 "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.

4.2.2 `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 adding 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 4.26. 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 4.26: *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.

4.2.3 `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 4.27 shows the following parameters that can be set in `model_configure` at run-time.

Table 4.27: *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.
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)
restart_interval	frequency to output restart file or forecast hours to write out restart file	integer	0 (0: write restart file at the end of integration; 12, -1: write out restart every 12 hours; 12 24 write out restart files at fh=12 and 24)
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)	netcdf
imo	i-dimension for output grid	integer	384
jmo	j-dimension for output grid	integer	190
output_fh	history file output forecast hours or history file output frequency if the second elelment is -1	real	-1 (negative: turn off the option, otherwise overwritten nfhout/nfhout_fh; 6 -1: output every 6 hoursr; 6 9: output history files at fh=6 and 9. Note: output_fh can only take 1032 characters)

Table 4.28 shows the following parameters in `model_configure` that are not usually changed.

Table 4.28: *Parameters that are not usually changed in model_configure at run-time.*

Parameter	Meaning	Type	Default Value
calendar	type of calendar year	character(*)	'gregorian'
fhrot	forecast hour at restart for nems/earth grid component clock in coupled model	integer	0
write_dopost	flag to do post on write grid component	logical	.true.
write_nsflip	flag to flip the latitudes from S to N to N to S on output domain	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)
iau_offset	IAU offset lengdth	integer	0

4.2.4 `ufs.configure` file

This file contains information about the various NEMS components and their run sequence. The active components for a particular model configuration are given in the *EARTH_component_list*. For each active component, the model name and compute tasks assigned to the component are given. A specific component might also require additional configuration information to be present. The `runSeq` describes the order and time intervals over which one or more component models integrate in time. Additional *attributes*, if present, provide additional configuration of the model components when coupled with the CMEPS mediator.

For the ATM application, since it consists of a single component, the `ufs.configure` is simple and does not need to be changed. A sample of the file contents is shown below:

```
# ESMF #
logKindFlag:            ESMF_LOGKIND_MULTI
globalResourceControl:  true

# EARTH #
EARTH_component_list: ATM
EARTH_attributes::
Verbosity = 0
::

# ATM #
ATM_model:                      @[atm_model]
ATM_petlist_bounds:             @[atm_petlist_bounds]
ATM_omp_num_threads:            @[atm_omp_num_threads]
ATM_attributes::
Verbosity = 0
Diagnostic = 0
::

# Run Sequence #
runSeq::
ATM
::
```

However, `ufs.configure` files for other configurations of the Weather Model are more complex. A full set of `ufs.configure` templates is available in the `ufs-weather-model/tests/parm/` directory here. Template names follow the pattern `ufs.configure.*.IN`. A number of samples are available below:

- ATMAQ configuration
- S2S (fully coupled `S2S` configuration that receives atmosphere-ocean fluxes from a mediator)
- S2SW (fully coupled `S2SW` configuration)
- S2SWA (coupled GOCART in the S2SAW configuration)
- ATM-LND (ATML configuration)
- For more HAFS, HAFSW, and HAFS-ALL configurations please see the following `ufs.configure` templates:
  - HAFS ATM-OCN
  - HAFS ATM-WAV
  - HAFS ATM-OCN-WAV
  - HAFS ATM-DOCN

Note: The `aoflux_grid` option is used to select the grid/mesh to perform atmosphere-ocean flux calculation. The possible options are `xgrid` (exchange grid), `agrid` (atmosphere model grid) and `ogrid` (ocean model grid).

Note: The `aoflux_code` option is used to define the algorithm that will be used to calculate atmosphere-ocean fluxes. The possible options are `cesm` and `ccpp`. If `ccpp` is selected then the suite file provided in the `aoflux_ccpp_suite` option is used to calculate atmosphere-ocean fluxes through the use of CCPP host model.

4.2.5 The Suite Definition File (SDF) File

There are two SDFs currently supported for the UFS Medium Range Weather App configuration:

- `suite_FV3_GFS_v15p2.xml`
- `suite_FV3_GFS_v16beta.xml`

There are four SDFs currently supported for the UFS Short Range Weather App configuration:

- `suite_FV3_GFS_v16.xml`
- `suite_FV3_RRFS_v1beta.xml`
- `suite_FV3_HRRR.xml`
- `suite_FV3_WoFS_v0.xml`

Detailed descriptions of the supported suites can be found with the CCPP v6.0.0 Scientific Documentation.

4.2.6 `datm.streams`

A data stream is a time series of input forcing files. A data stream configuration file (datm.streams) describes the information about those input forcing files.

Table 4.29: *Parameters that can be set in a data stream configuration file at run-time.*

Parameter	Meaning
taxmode01	time axis mode
mapalgo01	type of spatial mapping (default=bilinear)
tInterpAlgo01	time interpolation algorithm option
readMode01	number of forcing files to read in (current option is single)
dtimit01	ratio of max/min stream delta times (default=1.0. For monthly data, the ratio is 31/28.)
stream_offset01	shift of the time axis of a data stream in seconds (Positive offset advances the time axis forward.)
yearFirst01	the first year of the stream data
yearLast01	the last year of the stream data
yearAlign01	the simulation year corresponding to yearFirst01
stream_vectors01	the paired vector field names
stream_mesh_file01	stream mesh file name
stream_lev_dimname01	name of vertical dimension in data stream
stream_data_files01	input forcing file names
stream_data_variables01	a paired list with the name of the variable used in the file on the left and the name of the Fortran variable on the right

A sample of the data stream file is shown below:

```
stream_info:               cfsr.01
taxmode01:                 cycle
mapalgo01:                 bilinear
tInterpAlgo01:             linear
readMode01:                single
dtlimit01:                 1.0
stream_offset01:           0
yearFirst01:               2011
yearLast01:                2011
yearAlign01:               2011
stream_vectors01:          "u:v"
stream_mesh_file01:        DATM_INPUT/cfsr_mesh.nc
stream_lev_dimname01:      null
stream_data_files01:       DATM_INPUT/cfsr.201110.nc
stream_data_variables01:  "slmsksfc Sa_mask" "DSWRF Faxa_swdn" "DLWRF Faxa_lwdn" "vbdsf_
↪ave Faxa_swvdr" "vddsf_ave Faxa_swvdf" "nbdsf_ave Faxa_swndr" "nddsf_ave Faxa_swndf"
↪"u10m Sa_u10m" "v10m Sa_v10m" "hgt_hyblev1 Sa_z" "psurf Sa_pslv" "tmp_hyblev1 Sa_tbot"
↪"spfh_hyblev1 Sa_shum" "ugrd_hyblev1 Sa_u" "vgrd_hyblev1 Sa_v" "q2m Sa_q2m" "t2m Sa_t2m
↪" "pres_hyblev1 Sa_pbot" "precp Faxa_rain" "fprecp Faxa_snow"
```

4.2.7 datm_in

Table 4.30: *Parameters that can be set in a data stream namelist file (datm_in) at run-time.*

Parameter	Meaning
datamode	data mode (such as CFSR, GEFS, etc.)
factorfn_data	file containing correction factor for input data
factorfn_mesh	file containing correction factor for input mesh
flds_co2	if true, prescribed co2 data is sent to the mediator
flds_presaero	if true, prescribed aerosol data is sent to the mediator
flds_wiso	if true, water isotopes data is sent to the mediator
iradsw	the frequency to update the shortwave radiation in number of steps (or hours if negative)
model_maskfile	data stream mask file name
model_meshfile	data stream mesh file name
nx_global	number of grid points in zonal direction
ny_global	number of grid points in meridional direction
restfilm	model restart file namelist

A sample of the data stream namelist file is shown below:

```
&datm_nml
datamode = "CFSR"
factorfn_data = "null"
factorfn_mesh = "null"
flds_co2 = .false.
flds_presaero = .false.
flds_wiso = .false.
iradsw = 1
model_maskfile = "DATM_INPUT/cfsr_mesh.nc"
model_meshfile = "DATM_INPUT/cfsr_mesh.nc"
nx_global = 1760
ny_global = 880
restfilm = "null"
/
```

4.2.8 blkdat.input

The HYCOM model reads parameters from a custom formatted configuraiton file, blkdat.input. The HYCOM User's Guide provides an in depth description of the configuration settings.

4.2.9 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 CCPP Scientific Documentation provides an in-depth description of the namelist settings. Information describing the various physics-related namelist records can be viewed here.
- The Stochastic Physics Documentation describes the stochastic physics namelist records.
- The FV3 Dynamical Core Technical Documentation describes some of the other namelist records (dynamics, grid, etc).
- 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 4.31.

Table 4.31: *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_warr	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 4.2 contains a brief description of the climatological information in these files. The variables used in &namsfc to set the filenames are described in Table 4.32.

Table 4.32: *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 4.33.

Table 4.33: *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.
ccpp_suite	Name of the CCPP physics suite	character(len=256)	FV3_GFS_v15p2, set in `build.sh`
avg_max_len	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.
  ccpp_suite = 'FV3_GFS_v16beta'
/
```

The namelist section relating to the FMS diagnostic manager `&diag_manager_nml` is described in Section 4.4.1.

gfs_physics_nml

The namelist section `&gfs_physics_nml` contains physics-related information used by the atmosphere model and some of the variables are only relevant for specific parameterizations and/or configurations. The small set of variables used in `&gfs_physics_nml` are described in Table 4.34.

Table 4.34: *List of common variables in the *gfs_physics_nml* namelist section.

Variable Name	Description	Data Type	Default Value
cplflx	Flag to activate atmosphere-ocean coupling. If true, turn on receiving exchange fields from other components such as ocean.	logical	.false.
use_med_flu:	Flag to receive atmosphere-ocean fluxes from mediator. If true, atmosphere-ocean fluxes will be received into the CCPP physics and used there, instead of calculating them.	logical	.false.

A sample subset of this namelist is shown below:

```
&gfs_physics_nml
  use_med_flux = .true.
  cplflx       = .true.
/
```

Additional variables for the `&gfs_physics_nml` namelist can be found in the `FV3/ccpp/data/GFS_typedefs.F90` file.

4.3 Output files

4.3.1 FV3Atm

The output files generated when running `fv3.exe` are defined in the `diag_table` 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 4.2.3.

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. 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 4.2.1. 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.

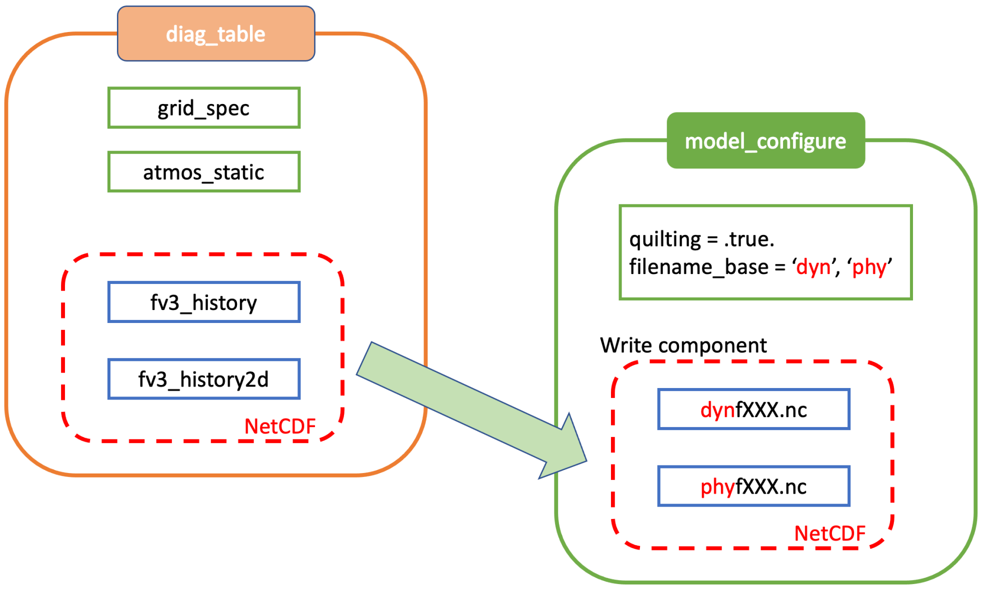

Fig. 4.1: Relationship between `diag_table`, `model_configure` and generated output files

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.

4.3.2 MOM6

MOM6 output is controlled via the FMS diag_manager using the `diag_table`. When MOM6 is present, the `diag_table` shown *above* includes additional requested MOM6 fields.

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

```
 #######################
 "ocn%4yr%2mo%2dy%2hr",      6,  "hours", 1, "hours", "time", 6, "hours", "1901 1 1 0 0 0
↪"
 "SST%4yr%2mo%2dy",      1,  "days",  1, "days",  "time", 1, "days",  "1901 1 1 0 0 0"
 ##############################################
 # static fields
 "ocean_model", "geolon",      "geolon",      "ocn%4yr%2mo%2dy%2hr", "all", .false.,
↪"none", 2
 "ocean_model", "geolat",      "geolat",      "ocn%4yr%2mo%2dy%2hr", "all", .false.,
↪"none", 2
 ...
# ocean output TSUV and others
 "ocean_model", "SSH",      "SSH",      "ocn%4yr%2mo%2dy%2hr","all",.true.,"none",2
 "ocean_model", "SST",      "SST",      "ocn%4yr%2mo%2dy%2hr","all",.true.,"none",2
```

(continues on next page)

(continued from previous page)

```
 "ocean_model", "SSS",      "SSS",      "ocn%4yr%2mo%2dy%2hr","all",.true.,"none",2
 ...
# save daily SST
 "ocean_model", "geolon",      "geolon",      "SST%4yr%2mo%2dy", "all", .false., "none",␣
↪2
 "ocean_model", "geolat",      "geolat",      "SST%4yr%2mo%2dy", "all", .false., "none",␣
↪2
 "ocean_model", "SST",         "sst",         "SST%4yr%2mo%2dy", "all", .true.,  "none",␣
↪2

# Z-Space Fields Provided for CMIP6 (CMOR Names):
#===============================================
 "ocean_model_z","uo","uo"      ,"ocn%4yr%2mo%2dy%2hr","all",.true.,"none",2
 "ocean_model_z","vo","vo"      ,"ocn%4yr%2mo%2dy%2hr","all",.true.,"none",2
 "ocean_model_z","so","so"      ,"ocn%4yr%2mo%2dy%2hr","all",.true.,"none",2
 "ocean_model_z","temp","temp"  ,"ocn%4yr%2mo%2dy%2hr","all",.true.,"none",2

# forcing
 "ocean_model", "taux",      "taux",          "ocn%4yr%2mo%2dy%2hr","all",.true.,"none",2
 "ocean_model", "tauy",      "tauy",          "ocn%4yr%2mo%2dy%2hr","all",.true.,"none",2
 ...
```

4.3.3 HYCOM

HYCOM output configuration is set in the *blkdat.input* file. A few common configuration options are described in Table 4.35

Table 4.35: *The following table describes HYCOM output configuration.*

Parameter	Description
dsurfq	Number of days between model diagnostics at the surface
diagfq	Number of days between model diagnostics
meanfq	Number of days between model time averaged diagnostics
rstrfq	Number of days between model restart output
itest	i grid point where detailed diagnostics are desired
jtest	j grid point where detailed diagnostics are desired

HYCOM outpus multiple datasets. These datasets contain both dot-a (.a), dot-b (.b), and dot-txt (.txt) files. Dot-a files contain data written as 32-bit IEEE real values (idm*jdm). Dot-b files contain plain text metadata for each field in the dot-a file. Dot-txt files contain plain text data for a single cell for profiling purposes. Post-processing utilties are available in the HYCOM-tools repository.

Table 4.36: *The following table describes HYCOM output files.*

Filename	Description
archs.YYYY_DDD_HH.(a,b,txt)	HYCOM surface archive data
archv.YYYY_DDD_HH.(a,b,txt)	HYCOM archive data
restart_out.(a,b)	HYCOM restart files

4.3.4 CICE6

CICE6 output is controlled via the namelist `ice_in`. The relevant configuration settings are

```
...
 histfreq       = 'm','d','h','x','x'
 histfreq_n     =  0 , 0 , 6 , 1 , 1
 hist_avg       = .true.
...
```

In this example, `histfreq_n` and `hist_avg` specify that output will be 6-hour means. No monthly (`m`), daily (`d`), yearly (`x`) or per-timestep (`x`) output will be produced.The `hist_avg` can also be set `.false.` to produce, for example, instaneous fields every 6 hours.

The output of any field is set in the appropriate `ice_in` namelist. For example,

```
...
&icefields_nml
f_aice         = 'mdhxx'
f_hi           = 'mdhxx'
f_hs           = 'mdhxx'
...
```

where the ice concentration (*aice*), ice thickness (*hi*) and snow thickness (*hs*) are set to be output on the monthly, daily, hourly, yearly or timestep intervals set by the *histfreq_n* setting. Since *histfreq_n* is *0* for both monthly and daily frequencies and neither yearly nor per-timestep output is requested, only 6-hour mean history files will be produced.

Further details of the configuration of CICE model output can be found in the CICE documentation Section 3.1.4.

4.3.5 WW3

The run directory includes WW3 binary outputs for the gridded outputs (`YYYYMMDD.HHMMSS.out_grd.<grd>`), point outputs (`YYYYMMDD.HHMMSS.out_pnt.points`) and restart files (`YYYYMMDD.HHMMSS.restart.<grd>`).

4.3.6 CMEPS

The CMEPS mediator writes general information about the run-time configuration to the file `mediator.log` in the model run directory. Optionally, the CMEPS mediator can be configured to write history files for the purposes of examining the field exchanges at various points in the model run sequence.

4.4 Additional Information about the FMS Diagnostic Manager

The FMS (Flexible Modeling System) diagnostic manager (`FMS/diag_manager`) manages the output for the ATM and, if present, the MOM6 component in the UFS Weather Model. It is configured using the `diag_table` file. Data can be written at any number of sampling and/or averaging intervals specified at run-time.

4.4.1 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 4.37. See `FMS/diag_manager/diag_manager.F90` for the complete list or view the FMS documentation for additional information.

Table 4.37: *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.
/
```

4.5 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 4.2.3. 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

CONFIGURATIONS

The UFS Weather Model (WM) can be run in any of several configurations, from a single-component atmospheric model to a fully coupled model with multiple earth system components (e.g., atmosphere, ocean, sea-ice, land, and mediator). This chapter documents a few of the currently supported configurations. For a full list of supported configurations, view the rt.conf file.

Attention: This chapter is a work in progress. There are a multitude of options for configuring the UFS WM, and this chapter merely details a few supported configurations. It will be expanded over time to include the full set of configurations supported for WM regression tests (RTs).

Table 5.1: *Documented UFS Weather Model Configuration Categories*

Configuration Category	Description
ATM	Standalone Atmospheric Model (*ATM*)
ATML	Coupled *ATM* and *LND*
LND	Coupled *CDEPS* - *DATM* - *LND* -*CMEPS*
RRFS	*ATM* with *data assimilation*

This chapter details the supported build/run options for each supported configuration. Click on the configuration category in Table 5.1 to go to that section. Each configuration category includes sample code for setting `CMAKE_FLAGS` and `CCPP_SUITES`. Additionally, there is a list of preferred physics suites, examples of `ufs.configure` files, and links to information on other input files required to run the model.

5.1 Background

Each RT configuration file (located in the `ufs-weather-model/tests/tests` directory) sets default variables by calling setup functions from `ufs-weather-model/tests/default_vars.sh` (see defaults here). Then, the RT configuration file sets test-specific variablesthese values will override the defaults. For example, the `control_c48` test file sets a list of files that it will use, calls the `export_fv3` function from `default_vars.sh`, and then exports test-specific variables. An excerpt is included below (`...` indicates omitted lines):

```
export LIST_FILES="sfcf000.nc \
                sfcf024.nc \
                atmf000.nc \
                atmf024.nc \
```

(continues on next page)

(continued from previous page)

```
                RESTART/20210323.060000.coupler.res \
                RESTART/20210323.060000.fv_core.res.nc \
                ...
                RESTART/20210323.060000.sfc_data.tile5.nc \
                RESTART/20210323.060000.sfc_data.tile6.nc"

export_fv3

export INPES=1
export JNPES=1
export WRTTASK_PER_GROUP=2
export NPZ=127
export NPZP=128
export NPX=49
export NPY=49
export DT_ATMOS=1200
...
```

`default_vars.sh` contains eight functions that set defaults for different types of tests. Table 5.2 describes what each function does.

Table 5.2: *default_vars.sh functions*

Function Name	Description
export_fv3	Set variables to the FV3 default values (first common variables, then model-specific ones). Different machines may have different defaults for some variables.
export_cpl	Set variables to the default values for *coupled* / S2S configurations.
ex-port_35d_run	Set variables to the default values for EMC's weekly coupled benchmark 35d tests (see rt_35d.conf).
ex-port_datm_c	Set variables to the default values for configurations that use the data atmosphere (*DATM*) component.
ex-port_hafs_da	Set variables to the default values for HAFS configurations that use the data atmosphere (DATM) component.
ex-port_hafs_do	Set variables to the default values for HAFS configurations that use the data ocean (*DOCN*) component.
ex-port_hafs_re	Set variables to the default values for regional HAFS configurations.
ex-port_hafs	Set variables to the default values for HAFS configurations.

Multiple `default_vars.sh` functions may be called in a given test. Values set in one function will be overridden when the same values are set in a subsequent function.

The most up-to-date list of `develop` branch data required for each test is available in the UFS WM RT Data Bucket. Users should click on "Browse Bucket" and navigate to the most recent date (in `develop-YYYY-MM-DD` format). Then, users should select *Intel* or *GNU* based on the compiler used in the test they want to run and then select the test name to see the required data.

5.2 Atmospheric Model Configurations

The atmospheric model configurations all use the UFS WM atmospheric component and may couple it with other models (e.g., a wave or aerosol model).

5.2.1 ATM - Standalone Atmospheric Model

The standalone atmospheric model (*ATM*) is an *FV3*-based prognostic atmospheric model that can be used for short- and medium-range research and operational forecasts. In standalone mode, `ATM` is not coupled to any other model.

Current ATM regression tests cover a wide variety of functionality and involve several physics tests. Table 5.3 contains a small selection of ATM-only RTs; it will be expanded to cover the full range of ATM-only supported configurations in time:

Table 5.3: *ATM regression test descriptions*

Test Name	Description	Physics Suite (see namelist options)	DT_ATMO	Start Date	Forecast Length (hours)
control_c48	Compare global control C48L127 results with previous trunk version	FV3_GFS_	1200	2021-03-22 06:00:00	24
control_p8	Compare global control results with previous trunk version	FV3_GFS_	720	2021-03-22 06:00:00	24
regional_cont	FV3 regional control (hi-res 3km, small domain) test	FV3_GFS_	1800	2016-10-03 00:00:00	6

Sample `CMAKE_FLAGS` Setting

```
export CMAKE_FLAGS="-DAPP=ATM -DCCPP_SUITES=FV3_GFS_v16,FV3_GFS_v17_p8,FV3_GFS_v15_
↪thompson_mynn_lam3km -D32BIT=ON"
```

Supported Physics Suites

Table 5.4: *Physics suites used in the ATM configurations above*

Physics Suite	Description
FV3_GFS_v16	The *CCPP* GFS_v16 physics suite is described in the CCPP documentation here.
FV3_GFS_v17_p	The CCPP GFS_v17_p8 physics suite is described in the CCPP documentation here.
FV3_GFS_v15_th	The CCPP GFS_v15 physics suite with the Thompson Aerosol-Aware Cloud Microphysics Scheme (see here) and Mynn Surface Layer Module (see here) tailored for a limited area model (LAM) 3-km resolution grid.

Additional Information

Input files required for ATM configurations can be viewed in Section 4.1.1 or in the UFS WM RT Data Bucket. Information on `ufs.configure` files is available in Section 4.2.4, and a sample ATM `ufs.configure` file (`ufs.configure.atm.IN`) is available here.

5.2.2 ATMW

COMING SOON!

5.2.3 ATMAERO

COMING SOON!

5.2.4 ATMAQ

COMING SOON!

5.2.5 ATML

The ATML configuration couples *ATM* with *LND*. These tests use default values set in the `export_fv3` function of `default_vars.sh`.

Attention: There is an issue with `-D32BIT=ON` in the ATM-LND tests, and NoahMP requires r8 libraries.

Table 5.5: *ATML regression test descriptions*

Test Name	Description	Physics Suite (see namelist options)	DT_ATMO	Start Date	Forecast Length (hours)
control_p8_atm	Compare global control results with previous trunk version	FV3_GFS_	720	2021-03-22 06:00:00	24

Sample `CMAKE_FLAGS` Setting

```
export CMAKE_FLAGS="-DAPP=ATML -DCCPP_SUITES=FV3_GFS_v17_p8"
```

Supported Physics Suites

Table 5.6: *Physics suites used in the ATM configurations above*

Physics Suite	Description
FV3_GFS_v17_p	The *CCPP* GFS_v17_p8 physics suite is described in the CCPP documentation here.

Additional Information

Input files required for ATML configurations can be viewed in Section 4.1.1 (ATM) and Section 4.1.9 (LND) or in the UFS WM RT Data Bucket. Information on `ufs.configure` files is available in Section 4.2.4, and a sample ATML `ufs.configure` file (`ufs.configure.atm_lnd.IN`) is available here.

5.3 Rapid Refresh Forecast System (RRFS)

The RRFS configurations use an *ATM*-only configuration on a high-resolution regional grid with data assimilation capabilities. These tests use the default values set in the `export_fv3`, `export_rap_common`, `export_rrfs_v1`, and/or `export_hrrr_conus13km` functions of `default_vars.sh` unless other values are explicitly set in a given test file. In all tests, the values in `export_fv3` are set first. Depending on the test, some of these values may be overriden by `export_rrfs_v1` (which includes values from `export_rap_common`) or `export_hrrr_conus13km`. Table 5.7 compares the values set in `export_fv3` to the values set in the other functions.

Note: `export_rrfs_v1` calls `export_rap_common`, which calls `export_fv3`. Values from `export_fv3` are set first, followed by values in `export_rap_common` and then values in `export_rrfs_v1`. Values in italics indicate that the value is inherited from a previously-called function.

Table 5.7: *RRFS Default Variables*

Variable	`export_fv`	`export_ra`	`export_rr`	`export_hrrr_conus13km`
DATE (SYEAR-SMONTH-SDAY SHOUR:00:00)	2016-10-03 00:00:00	2021-03-22 06:00:00	*2021-03-22 06:00:00*	2021-05-12 16:00:00
Forecast Length in hours (FHMAX)	24	*24*	*24*	2
CCPP_SUITE	None set (set in subsequent functions or test file)	None set	FV3_RRFS	FV3_HRRR
IMP_PHYSICS	11	8	*8*	8
DT_ATMOS	1800	300	*300*	120
OUTPUT_GRID	"'cubed_sp	'gaussian_grid'	*'gaussian_grid'*	lambert_conformal
NTILES	6	*6*	*6*	1
WRITE_DOPOST	.false.	.true.	*.true.*	.false.
NSTF_NAME	2,1,1,0,5	'2,0,0,0,0'	*'2,0,0,0,0'*	*'2,0,0,0,0'*
IAER	111	5111	*5111*	1011
NPX	97	*97*	*97*	397
NPY	97	*97*	*97*	233
NPZ	64	127	*127*	65
NPZP	65	128	*128*	66
INPES	3 (`$INPES_d` — set in machine section)	*3*	*3*	12
JNPES	8 (`$JNPES_d` — set in machine section)	*8*	*8*	12
UFS_CONFIGURE	ufs.configu	*ufs.configu*	*ufs.configu*	*ufs.configure.atm.IN*
MODEL_CONFIGURE	model_con	*model_con*	*model_con*	model_configure_rrfs_conus1

continues on next page

Table 5.7 – continued from previous page

Variable	`export_fv`	`export_ra`	`export_rr`	`export_hrrr_conus13km`
DIAG_TABLE	diag_table_	*diag_table_*	diag_table_	diag_table_hrrr
DIAG_TABLE_ADDITIONAL	Not set	Not set	Not set	diag_additional_rrfs_smoke
FIELD_TABLE	field_table_	field_table_	*field_table_*	field_table_thompson_aero_tl
FV3_RUN	None set	con-trol_run.IN	*con-trol_run.IN*	rrfs_warm_run.IN
INPUT_NML	None set	rap.nml.IN	*rap.nml.IN*	rrfs_conus13km_hrrr.nml.IN
MAKE_NH	.true.	*.true.*	*.true.*	.false.
NA_INIT	1	*1*	*1*	0
LHEATSTRG	.true.	.false.	.false.	.false.
SEDI_SEMI	.true.	*.true.*	*.true.*	.false.
DECFL	10	*10*	*10*	8
RRFS_SMOKE	.false.	*.false.*	*.false.*	.true.
SEAS_OPT	2	*2*	*2*	0
LKM	0	*0*	*0*	1
SFCLAY_COMPUTE_FLUX	.false.	*.false.*	*.false.*	.true.
ICLIQ_SW	1	*1*	*1*	2
IOVR	1	*1*	*1*	3
KICE	2	9	*9*	9
EXTERNAL_IC	.true.	*.true.*	*.true.*	.false.
NGGPS_IC	.true.	*.true.*	*.true.*	.false.
MOUNTAIN	.false.	*.false.*	*.false.*	.true.
WARM_START	.false.	*.false.*	*.false.*	.true.
RES_LATLON_DYNAMICS	"”"	*"”"*	*"”"*	"’fv3_increment.nc’"
FHZERO	6	*6*	*6*	1.0
PRINT_DIFF_PGR	.false.	*.false.*	*.false.*	.true.
FHCYC	24	*24*	*24*	0.0
CNVCLD	.true.	*.true.*	*.true.*	.false.
CDMBWD	‘0.14,1.8,1. (`${CDMBWD`	*‘0.14,1.8,1.*	*‘0.14,1.8,1.*	‘3.5,1.0’
GWD_OPT	1	*1*	*1*	3
DO_GSL_DRAG_LS_BL	.false.	*.false.*	*.false.*	.true.
DO_GSL_DRAG_SS	.false.	*.false.*	*.false.*	.true.
DO_GSL_DRAG_TOFD	.false.	*.false.*	*.false.*	.true.
DNATS	1	0	*0*	0
DO_SAT_ADJ	.true.	.false.	*.false.*	.false.
IALB	1	2	*2*	2
IEMS	1	2	*2*	2
HYBEDMF	.true.	.false.	*.false.*	.false.
DO_MYNNEDMF	.false.	.true.	*.true.*	.true.
DO_MYNNSFCLAY	.false.	.true.	*.true.*	.true.
DO_MYJPBL	.false.	*.false.*	*.false.*	.true.
DO_DEEP	.true.	*.true.*	.false.	.false.
SHAL_CNV	.true.	*.true.*	.false.	.false.
IMFSHALCNV	2	*2*	-1	-1
IMFDEEPCNV	2	*2*	-1	-1
LSM	1	*1*	2	3
LSOIL_LSM	4	*4*	4	9
RESTART_INTERVAL	0	*0*	*0*	1
OUTPUT_FH	"12 -1"	*"12 -1"*	*"12 -1"*	*"12 -1"*

Current RRFS regression tests cover a wide variety of functionality and involve several physics tests. Table 5.8 (below) contains a selection of RTs for RRFS functionality. Blanks indicate that the value comes from the default setting file. These default values are listed in Table 5.7 above.

Table 5.8: *RRFS regression test descriptions*

Test Name	Description	Default Settings	Physics Parameters (see namelist options for variable definitions)	Fcst Lengt (hours	FIELD	Other.E
rrfs_v1beta	Compare RRFS_v1beta results with previous trunk version	`export_rrfs_v1`				RESTART_INTERVAL= -1”; OUTPUT_FH=’0 09 12’
rrfs_v1beta_d	Compare RRFS_v1beta debug results with previous trunk version	`export_rrfs_v1`		1		OUTPUT_FH=”0 1”
rrfs_v1nssl	Compare RRFS_v1nssl results with previous trunk version	`export_rrfs_v1`	CCPP_SUITE=FV3_RRFS_v1nssl IMP_PHYSICS=17 **Set to FALSE:** LTAEROSOL **Set to TRUE:** NSSL_CCN_ON; NSSL_HAIL_ON; NSSL_INVERTCCN **Set to VALUE:** NWAT=7		field_t	RESTART_INTERVAL= -1”; OUTPUT_FH=’0 09 12’
rrfs_v1nssl_n	Compare RRFS_v1nssl_ results with previous trunk version	`export_rrfs_v1`	CCPP_SUITE=FV3_RRFS_v1nssl IMP_PHYSICS=17 **Set to FALSE:** NSSL_CCN_ON; NSSL_HAIL_ON; LTAEROSOL **Set to TRUE:** NSSL_INVERTCCN **Set to VALUE:** NWAT=6		field_t	RESTART_INTERVAL -1”; OUTPUT_FH=’0 09 12’
conus13km_c	HRRR physics on 13km domain, control	`export_hrrr_conus`				RESTART_INTERVAL= QUILTING_RESTART=.false.
conus13km_d	HRRR physics on 13km domain, debug run	`export_hrrr_conus`		1		RESTART_INTERVAL= QUILTING_RESTART=.false.
conus13km_r	HRRR physics on 13km domain, restart run	`export_hrrr_conus`				FHROT=1;RESTART_F “%02d” $((${SHOUR} + ${FHROT})))0000”; RRFS_RESTART=YES QUILTING_RESTART=.false.
conus13km_2	HRRR physics on 13km domain, two threads	`export_hrrr_conus`		1		RESTART_INTERVAL= atm_omp_num_threads= QUILTING_RESTART=.false.; WRT_TASK_PER_GROUP=6
conus13km_d	HRRR physics on 13km do-	`export_hrrr_conus`		1		RESTART_INTERVAL= atm_omp_num_threads= WRT_

Sample CMAKE_FLAGS Setting

```
export CMAKE_FLAGS="-DAPP=ATM -DCCPP_SUITES=FV3_RAP,FV3_HRRR,FV3_RRFS_v1beta,FV3_RRFS_
↪v1nssl -D32BIT=ON"
```

Supported Physics Suites

Table 5.9: *Physics suites used in the RRFS configurations above*

Physics Suite	Description
FV3_HRRR	The FV3_HRRR physics suite is described in the *CCPP* documentation here.
FV3_RRFS_v1be	The FV3_RRFS_v1beta physics suite is described in the CCPP documentation here.
FV3_RRFS_v1ns	The FV3_RRFS_v1nssl physics suite is similar to the *FV3_RRFS_v1beta* suite; however, it uses the NSSL 2-moment microphysics scheme instead of the Thompson microphysics scheme.

Additional Information

Each test file lists the input files required for a given test. Input files required for RRFS ATM configurations can be downloaded from the UFS WM RT Data Bucket. Users who wish to run additional (unsupported) cases may also find useful data in the NOAA RRFS data bucket.

Information on `ufs.configure` files is available in Section 4.2.4. The supported RRFS WM RTs use the same `ufs.configure` file that ATM-only tests do (`ufs.configure.atm.IN`). This file can be viewed in the `ufs-weather-model/tests/parm` directory.

Additionally, users can find examples of various RRFS configuration files in the `ufs-weather-model/tests/parm` directory. These files include `model_configure_*`, `*_run.IN` (input run), `*.nml.IN` (input namelist), `field_table_*`, and `diag_table_*` files.

5.4 LND

The LND configuration couples *DATM*, *CDEPS*, and *CMEPS* with *LND*. These tests use default values set in the `export_datm_cdeps` function of `default_vars.sh`.

Table 5.10: *LND regression test descriptions*

Test Name	Description	Physics Suite	DT_ATMO	Start Date	Forecast Length (hours)
datm_cdeps	DATM_CDEPS_NOAHMP_GSWP3 - control	N/A	N/A	2000-01-01 00:00:00	24
datm_cdeps	DATM_CDEPS_NOAHMP_GSWP3_RST - control restart	N/A	N/A	2000-01-01 12:00:00	12

Sample CMAKE_FLAGS Setting

```
export CMAKE_FLAGS="-DAPP=LND"
```

Additional Information

Input files required for LND configurations can be viewed in Section 4.1.9 (LND) or in the UFS WM RT Data Bucket. Information on `ufs.configure` files is available in Section 4.2.4, and a sample ATML `ufs.configure` file (`ufs.configure.atm_lnd.IN`) is available here.

5.5 Seasonal to Subseasonal (S2S) Configurations

COMING SOON!

5.6 NG-GODAS

COMING SOON!

5.7 Hurricane Analysis and Reforecast System Configurations

COMING SOON!

CHAPTER

SIX

CONFIGURATION PARAMETERS

6.1 Build Configuration Parameters

6.1.1 Configuration Options

-DAPP:

Sets the *WM* configuration to build. Valid values: `ATM`, `ATMW`, `ATMAERO`, `ATMAQ`, `S2S`, `S2SA`, `S2SW`, `S2SWA`, `NG-GODAS`, `HAFS`, `HAFSW`, `HAFS-ALL`

6.1.2 Physics Options

-DCCPP_SUITES:

Sets the physics suites that will be made available when the *WM* is built.

Physics suites supported in regression testing:

```
FV3_GFS_cpld_rasmgshocnsstnoahmp_ugwp
FV3_GFS_v15p2
FV3_GFS_v15_thompson_mynn
FV3_GFS_v15_thompson_mynn_lam3km
FV3_GFS_v16
FV3_GFS_v16_csawmg
FV3_GFS_v16_fv3wam
FV3_GFS_v16_noahmp
FV3_GFS_v16_ras
FV3_GFS_v16_ugwpv1
FV3_GFS_v17_p8
FV3_GFS_v17_p8_rrtmgp
FV3_GFS_v17_coupled_p8
FV3_GFS_v17_coupled_p8_sfcocn
FV3_HAFS_v0_gfdlmp_tedmf
FV3_HAFS_v0_gfdlmp_tedmf_nonsst
FV3_HAFS_v0_thompson_tedmf_gfdlsf
FV3_HRRR
FV3_HRRR_smoke
FV3_RAP
```

```
FV3_RAP_RRTMGP
FV3_RAP_sfcdiff
FV3_RRFS_v1beta
FV3_RRFS_v1nssl
```

Other valid values:

```
FV3_CPT_v0
FV3_GFS_2017
FV3_GFS_2017_csawmg
FV3_GFS_2017_csawmgshoc
FV3_GFS_2017_gfdlmp
FV3_GFS_2017_gfdlmp_noahmp
FV3_GFS_2017_gfdlmp_regional
FV3_GFS_2017_gfdlmp_regional_c768
FV3_GFS_2017_h2ophys
FV3_GFS_2017_myj
FV3_GFS_2017_ntiedtke
FV3_GFS_2017_ozphys_2015
FV3_GFS_2017_sas
FV3_GFS_2017_satmedmf
FV3_GFS_2017_satmedmfq
FV3_GFS_2017_shinhong
FV3_GFS_2017_stretched
FV3_GFS_2017_ysu
FV3_GFS_cpld_rasmgshoc
FV3_GFS_cpld_rasmgshocnsst
FV3_GFS_cpld_rasmgshocnsst_flake
FV3_GFS_cpld_rasmgshocnsst_ugwp
FV3_GFS_cpldnst_rasmgshoc
FV3_GFS_rasmgshoc
FV3_GFS_v15
FV3_GFS_v15_gf
FV3_GFS_v15_gf_thompson
FV3_GFS_v15_mynn
FV3_GFS_v15_ras
FV3_GFS_v15_rasmgshoc
FV3_GFS_v15_thompson
FV3_GFS_v15p2_no_nsst
FV3_GFS_v15plus
FV3_GFS_v15plusras
FV3_GFS_v16_coupled
FV3_GFS_v16_coupled_noahmp
FV3_GFS_v16_coupled_nsstNoahmp
FV3_GFS_v16_coupled_nsstNoahmpUGWPv1
FV3_GFS_v16_coupled_p8
```

```
FV3_GFS_v16_coupled_p8_sfcocn
FV3_GFS_v16_couplednsst
FV3_GFS_v16_flake
FV3_GFS_v16_no_nsst
FV3_GFS_v16_nsstNoahmpUGWPv1
FV3_GFS_v16_p8
FV3_GFS_v16_thompson
FV3_GFSv17alp_cpldnsstrasnoahmp
FV3_GFSv17alp_cpldnsstrasugwpnoahmp
FV3_GFSv17alp_cpldnsstsasugwpnoahmp
FV3_GFSv17alpha_cpldnsstras
FV3_GFSv17alpha_cpldnsstras_flake
FV3_GFSv17alpha_cpldnsstras_ugwp
FV3_GFSv17alpha_cpldnsstrasnoshal
FV3_GFSv17alpha_cpldnsstsas
FV3_GFSv17alpha_cpldnsstsas_ugwp
FV3_GFSv17alpha_ras
FV3_GFSv17alpha_ras_flake
FV3_GFSv17alpha_ras_ugwp
FV3_GFSv17alpha_sas
FV3_RAP_cires_ugwp
FV3_RAP_flake
FV3_RAP_noah
FV3_RAP_noah_sfcdiff_cires_ugwp
FV3_RAP_noah_sfcdiff_ugwpv1
FV3_RAP_noah_sfcdiff_unified_ugwp
FV3_RAP_unified_ugwp
FV3_RRFS_v1alpha
```

6.1.3 Other Build Options

`-DCMEPS_AOFLUX`: (Default: OFF)
Enables atmosphere-ocean flux calculation in mediator. Valid values: `ON` | `OFF`

`-DDEBUG`: (Default: OFF)
Enables DEBUG mode. Valid values: `ON` | `OFF`

`-D32BIT`: (Default: OFF)
Enables 32-bit, single precision arithmetic in dycore and fast physics. Valid values: `ON` | `OFF`

`-DCCPP_32BIT`: (Default: OFF)
Enables 32-bit, single precision arithmetic in slow physics. Valid values: `ON` | `OFF`

`-DMOVING_NEST`: (Default: OFF)
Enables moving nest code. Valid values: `ON` | `OFF`

`-DMULTI_GASES`: (Default: OFF)
Enable `MULTI_GASES`. Valid values: `ON` | `OFF`

CHAPTER

SEVEN

AUTOMATED TESTING

The UFS Weather Model repository on GitHub employs two types of automated testing:

1. CI/CD (Continuous Integration/Continuous Development) testing on the cloud
2. AutoRT on NOAA R&D platforms

Both are application level tests and utilize the regression testing framework discussed in Section 3.6.1.

7.1 CI/CD

The UFS Weather Model (*WM*) uses GitHub Actions (GHA), a GitHub-hosted continuous integration service, to perform CI/CD testing. Build jobs are done on GHA-provided virtual machines. Test jobs are performed on the Amazon Web Services (AWS) cloud platform using a number of EC2 instances. Builds and tests are carried out in a Docker container. The container includes a pre-installed version of the *HPC-Stack*, which includes all prerequisite libraries. Input data needed to run the tests are stored as a separate Docker container.

When a developer makes a pull request (PR) to the UFS WM repository, a code manager may add the *run-ci* label, which triggers the CI/CD workflow. The CI/CD workflow then executes the following steps:

1. A check is performed to make sure the UFS Weather Model and its first level subcomponents are up to date with the top of the `develop` branch.
2. If the check is successful, build jobs are started on GHA-provided virtual machines by downloading the HPC-Stack Docker container stored in Docker Hub.
3. Once all build jobs are successful, the created executable files are stored as artifacts in GHA.
4. A number of AWS EC2 instances are started.
5. Test jobs are started on AWS after downloading the HPC-Stack Docker container, the executable file from the build job, and the input-data Docker container.
6. When all tests are complete, EC2 instances are stopped. Test results are reported on GitHub.

The GHA-related `yaml` scripts are located in the `.github/workflows/` directory. `build_test.yml` is the main workflow file, and `aux.yml` is an auxiliary file responsible for (1) checking that the PR branch is up-to-date and (2) starting/stopping the EC2 instances.

Other CI-related scrips are located in the `tests/ci/` directory. `ci.sh` is the main script that invokes Docker build and run. `Dockerfile` is used to build the UFS Weather Model. Other shell and python scripts help with various tasks. For example:

- `repo_check.sh` checks that the PR branch is up-to-date.
- `check_status.py` checks the status of EC2 instances.
- `setup.py` and `ci.test` configure the test cases to execute in the CI/CD workflow.

7.2 Auto RT

The Automated Regression Testing (AutoRT) system is a python program that automates the process of regression testing on NOAA HPC platforms. It contains the files in Table 7.1 below:

Table 7.1: *Files for Automated Regression Testing (AutoRT) system*

File Name	Description
start_rt_auto.sh	Verifies HPC name, sets the python paths
rt_auto.py	Python interface between the HPC and the github API
jobs/bl.py	Functions for the baseline job
jobs/rt.py	Functions for the regression test job

7.2.1 AutoRT Workflow

On supported HPC systems, a *cron job* runs the `start_rt_auto.sh` bash script every 15 minutes. This script checks the HPC name and sets certain python paths. Then, it runs `rt_auto.py`, which uses the Github API (through pyGitHub) to check the labels on pull requests to `ufs-weather-model`. If a PR label matches the HPC name (e.g., hera-intel-RT or derecho-gnu-BL), the label provides the HPC with the compiler and job information to run a test or task on the machine. If no PR label matches HPC name, the script exits.

For example, a PR labeled `gaea-intel-BL` will be recognized by the HPC machine 'Gaea'. It will set the `RT_COMPILER` variable to 'intel' and run the baseline creation script (`bl.py`). This script creats a job class that contains all information from the machine that the job will need to run. That information is sent into the `jobs/rt[bl].py` script.

`rt.py` sets directories for storage, gets repo information, runs the regression test, and completes any required post processing.

```python
def run(job_obj):
    logger = logging.getLogger('RT/RUN')
    workdir = set_directories(job_obj)
    branch, pr_repo_loc, repo_dir_str = clone_pr_repo(job_obj, workdir)
    run_regression_test(job_obj, pr_repo_loc)
    post_process(job_obj, pr_repo_loc, repo_dir_str, branch)
```

`bl.py`: (similar to `rt.py`) Adds functionality to create baselines before running regression testing.

```python
    def run(job_obj):
        logger = logging.getLogger('BL/RUN')
        workdir, rtbldir, blstore = set_directories(job_obj)
        pr_repo_loc, repo_dir_str = clone_pr_repo(job_obj, workdir)
        bldate = get_bl_date(job_obj, pr_repo_loc)
        bldir = f'{blstore}/develop-{bldate}/{job_obj.compiler.upper()}'
        bldirbool = check_for_bl_dir(bldir, job_obj)
        run_regression_test(job_obj, pr_repo_loc)
        post_process(job_obj, pr_repo_loc, repo_dir_str, rtbldir, bldir)
```

CHAPTER

EIGHT

FAQ

8.1 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 3.4, 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 Derecho 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 `derecho.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    | derecho.
   ↪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 = derecho.* ]]; then stanza:
...
dprefix=/glade/scratch
```

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

```
if [[ $MACHINE_ID = derecho.* ]]; then
   RTPWD=${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
```

8.2 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.

8.3 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 8.1 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 8.1: *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

8.4 How do I turn off IO for the components of the coupled model?

8.4.1 FV3atm restart and history files

To turn off FV3atm restart files, set the `restart_interval` in `model_configure` to a value greater than the forecast length.

To turn off history files, in `model_configure` there are two options:

- Set `quilting` to .false., then in `diag_table`, remove the history output file definitions `fv3_history` and `fv3_history2d` and the associated fields. This will turn off the write_grid component and the number of tasks used by FV3atm must also be adjusted to remove the tasks assigned to the write grid component.
- Set `quilting` to .true., then in `model_configure` set `write_dopost` to .false. and set `output_fh` to a value greater than the forecast length. This will turn off the writing of output but the write grid component tasks will still be necessary.

8.4.2 MOM6, CICE6 and CMEPS restart files

In `nems.configure`, set the ALLCOMP_attribute `restart_n` to a value greater than the forecast length.

8.4.3 MOM6 history files

In the `diag_table` file, remove the `ocn` and `SST` history output file definitions and fields.

MOM6 history output speed can also be increased by setting the `IO_LAYOUT` parameter in `INPUT/MOM_input`.

```
IO_LAYOUT = 4,2
```

8.4.4 CICE history files

In the CICE namelist `ice_in`, set the `histfreq` to none with

```
histfreq = 'x','x','x','x','x'
```

The initial condition file can be turned off using

```
write_ic = .false.
```

8.4.5 GOCART history files

In AERO_HISTORY.rc, remove all the fields listed in `COLLECTIONS`

```
COLLECTIONS:
::
```

8.4.6 WW3 history and restart files

In `ww3_shel.inp`, change the output interval for gridded frequency from 3600 to 0 on line 68. To turn off point output, change the output frequency from 900 to 0 on line 296. To turn off restart files, change the frequency from 3600 to 0 on line 321.

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

In the UFS WM, each component's MPI task information, including the starting and ending tasks and the number of threads, are specified using the component-specific `petlist_bounds` and `omp_num_threads` in `nems.configure`. In general, the total number of MPI tasks required is the sum of all the sub-component tasks, as long as those components do not overlap (i.e., share the same PETs). An example of a global 5 component coupled configuration nems.configure at the end of this section.

8.5.1 FV3atm

The FV3atm component consists of one or more forecast grid components and write grid components.

The MPI tasks for the forecast grid components are specified in the layout variable in one or more namelist files `input*.nml` (e.g. input.nml and input_nest02.nml). The total number of mpi tasks required is given by the product of the specified layout, summed over all domains. For example, for a global domain with 6 tiles and `layout = 6,8`, the total number required is `6*6*8 = 288`. For two regional domains using `input.nml` and `input_nest02.nml`, each with `layout = 6,10`, the total required is the sum `6*10 + 6*10 = 120`.

For the global configuration, an additional requirement is that the layout specified must be a multiple of the `blocksize` parameter in `input.nml`. For example, using `layout=8,8` for C96 yields subdomains of `12 x 12`. The subdomain product is `12*12 = 144`, which is not divisible by a `blocksize=32`. Therefore, the C96 does not support an `8,8` layout for a blocksize of 32. If `layout = 4,6`, the subdomain product is `24*16 = 384`, which is divisible by a `blocksize=32`. A layout of `4,6` is supported for C96 with a blocksize of 32.

The FV3atm will utilize the write grid component if `quilting` is set to .true. In this case, the required mpi tasks for the write grid components is the product of the `write_groups` and the `write_tasks_per_group` in the `model_configure` file.

```
quilting:                .true.
write_groups:            1
write_tasks_per_group:   60
```

In the above case, the write grid component requires 60 tasks.

The total number of MPI ranks for FV3atm is the sum of the forecast tasks and any write grid component tasks.

```
total_tasks_atm = forecast tasks +  write grid component tasks
```

If ESMF-managed threading is used, the total number of PETs for the atmosphere component is given by the product of the number of threads requested and the total number of MPI ranks (both forecast and write grid component). If `num_threads_atm` is the number of threads specified for the FV3atm component, in `nems.configure` the ATM PET bounds are given by

```
ATM_petlist_bounds     0 total_tasks_atm*num_threads_atm-1
ATM_omp_num_threads    num_threads_atm
```

Note that in UWM, the ATM component is normally listed first in `nems.configure` so that the starting PET for the ATM is 0.

8.5.2 GOCART

GOCART shares the same grid and forecast tasks as FV3atm but it does not have a separate write grid component in its NUOPC CAP. Also, while GOCART does not have threading capability, it shares the same data structure as FV3atm and so it has to use the same number of threads used by FV3atm. Therefore, the total number of MPI ranks and threads in GOCART is the same as the those for the FV3atm forecast component (i.e., excluding any write grid component). Currently GOCART only runs on the global forecast grid component, for which only one namelist is needed.

```
total_tasks_chm = FV3atm forecast tasks

CHM_petlist_bounds:             0 total_tasks_chm*num_threads_atm-1
CHM_omp_num_threads:            num_threads_atm
```

8.5.3 CMEPS

The mediator MPI tasks can overlap with other components and in UFS the tasks are normally shared on the FV3atm forecast tasks. However, a large number of tasks for the mediator is generally not recommended since it may cause slow performance. This means that the number of MPI tasks for CMEPS is given by

```
total_tasks_med = smaller of (300, FV3atm forecast tasks)
```

and in `nems.configure`

```
MED_petlist_bounds:             0 total_tasks_med*num_threads_atm-1
MED_omp_num_threads:            num_threads_atm
```

8.5.4 MOM6

For MOM6 the only restriction currently on the number of MPI ranks used by MOM6 is that it is divisible by 2. The starting PET in `nems.configure` will be the last PET of the preceding component, incremented by one. Threading in MOM6 is not recommended at this time.

```
OCN_petlist_bounds:             starting_OCN_PET  total_tasks_ocn+starting_OCN_PET-1
OCN_omp_num_threads:            1
```

8.5.5 CICE

CICE requires setting the decomposition shape, the number of requested processors and the calculated block sizes in the `ice_in` namelist. In UFS, the decomposition shape is always `SlenderX2`, except for the 5 deg configuration, which is `SlenderX1`.

For `SlenderX2` decomposition, a given `nprocs`, and global domain `nx_global`, `ny_global`, the block sizes are given by

```
block_size_y = ny_global/2
block_size_x = nx_global/(nprocs/2)
```

Similarily, for `SlenderX1`

```
block_size_y = ny_global
block_size_x = nx_global/nprocs
```

For the 1-deg CICE domain for example, `ice_in` would be

```
nprocs            = 10
nx_global         = 360
ny_global         = 320
block_size_x      = 72
block_size_y      = 160
max_blocks        = -1
processor_shape   = 'slenderX2'
```

In UFS, only a single thread is used for CICE so for `nprocs` set in `ice_in`, the tasks in `nems.configure` are set as:

```
ICE_petlist_bounds:             starting_ICE_PET  nprocs+starting_ICE_PET-1
ICE_omp_num_threads:            1
```

The starting ICE PET in `nems.configure` will be the last PET of the preceding component, incremented by one.

8.5.6 WW3

The WW3 component requires setting only the MPI ranks available for WW3 and the number of threads to be used.

```
WAV_petlist_bounds:           starting_WAV_PET  num_tasks_wav*num_threads_wav+starting_WAV_
↪PET-1
WAV_omp_num_threads:          num_threads_wav
```

The starting WAV PET in `nems.configure` will be the last PET of the preceding component, incremented by one.

8.5.7 Example: 5-component nems.configure

For the fully coupled S2SWA application, a sample `nems.configure` is shown below :

```
#############################################
####  NEMS Run-Time Configuration File  #####
#############################################

# ESMF #
logKindFlag:            ESMF_LOGKIND_MULTI
globalResourceControl:  true

# EARTH #
EARTH_component_list: MED ATM CHM OCN ICE WAV
EARTH_attributes::
  Verbosity = 0
::

# MED #
MED_model:                      cmeps
MED_petlist_bounds:             0 767
MED_omp_num_threads:            2
::


# ATM #
ATM_model:                      fv3
ATM_petlist_bounds:             0 863
ATM_omp_num_threads:            2
ATM_attributes::
  Verbosity = 0
  DumpFields = false
  ProfileMemory = false
  OverwriteSlice = true
::

 # CHM #
 CHM_model:                      gocart
 CHM_petlist_bounds:             0 767
 CHM_omp_num_threads:            2
 CHM_attributes::
   Verbosity = 0
 ::

 # OCN #
 OCN_model:                      mom6
 OCN_petlist_bounds:             864 983
 OCN_omp_num_threads:            1
 OCN_attributes::
   Verbosity = 0
   DumpFields = false
   ProfileMemory = false
   OverwriteSlice = true
```

(continues on next page)

(continued from previous page)

```
  mesh_ocn = mesh.mx025.nc
::

# ICE #
ICE_model:                        cice6
ICE_petlist_bounds:               984 1031
ICE_omp_num_threads:              1
ICE_attributes::
  Verbosity = 0
  DumpFields = false
  ProfileMemory = false
  OverwriteSlice = true
  mesh_ice = mesh.mx025.nc
  stop_n = 3
  stop_option = nhours
  stop_ymd = -999
::

# WAV #
WAV_model:                        ww3
WAV_petlist_bounds:               1032 1191
WAV_omp_num_threads:              2
WAV_attributes::
  Verbosity = 0
  OverwriteSlice = false
  diro = "."
  logfile = wav.log
  mesh_wav = mesh.gwes_30m.nc
  multigrid = false
::

CMEPS warm run sequence
runSeq::
@1800
MED med_phases_prep_ocn_avg
MED -> OCN :remapMethod=redist
OCN
@300
  MED med_phases_prep_atm
  MED med_phases_prep_ice
  MED med_phases_prep_wav_accum
  MED med_phases_prep_wav_avg
  MED -> ATM :remapMethod=redist
  MED -> ICE :remapMethod=redist
  MED -> WAV :remapMethod=redist
  ATM phase1
  ATM -> CHM
  CHM
  CHM -> ATM
  ATM phase2
  ICE
  WAV
```

(continues on next page)

(continued from previous page)

```
   ATM -> MED :remapMethod=redist
   MED med_phases_post_atm
   ICE -> MED :remapMethod=redist
   MED med_phases_post_ice
   WAV -> MED :remapMethod=redist
   MED med_phases_post_wav
   MED med_phases_prep_ocn_accum
 @
 OCN -> MED :remapMethod=redist
 MED med_phases_post_ocn
 MED med_phases_restart_write
@
::

# CMEPS variables

DRIVER_attributes::
::

MED_attributes::
  ATM_model = fv3
  ICE_model = cice6
  OCN_model = mom6
  WAV_model = ww3
  history_n = 1
  history_option = nhours
  history_ymd = -999
  coupling_mode = nems_frac
  history_tile_atm = 384
::
ALLCOMP_attributes::
  ScalarFieldCount = 2
  ScalarFieldIdxGridNX = 1
  ScalarFieldIdxGridNY = 2
  ScalarFieldName = cpl_scalars
  start_type = startup
  restart_dir = RESTART/
  case_name = ufs.cpld
  restart_n = 3
  restart_option = nhours
  restart_ymd = -999
  dbug_flag = 0
  use_coldstart = false
  use_mommesh = true
  eps_imesh = 1.0e-1
  stop_n = 6
  stop_option = nhours
  stop_ymd = -999
::
```

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

TEN

GLOSSARY

advect
To transport substances in the atmostphere by *advection*.

advection
According to the American Meteorological Society (AMS) definition, advection is "The process of transport of an atmospheric property solely by the mass motion (velocity field) of the atmosphere." In common parlance, advection is movement of atmospheric substances that are carried around by the wind.

ATM
The Weather Model configuration that runs only the standalone atmospheric model.

AQM
The Air Quality Model (AQM) is a UFS Application that dynamically couples the Community Multiscale Air Quality (*CMAQ*) model with the UFS Weather Model through the *NUOPC* Layer to simulate temporal and spatial variations of atmospheric compositions (e.g., ozone and aerosol compositions). The CMAQ, treated as a column chemistry model, updates concentrations of chemical species (e.g., ozone and aerosol compositions) at each integration time step. The transport terms (e.g., *advection* and diffusion) of all chemical species are handled by the UFS Weather Model as *tracers*.

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

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

CDEPS
The Community Data Models for Earth Predictive Systems repository (CDEPS) contains a set of *NUOPC*-compliant data components and *ESMF*-based "stream" code that selectively removes feedback in coupled model systems. In essence, CDEPS handles the static Data Atmosphere (*DATM*) integration with dynamic coupled model components (e.g., *MOM6*). The CDEPS data models perform the basic function of reading external data files, modifying those data, and then sending the data back to the *CMEPS* mediator. The fields sent to the *mediator* are the same as those that would be sent by an active component. This takes advantage of the fact that the mediator and other CMEPS-compliant model components have no fundamental knowledge of whether another component is fully active or just a data component. More information about DATM is available in the CDEPS Documentation.

CESM
The Community Earth System Model (CESM) is a fully-coupled global climate model developed at the National Center for Atmospheric Research (*NCAR*) in collaboration with colleagues in the research community.

chgres_cube
The preprocessing software used to create initial and boundary condition files to "coldstart" the forecast model. It is part of *UFS_UTILS*.

CICE
CICE6
Sea Ice Model
CICE is a computationally efficient model for simulating the growth, melting, and movement of polar sea ice. It was designed as one component of a coupled atmosphere-ocean-land-ice global climate model. CICE has several interacting components, including a model of ice dynamics, a transport model that describes *advection* of different state variables; and a vertical physics package called "Icepack".

CMAQ
The Community Multiscale Air Quality Model (CMAQ, pronounced "cee-mak") is a numerical air quality model that predicts the concentration of airborne gases and particles and the deposition of these pollutants back to Earth's surface. The purpose of CMAQ is to provide fast, technically sound estimates of ozone, particulates, toxics, and acid deposition. CMAQ is an active open-source development project of the U.S. Environmental Protection Agency (EPA). Code is publicly availably at https://github.com/USEPA/CMAQ.

CMEPS
The Community Mediator for Earth Prediction Systems (CMEPS) is a *NUOPC*-compliant *mediator* used for coupling Earth system model components. It is currently being used in NCAR's Community Earth System Model (*CESM*) and NOAA's subseasonal-to-seasonal (S2S) coupled system. More information is available in the CMEPS Documentation.

cron
cron job
crontab
cron table
Cron is a job scheduler accessed through the command-line on UNIX-like operating systems. It is useful for automating tasks such as regression testing. Cron periodically checks a cron table (aka crontab) to see if any tasks are are ready to execute. If so, it runs them.

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

DATM
DATM is the *Data Atmosphere* component of *CDEPS*. It uses static atmospheric forcing files (derived from observations or previous atmospheric model runs) instead of output from an active atmospheric model. This reduces the complexity and computational cost associated with coupling to an active atmospheric model. The *Data Atmosphere* component is particularly useful when employing computationally intensive Data Assimilation (DA) techniques to update ocean and/or sea ice fields in a coupled model. In general, use of DATM in place of *ATM* can be appropriate when users are running a coupled model and only want certain components of the model to be active. More information about DATM is available in the CDEPS Documentation.

DOCN

DOCN is the *Data Ocean* component of *CDEPS*. It uses static ocean forcing files (derived from observations or previous ocean model runs) instead of output from an active ocean model. This reduces the complexity and computational cost associated with coupling to an active ocean model. The *Data Ocean* component is particularly useful when employing computationally intensive Data Assimilation (DA) techniques to update atmospheric fields in a coupled model. In general, use of DOCN in place of *MOM6* or *HYCOM* can be appropriate when users are running a coupled model and only want certain components of the model to be active. More information about DOCN is available in the CDEPS Documentation.

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

EMC
The Environmental Modeling Center is one of *NCEP*'s nine centers and leads the *National Weather Service*'s modeling efforts.

ESMF
Earth System Modeling Framework. The ESMF defines itself as "a suite of software tools for developing high-performance, multi-component Earth science modeling applications." It is a community-developed software infrastructure for building and coupling models.

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.

FV3
FV3 dycore
FV3 dynamical core
The Finite-Volume Cubed-Sphere *dynamical core* (dycore). Developed at NOAA's Geophysical Fluid Dynamics Laboratory (GFDL), it is a scalable and flexible dycore capable of both hydrostatic and non-hydrostatic atmospheric simulations. It is the dycore used in the UFS Weather Model.

GOCART
NASA's Goddard Chemistry Aerosol Radiation and Transport (GOCART) model simulates the distribution of major tropospheric aerosol types, including sulfate, dust, organic carbon (OC), black carbon (BC), and sea salt aerosols. The UFS Weather Model integrates a prognostic aerosol component using GOCART. The code is publicly available on GitHub at https://github.com/GEOS-ESM/GOCART.

HPC-Stack
The HPC-Stack is a repository that provides a unified, shell script-based build system for building the software stack required for numerical weather prediction (NWP) tools such as the Unified Forecast System (UFS) and the Joint Effort for Data assimilation Integration (JEDI) framework.

HAFS
The Hurricane Analysis and Forecast System (HAFS) is a *UFS* application for hurricane forecasting. It is an *FV3*-based multi-scale model and data assimilation (DA) system capable of providing analyses and forecasts of the inner core structure of tropical cyclones (TC) — including hurricanes and typhoons — out to 7 days. This is key to improving size and intensity predictions. HAFS also provides analyses and forecasts of the large-scale environment that is known to influence a TC's motion. HAFS development targets an operational analysis and forecast system for hurricane forecasters with reliable, robust and skillful guidance on TC track and intensity (including rapid intensification), storm size, genesis, storm surge, rainfall, and tornadoes associated with TCs. Currently, HAFS is under active development with collaborative efforts among NCEP/EMC, AOML/HRD, GFDL, ESRL/GSD, ESRL/NESII, OFCM/AOC, and NCAR/DTC.

HYCOM
The HYbrid Coordinate Ocean Model (HYCOM) was developed to address known shortcomings in the vertical coordinate scheme of the Miami Isopycnic-Coordinate Ocean Model (MICOM). HYCOM is a primitive equation, general circulation model with vertical coordinates that remain isopycnic in the open, stratified ocean. However,

the isopycnal vertical coordinates smoothly transition to z-coordinates in the weakly stratified upper-ocean mixed layer, to terrain-following sigma coordinates in shallow water regions, and back to z-level coordinates in very shallow water. The latter transition prevents layers from becoming too thin where the water is very shallow. See the HYCOM User's Guide for more information on using the model. The HYCOM model code is publicly available on GitHub.

LND
land component
The Noah Multi-Physics (Noah-MP) land surface model (LSM) is an open-source, community-developed LSM that has been incorporated into the UFS Weather Model (WM). It is the UFS WM's land component.

Mediator
A mediator, sometimes called a coupler, is a software component that includes code for representing component interactions. Typical operations include merging data fields, ensuring consistent treatment of coastlines, computing fluxes, and temporal averaging.

MOM
MOM6
Modular Ocean Model
MOM6 is the latest generation of the Modular Ocean Model. It is numerical model code for simulating the ocean general circulation. MOM6 was originally developed by the Geophysical Fluid Dynamics Laboratory. Currently, MOM6 code and an extensive suite of test cases are available under an open-development software framework. Although there are many public forks of MOM6, the NOAA EMC fork is used in the UFS Weather Model.

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

NCAR
The National Center for Atmospheric Research.

NCEP
National Centers for Environmental Prediction (NCEP) is a branch of the *National Weather Service* and consists of nine centers, including the *Environmental Modeling Center*. More information can be found at https://www.ncep.noaa.gov.

NCEPLIBS
The software libraries created and maintained by *NCEP* that are required for running *chgres_cube*, the UFS Weather Model, and the *UPP*. They are included in spack-stack and HPC-Stack.

NCEPLIBS-external
A collection of third-party libraries required to build *NCEPLIBS*, *chgres_cube*, the UFS Weather Model, and the *UPP*. They are included in *spack-stack* and *HPC-Stack*.

NEMS
The NOAA Environmental Modeling System is a common modeling framework whose purpose is to streamline components of operational modeling suites at *NCEP*.

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

NG-GODAS
Next Generation-Global Ocean Data Assimilation System. NG-GODAS is a UFS Weather Model configuration that couples ocean (*MOM6*), sea ice (*CICE6*), and Data Assimilation (DA) capabilities with the *DATM* component of *CDEPS*.

NUOPC
National Unified Operational Prediction Capability

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.

NUOPC Layer

The *NUOPC* Layer "defines conventions and a set of generic components for building coupled models using the Earth System Modeling Framework (*ESMF*)." NUOPC applications are built on four generic components: driver, model, *mediator*, and connector. For more information, visit the NUOPC website.

NWP
Numerical Weather Prediction

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

NWS

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

Parameterizations

Simplified functions that approximate the effects of small-scale processes (e.g., microphysics, gravity wave drag) that cannot be explicitly resolved by a model grid's representation of the earth. Common categories of parameterizations include radiation, surface layer, planetary boundary layer and vertical mixing, deep and shallow cumulus, and microphysics. Parameterizations can be grouped together into physics suites (such as the *CCPP* physics suites), which are sets of parameterizations known to work well together.

Post-processor

Software that enhances the value of the raw forecasts produced by the modeling application to make them more useful. At *NCEP*, the *UPP* (Unified Post Processor) software is used to convert data from spectral to gridded format, de-stagger grids, interpolate data vertically (e.g., to isobaric levels) and horizontally (to various predefined grids), and to compute derived variables. Some types of post-processors, such as statistical post-processors, use historical information of previous runs and observations to de-bias and calibrate its output.

RT
Regression test

Tests to validate that software still performs as expected after a change. In general, RTs ensure that the code should produce the same results and performance, within predefined measures of variance. When a code change is designed to change results or performance, then a new baseline is created. From these baselines, regression tests determine whether a change has occurred.

SRW
SRW App
Short-Range Weather Application

The Short-Range Weather Application is a UFS Application that targets predictions of atmospheric behavior on a limited spatial domain and on time scales from minutes out to about two days. It packages a prognostic atmospheric model (the UFS Weather Model), pre- and post-processing tools, and a community workflow.

spack-stack

The spack-stack is a collaborative effort between the NOAA Environmental Modeling Center (EMC), the UCAR Joint Center for Satellite Data Assimilation (JCSDA), and the Earth Prediction Innovation Center (EPIC). *spack-stack* is a repository that provides a Spack-based method for building the software stack required for numerical weather prediction (NWP) tools such as the Unified Forecast System (UFS) and the Joint Effort for Data assimilation Integration (JEDI) framework. *spack-stack* uses the Spack package manager along with custom Spack configuration files and Python scripts to simplify installation of the libraries required to run various applications.

The *spack-stack* can be installed on a range of platforms and comes pre-configured for many systems. Users can install the necessary packages for a particular application and later add the missing packages for another application without having to rebuild the entire stack.

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

tracer
According to the American Meteorological Society (AMS) definition, a tracer is "Any substance in the atmosphere that can be used to track the history [i.e., movement] of an air mass." Tracers are carried around by the motion of the atmosphere (i.e., by *advection*). These substances are usually gases (e.g., water vapor, CO2), but they can also be non-gaseous (e.g., rain drops in microphysics parameterizations). In weather models, temperature (or potential temperature), absolute humidity, and radioactivity are also usually treated as tracers. According to AMS, "The main requirement for a tracer is that its lifetime be substantially longer than the transport process under study."

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

UFS_UTILS
The UFS Utilities repository (UFS_UTILS) contains a collection of pre-processing programs for use with the UFS Weather Model and UFS applications. These programs set up the model grid and create coldstart initial conditions. The code is publicly available on the UFS_UTILS Github repository.

UPP
Unified Post Processor
The Unified Post Processor is the *post-processor* software developed at *NCEP*. It is used operationally to convert the raw output from a variety of *NCEP*'s *NWP* models, including the *FV3 dycore*, to a more useful form.

WW3
WWIII
WaveWatch III
WAVEWATCH III (WW3) is a community wave modeling framework that includes the latest scientific advancements in the field of wind-wave modeling and dynamics. The core of the framework consists of the WAVEWATCH III third-generation wave model (WAVE-height, WATer depth and Current Hindcasting), developed at NOAA/*NCEP*. WAVEWATCH III differs from its predecessors in many important points such as governing equations, model structure, numerical methods and physical parameterizations. The model code is publicly available on GitHub at https://github.com/NOAA-EMC/WW3.

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

WM
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. The UFS Weather Model repository is publicly available on GitHub.

BIBLIOGRAPHY

[BDT+20] L. Bengtsson, J. Dias, S. Tulich, M. Gehne, and J. Bao. A stochastic parameterization of organized tropical convection using cellular automata for global forecasts in noaa's unified forecast system. *Journal of Advances in Modeling Earth Systems*, 2020. URL: https://agupubs.onlinelibrary.wiley.com/doi/10.1029/2020MS002260.

INDEX