
UFS Weather Model Users Guide

May 29, 2022

CONTENTS

1	Introduction	1
2	Code Overview	3
2.1	UFS Weather Model Hierarchical Repository Structure	3
2.2	Directory Structure	4
3	Building and Running the UFS Weather Model	7
3.1	Prerequisite Libraries	7
3.2	Downloading the Weather Model Code	8
3.3	Building the Weather Model	8
3.3.1	Loading the required modules	8
3.3.2	Setting the CMAKE_FLAGS and CCPP_SUITES environment variables	8
3.3.3	Building the model	9
3.4	Running the model	10
3.4.1	Using the regression test script	10
3.4.2	Using the operational requirement test script	11
4	Inputs and Outputs	13
4.1	Input files	14
4.1.1	UFSAtm	14
4.1.2	MOM6	16
4.1.3	HYCOM	17
4.1.4	CICE6	18
4.1.5	WW3	19
4.1.6	CDEPS	21
4.1.7	GOCART	22
4.2	Model configuration files	22
4.2.1	<i>diag_table</i> file	23
4.2.2	<i>field_table</i> file	27
4.2.3	<i>model_configure</i> file	28
4.2.4	<i>nems.configure</i> file	29
4.2.5	<i>The SDF (Suite Definition File) file</i>	36
4.2.6	<i>datm.streams</i>	36
4.2.7	<i>datm_in</i>	37
4.2.8	<i>blkdat.input</i>	38
4.2.9	Namelist file <i>input.nml</i>	38
4.2.10	<i>fms_io_nml</i>	38
4.2.11	<i>namsfc</i>	40
4.2.12	<i>atmos_model_nml</i>	41
4.3	Output files	41

4.3.1	FV3Atm	41
4.3.2	MOM6	43
4.3.3	HYCOM	44
4.3.4	CICE6	44
4.3.5	WW3	45
4.3.6	CMEPS	45
4.4	Additional Information about the FMS Diagnostic Manager	45
4.4.1	Diagnostic Manager namelist	45
4.5	Additional Information about the Write Component	46
5	Automated Testing	47
5.1	CI/CD	47
5.2	Auto RT	48
5.2.1	AutoRT Workflow	48
6	FAQ	49
6.1	How do I build and run a single test of the UFS Weather Model?	49
6.2	How do I change the length of the model run?	50
6.3	How do I select the file format for the model output (netCDF or NEMSIO)?	50
6.4	How do I set the output history interval?	50
6.5	How do I set the total number of tasks for my job?	51
7	Acronyms	53
8	Glossary	55
	Index	57

INTRODUCTION

The Unified Forecast System (*UFS Weather Model* (WM) is a prognostic model that can be used for short- and medium-range research and operational forecasts, as exemplified by its use in the operational Global Forecast System (GFS) of the National Oceanic and Atmospheric Administration (NOAA). The UFS WM v2.0 is the latest public release of this software and represents a snapshot of a continuously evolving system undergoing open development. More information about the UFS can be found in its portal at <https://ufscommunity.org/>.

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

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

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

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

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

The UFS WM v2 code is portable and can be used with Linux and Mac operating systems with Intel and GNU compilers. It has been tested in a variety of platforms widely used by atmospheric scientists, such as the NOAA research Hera system, the National Center for Atmospheric Research (NCAR) Cheyenne system, the National Science Foundation Stampede system, and Mac laptops.

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

Those wishing to contribute development to the UFS WM should become familiar with the procedures for running the model as a standalone component and for executing the regression tests described in the UFS WM GitHub [wiki](#) to make sure no inadvertent changes to the results have been introduced during the development process.

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

This WM User's Guide is organized as follows:

- [Chapter 2](#) (Code Overview) provides a description of the various code repositories from which source code is pulled and an overview of the directory structure.
- [Chapter 3](#) (Building and Running the WM) explains how to use the WM without an application.
- [Chapter 4](#) (Inputs and Outputs) lists the model inputs and outputs and has a description of the key files.
- [Chapter 6](#) (FAQ) lists frequently asked questions and answers.

Finally, [Chapters 7](#) and [8](#) contain a list of acronyms and a glossary, respectively.

CODE OVERVIEW

2.1 UFS Weather Model Hierarchical Repository Structure

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

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

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

Todo: GOCART information in progress

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

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

ufs-weather-model/	
├── FV3	https://github.com/NOAA-EMC/fv3atm
│ ├── atmos_cubed_sphere	https://github.com/NOAA-GFDL/GFDL_atmos_
│ └── cubed_sphere	
│ ├── ccpp	
│ ├── framework	https://github.com/NCAR/ccpp-framework
│ └── physics	https://github.com/NCAR/ccpp-physics
├── NEMS	https://github.com/NOAA-EMC/NEMS
├── stochastic_physics	https://github.com/noaa-psd/stochastic_
└── physics	
├── MOM6-interface	
│ └── MOM6	https://github.com/NOAA-EMC/MOM6
├── HYCOM-interface	
│ └── HYCOM	https://github.com/NOAA-EMC/HYCOM-src
├── CICE-interface	
│ └── CICE	https://github.com/NOAA-EMC/CICE
├── WW3	https://github.com/NOAA-EMC/WW3
├── CMEPS-interface	
│ └── CMEPS	https://github.com/NOAA-EMC/CMEPS
├── CDEPS-interface	
│ └── CDEPS	https://github.com/NOAA-EMC/CDEPS

2.2 Directory Structure

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

ufs-weather-model/	
├── cmake	----- cmake configuration files
├── doc	----- User Guide files
├── FV3	----- UFSAtm atmosphere model
│ ├── atmos_cubed_sphere	----- FV3 dynamic core
│ │ ├── docs	
│ │ ├── driver	
│ │ ├── model	
│ │ └── tools	
│ ├── ccpp	----- Common Community Physics Package
│ │ ├── config	
│ │ ├── driver	
│ │ ├── framework	----- CCPP framework
│ │ ├── physics	----- CCPP compliant physics schemes
│ │ └── suites	----- CCPP physics suite definition files (SDFs)
│ ├── cpl	----- Coupling field data structures
│ ├── io	----- UFSAtm write grid comp code
│ └── stochastic_physics	----- Wrapper for stochastic physics
├── NEMS	----- NOAA Earth Modeling System framework
├── stochastic_physics	----- stochastic physics pattern generator
├── MOM6-interface	
│ └── MOM6	
│ └── src	----- MOM6 ocean model

(continues on next page)

(continued from previous page)

└─ config_source/drivers/nuopc_cap	-----	NUOPC MOM6 cap
─ HYCOM-interface		
└─ HYCOM	-----	HYCOM ocean model
└─ NUOPC	-----	NUOPC HYCOM cap
─ CICE-interface		
─ CICE-interface		
└─ CICE	-----	CICE6 sea ice model
└─ icepack	-----	Sea ice column physics
└─ cicecore/drivers/nuopc/cmeps	-----	NUOPC CICE6 cap
─ WW3		
└─ model	-----	WW3 model
└─ esmf	-----	NUOPC WW3 cap
─ CDEPS-interface		
└─ CDEPS		
└─ datm	-----	CDEPS DATM
└─ docn	-----	CDEPS DOCN
─ modulefiles	-----	system module files for supported HPC systems
─ tests	-----	regression test infrastructure
└─ parm		
└─ tests		
└─ fv3_conf		

Todo: GOCART information in progress

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

BUILDING AND RUNNING THE UFS WEATHER MODEL

3.1 Prerequisite Libraries

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

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

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

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

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

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

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

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

3.2 Downloading the Weather Model Code

To clone the develop branch of the ufs-weather-model repository, execute the following commands:

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

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

3.3 Building the Weather Model

3.3.1 Loading the required modules

Modulefiles for [pre-configured 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 3.1](#).

Table 3.1: CMake environment variables required to configure the build for the Weather Model

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

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

3.3.2 Setting the CMAKE_FLAGS and CCPP_SUITES environment variables

You need to use the `CMAKE_FLAGS` environment variable to specify which application to build. In order to have one or more CCPP physics suites available at runtime, you also need to select those suites at build time by setting the `CCPP_SUITES` environment variable. Multiple suites can be set. Following examples are for the bash shell.

For the ufs-weather-model ATM app (standalone ATM):

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

For the ufs-weather-model ATM app (standalone ATM) in 32 bit:

```
export CMAKE_FLAGS="-DAPP=ATM -D32BIT=ON -DCCPP_SUITES=FV3_GFS_v16"
```

For the ufs-weather-model ATMW app (standalone ATM with wave):

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

For the ufs-weather-model S2S app (atm/ice/ocean):

```
export CMAKE_FLAGS="-DAPP=S2S -DCCPP_SUITES=FV3_GFS_2017_coupled,FV3_GFS_2017_satmedmf_
↪coupled,FV3_GFS_v15p2_coupled,FV3_GFS_v16_coupled,FV3_GFS_v16_couplednsst"
```

For the ufs-weather-model S2S app (atm/ice/ocean) with debugging flags turned on, with verbose build messages:

```
export CMAKE_FLAGS="-DAPP=S2S -DDEBUG=ON -DCCPP_SUITES=FV3_GFS_2017_coupled,FV3_GFS_2017_
↪satmedmf_coupled,FV3_GFS_v15p2_coupled,FV3_GFS_v16_coupled,FV3_GFS_v16_couplednsst"
export BUILD_VERBOSE=1
```

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

```
export CMAKE_FLAGS="-DAPP=S2SW -DCCPP_SUITES=FV3_GFS_2017_coupled,FV3_GFS_v15p2_coupled,
↪FV3_GFS_v16_coupled,FV3_GFS_v16_coupled_noahmp"
```

For the ufs-weather-model HAFS app (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,FV3_HAFS_v0_hwrf_thompson,FV3_HAFS_v0_hwrf"
```

For the ufs-weather-model HAFSW app (atm/ocean/wave) in 32 bit:

```
export CMAKE_FLAGS="-DAPP=HAFSW -D32BIT=ON -DCCPP_SUITES=FV3_HAFS_v0_gfdlmp_tedmf_nonsst,
↪FV3_HAFS_v0_gfdlmp_tedmf,FV3_HAFS_v0_hwrf_thompson,FV3_HAFS_v0_hwrf"
```

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

```
export CMAKE_FLAGS="-DAPP=HAFS-ALL -D32BIT=ON -DCCPP_SUITES=FV3_HAFS_v0_gfdlmp_tedmf_
↪nonsst,FV3_HAFS_v0_gfdlmp_tedmf,FV3_HAFS_v0_hwrf_thompson,FV3_HAFS_v0_hwrf"
```

3.3.3 Building the model

The UFS Weather Model uses the CMake build system. There is a build script called `build.sh` in the top-level directory of the WM repository that configures the build environment and runs the `make` command. This script also checks that all necessary environment variables have been set.

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

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

The WM can be built by running the following command from the *ufs-weather-model* directory:

```
./build.sh
```

Once `build.sh` is finished, you should see the executable, named `ufs_model`, in the *ufs-weather-model/build/* directory. If it is desired to build in a different directory, specify the `BUILD_DIR` environment variable: e.g. `export BUILD_DIR=test_cp1d` will build in the *ufs-weather-model/test_cp1d* directory instead.

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

3.4 Running the model

3.4.1 Using the regression test script

The regression test script `rt.sh` in the `tests/` directory can be used to run a number of preconfigured test cases. It is the top-level script that calls lower-level scripts to build, set up environments and run tests. On [Tier-1 platforms](#), it can be as simple as editing the `rt.conf` file and subsequently executing

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

Following discussions are general, but the user may not be able to successfully execute the script as is unless s/he is on one of the Tier-1 platforms.

Each line in the PSV (Pipe-separated values) file `rt.conf` is used to either build or run. The `COMPILE` line specifies the application to build (e.g. `APP=S2S`), CCpp suite to use (e.g. `SUITES=FV3_GFS_2017_coupled`), and additional build options (e.g. `DEBUG=Y`) as necessary. The `RUN` line specifies the name of a test to run. 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 order of lines in `rt.conf` matters since `rt.sh` processes them sequentially; a `RUN` line should be preceded by a `COMPILE` line that builds the model used in the test. The following example `rt.conf` file builds the standalone ATM model in 32 bit and then runs the `control` test:

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

The third column of `rt.conf` relates to the platform; if left blank, the test runs on all Tier-1 platforms. The fourth column deals with baseline creation (more on this later) and `fv3` means the test will be included during baseline creation. The `rt.conf` file includes a large number of tests. If the user wants to run only a specific test, s/he can either comment out (using the `#` prefix) the tests to be skipped, or create a new file, e.g. `my_rt.conf`, then execute `./rt.sh -l my_rt.conf`.

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 specific to a given platform and reports the outcome (hence, the ‘regression’ test): ‘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 found in the `tests/log_<machine>.<compiler>/` directory. In particular, the user may find useful the run directory path provided as the value of `RUNDIR` variable in the `run_<test-name>` file. `$RUNDIR` is a self-contained (i.e. sandboxed) directory with the executable file, initial conditions, model configuration files, environment setup scripts and a batch job submission script. The user can run the test by `cd`’ing into `$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`.

Found 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 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 depend on the test being run, and are set in two stages: default values are specified in `tests/default_vars.sh` and the default values are overridden if necessary by those specified in a test file `tests/tests/<test-name>`. For example, the variable `DT_ATMOS`, which is substituted into the template file `model_configure.IN` to generate `model_configure`, is initially assigned 1800 in the function `export_fv3` of the script `default_vars.sh`, but the test file `tests/tests/control` overrides by reassigning 720 to the variable.

Also found inside the \$RUNDIR directory are the files `fv3_run` and `job_card`, which are generated from the template files in the `tests/fv3_conf/` directory. The latter is a platform-specific batch job submission script, while the former prepares the initial conditions by copying relevant data from the input data directory of a given platform to the \$RUNDIR directory. Table 3.2 summarizes the subdirectories discussed above.

Table 3.2: Regression test subdirectories

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

There are a number of command line options available to the `rt.sh` script. The user can execute `./rt.sh` to see information on these options. A couple of them are discussed here. When running a large number (10's or 100's) of tests, the `-e` option to use the ecFlow workflow manager can significantly decrease the testing time by queuing the jobs according to dependencies and running 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 baseline. New baselines are needed when code changes lead to result changes, and therefore deviate from existing baselines on a bit-for-bit basis.

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 the test. Default variables and their values are defined in the `export_fv3` function of the `default_vars.sh` script for ATM application, `export_cpl` function for S2S application and `export_datm` function for GODAS application. Also, the names of template files for model configuration and initial conditions can be identified via variables `INPUT_NML`, `NEMS_CONFIGURE` and `FV3_RUN`; for example, by trying `grep -n INPUT_NML *` inside the `tests/` and `tests/tests/` directories.

3.4.2 Using the operational requirement test script

The operational requirement test script `opnReqTest` in the `tests/` directory can also be used to run tests. Given the name of a test, `opnReqTest` carries out a suite of test cases. Each test case addresses an aspect of the requirements new implementations should satisfy, which are shown in Table 3.3. For the following discussions on `opnReqTest`, the user should note the distinction between ‘test name’ and ‘test case’: examples of test name are `control`, `cpld_control` and `regional_control` which are all found in the `/tests/tests/` directory, whereas test case refers to any one of `thr`, `mpi`, `dcp`, `rst`, `bit` and `dbg`.

Table 3.3: Operational requirements

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

The operational requirement test uses the same testing framework used by the regression test, and therefore it is recommended that the user first read Section 3.4.1. All the files in the subdirectories shown in Table 3.2 are relevant to the operational requirement test except 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 3.3](#) for control test. At the end of the run, a log file `OpnReqTests_<machine>.<compiler>.log` is generated in `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`; frequently used options are `-e` to use the `ecFlow` workflow manager, and `-k` to keep the `$RUNDIR`. In the following, comparisons are made between the regression and operational requirement tests on how they handle different reproducibility tests.

As discussed in [Section 3.4.1](#), 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 3.3](#) as different tests. Therefore, each test case requires a test file in the `tests/tests/` subdirectory; examples are `control_2threads`, `control_decomp`, `control_restart` and `control_debug`, which are just variations of control test to check various reproducibilities. There are two potential issues with this approach. First, if several different variations of a given test were to be created and included in the `rt.conf` file, there are 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 3.3](#) 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, and the related scripts can be found in the `tests/opnReqTests/` directory.

INPUTS AND OUTPUTS

The ufs-weather-model can be configured as one of several applications, from a single component atmosphere model to a fully coupled model with multiple earth system components (atmosphere, ocean, sea-ice and mediator). Currently the supported configurations are:

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

App name	Description
ATM	Standalone UFSAtm
ATMW	UFSAtm coupled to WW3
AT- MAERO	UFSAtm coupled to GOCART
S2S	Coupled UFSATM-MOM6-CICE6-CMEPS
S2SA	Coupled UFSATM-MOM6-CICE6-GOCART-CMEPS
S2SW	Coupled UFSATM-MOM6-CICE6-WW3-CMEPS
S2SWA	Coupled UFSATM-MOM6-CICE6-WW3-GOCART-CMEPS
NG- GODAS	Coupled CDEPS-DATM-MOM6-CICE6-CMEPS
HAFS	Coupled UFSATM-HYCOM-CMEPS
HAFSW	Coupled UFSATM-HYCOM-WW3-CMEPS
HAFS- ALL	Coupled CDEPS-UFSATM-HYCOM-WW3-CMEPS

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

4.1 Input files

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

4.1.1 UFSAtm

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.

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 CO ₂ 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_h2oprds.f77	Coefficients for the parameterization of photochemical production and loss of water (H ₂ O)
global_maxice.2x2.grb	Maximum ice extent, permanent/extreme features
global_mxsnoalb.uariz.t126.384.190.rg.grb	Climatological maximum snow albedo
global_o3prds.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 are 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.nc	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	horizontal grid information	1.00, 0.50, 0.25	
ocean_mosaic.nc	specify horizontal 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	lands/sea mask	1.00, 0.50, 0.25	
hy-com1_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.nc	specify grid points where topography are manually modified to adjust throughflow strength for narrow channels	1.00	
MOM_channels_global.nc	specifies restricted channel widths	0.50, 0.25	
MOM_channel_SPEAR.nc	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
<i>blkdat.input</i>	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 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 (sym-link)	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_YYYYMMDDHHmm	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_001	Conservative remapping gsh_15m to gnh_10m	na	na	na
rmp_src_to_dst_conserv_003_001	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.

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	✓

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	
GHR SST_mesh.nc	ESMF mesh file for GHR SST data source	
hycom.YYYYMM_surf_nolev.nc	MOM6 forcing file for year YYYY and month MM	✓
ghrsst.YYYYMMDD.nc	GHR SST forcing file for year YYYY, month MM and day DD	✓

4.1.7 GOCART

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

Todo: GOCART information in progress

The static input files for global configurations are listed and described in [Table 4.18](#).

Table 4.18: *Fix files containing climatological information*

Filename	Description

Grid description and initial condition files

Todo: GOCART information in progress

The input files containing grid information and the initial conditions for global configurations are listed and described in [Table 4.19](#).

Table 4.19: *Input files containing grid information and initial conditions for global configurations*

Filename	Description	Date-dependent

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*
- *nems.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 4.20](#).

Table 4.20: 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 )
```

(continues on next page)

(continued from previous page)

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 4.21](#).

Table 4.21: *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”.
reduc- tion_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,
"dynamics", "grid_lont", "grid_lont", "grid_spec", "all", .false., "none", 2,
```

(continues on next page)

(continued from previous page)

```

"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 having a hash symbol (#) as the first character in the line.

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

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

In this case, methods applied to this tracer include setting the long name to “specific humidity”, the units to “kg/kg”. Finally a field named “profile_type” will be given a child field called “fixed”, and that field will be given a field called “surface_value” with a real value of 3.E-6. The “profile_type” options are listed in [Table 4.22](#). 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 4.22: *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 4.23 shows the following parameters that can be set in *model_configure* at run-time.

Table 4.23: *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)
output_1st_tstep_rst	output first time step history file after restart	logical	.false.
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
nfhout	history file output frequency	integer	3
nfhmax_hf	forecast length of high history file	integer	0 (0:no high frequency output)
nfhout_hf	high history file output frequency	integer	1
nsout	output frequency of number of time step	integer	-1 (negative: turn off the option, 1: output history file at every time step)
output_fh	history file output forecast hours or history file output frequency if the second element is -1	real	-1 (negative: turn off the option, otherwise overwritten nfhout/nfhout_fh; 6 -1: output every 6 hours; 6 9: output history files at fh=6 and 9. Note: output_fh can only take 1032 characters)

Table 4.24 shows the following parameters in *model_configure* that are not usually changed.

Table 4.24: *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 length	integer	0

4.2.4 nems.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 *nems.configure* is simple and does not need to be changed. A sample of the file contents is shown below:

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

For the fully coupled S2SW application, a sample *nems.configure* is shown below :

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

# MED #
MED_model:           cmeps
MED_petlist_bounds:  0 143
::

# ATM #
ATM_model:           fv3
ATM_petlist_bounds:  0 149
```

(continues on next page)

(continued from previous page)

```

ATM_attributes::
::

# OCN #
OCN_model:                mom6
OCN_petlist_bounds:       150 179
OCN_attributes::
    mesh_ocn = mesh.mx100.nc
::

# ICE #
ICE_model:                cice6
ICE_petlist_bounds:       180 191
ICE_attributes::
    mesh_ice = mesh.mx100.nc
::

# WAV #
WAV_model:                ww3
WAV_petlist_bounds:       192 395
WAV_attributes::
::

# CMEPS warm run sequence
runSeq::
@3600
    MED med_phases_prep_ocn_avg
    MED -> OCN :remapMethod=redist
    OCN -> WAV
    WAV -> OCN :srcMaskValues=1
    OCN
    @900
        MED med_phases_prep_atm
        MED med_phases_prep_ice
        MED -> ATM :remapMethod=redist
        MED -> ICE :remapMethod=redist
        WAV -> ATM :srcMaskValues=1
        ATM -> WAV
        ICE -> WAV
        ATM
        ICE
        WAV
        ATM -> MED :remapMethod=redist
        MED med_phases_post_atm
        ICE -> MED :remapMethod=redist
        MED med_phases_post_ice
        MED med_phases_prep_ocn_accum
    @
    OCN -> MED :remapMethod=redist
    MED med_phases_post_ocn
    MED med_phases_restart_write
@

```

(continues on next page)

(continued from previous page)

```

::
# CMEPS variables
::
MED_attributes::
    ATM_model = fv3
    ICE_model = cice6
    OCN_model = mom6
    history_n = 1
    history_option = nhours
    history_ymd = -999
    coupling_mode = nems_orig
::
ALLCOMP_attributes::
    ScalarFieldCount = 2
    ScalarFieldIdxGridNX = 1
    ScalarFieldIdxGridNY = 2
    ScalarFieldName = cpl_scalars
    start_type = startup
    restart_dir = RESTART/
    case_name = ufs.cpld
    restart_n = 24
    restart_option = nhours
    restart_ymd = -999
    debug_flag = 0
    use_coldstart = false
    use_mommesh = true
::

```

For the coupled NG_GODAS application, a sample *nems.configure* is shown below :

```

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

# MED #
MED_model:                                cmeps
MED_petlist_bounds:                       0 11
    Verbosity = 5
    debug_flag = 5
::

# ATM #
ATM_model:                                datm
ATM_petlist_bounds:                       0 11
ATM_attributes::
    Verbosity = 0
    DumpFields = false

```

(continues on next page)

(continued from previous page)

```

mesh_atm = DATM_INPUT/cfsr_mesh.nc
diro = "."
logfile = atm.log
stop_n = 24
stop_option = nhours
stop_ymd = -999
write_restart_at_endofrun = .true.
::

# OCN #
OCN_model:                mom6
OCN_petlist_bounds:       12 27
OCN_attributes::
  Verbosity = 0
  DumpFields = false
  ProfileMemory = false
  OverwriteSlice = true
  mesh_ocn = mesh.mx100.nc
::

# ICE #
ICE_model:                cice6
ICE_petlist_bounds:       28 39
ICE_attributes::
  Verbosity = 0
  DumpFields = false
  ProfileMemory = false
  OverwriteSlice = true
  mesh_ice = mesh.mx100.nc
  stop_n = 12
  stop_option = nhours
  stop_ymd = -999
::

# CMEPS concurrent warm run sequence

runSeq::
@3600
  MED med_phases_prep_ocn_avg
  MED -> OCN :remapMethod=redist
  OCN
  @900
    MED med_phases_prep_ice
    MED -> ICE :remapMethod=redist
    ATM
    ICE
    ATM -> MED :remapMethod=redist
    MED med_phases_post_atm
    ICE -> MED :remapMethod=redist
    MED med_phases_post_ice
    MED med_phases_aofluxes_run
    MED med_phases_prep_ocn_accum

```

(continues on next page)

(continued from previous page)

```

@
OCN -> MED :remapMethod=redist
MED med_phases_post_ocn
MED med_phases_restart_write
@
::

# CMEPS variables

DRIVER_attributes::
    mediator_read_restart = false
::
MED_attributes::
    ATM_model = datm
    ICE_model = cice6
    OCN_model = mom6
    history_n = 1
    history_option = nhours
    history_ymd = -999
    coupling_mode = nems_orig_data
::
ALLCOMP_attributes::
    ScalarFieldCount = 3
    ScalarFieldIdxGridNX = 1
    ScalarFieldIdxGridNY = 2
    ScalarFieldIdxNextSwCday = 3
    ScalarFieldName = cpl_scalars
    start_type = startup
    restart_dir = RESTART/
    case_name = DATM_CFSR
    restart_n = 12
    restart_option = nhours
    restart_ymd = -999
    debug_flag = 0
    use_coldstart = false
    use_mommesh = true
    coldair_outbreak_mod = .false.
    flds_wiso = .false.
    flux_convergence = 0.0
    flux_max_iteration = 2
    ocn_surface_flux_scheme = 0
    orb_eccen = 1.e36
    orb_iyear = 2000
    orb_iyear_align = 2000
    orb_mode = fixed_year
    orb_mvlp = 1.e36
    orb_obliq = 1.e36
::

```

For the coupled HAFS application, a sample *nems.configure* is shown below :

```
# EARTH #
```

(continues on next page)

(continued from previous page)

```

EARTH_component_list: ATM OCN MED

# MED #
MED_model:                cmeps
MED_petlist_bounds:       1340 1399
MED_attributes::
    coupling_mode = hafs
    system_type = ufs
    normalization = none
    merge_type = copy
    ATM_model = fv3
    OCN_model = hycom
    history_ymd = -999
    ScalarFieldCount = 0
    ScalarFieldIdxGridNX = 0
    ScalarFieldIdxGridNY = 0
    ScalarFieldName = cpl_scalars
::

# ATM #
ATM_model:                fv3
ATM_petlist_bounds:       0000 1339
ATM_attributes::
    Verbosity = 1
    Diagnostic = 0
::

# OCN #
OCN_model:                hycom
OCN_petlist_bounds:       1340 1399
OCN_attributes::
    Verbosity = 1
    Diagnostic = 0
    cdf_impexp_freq = 3
    cpl_hour = 0
    cpl_min = 0
    cpl_sec = 360
    base_dtg = 2020082512
    merge_import = .true.
    skip_first_import = .true.
    hycom_arche_output = .false.
    hyc_esmf_exp_output = .true.
    hyc_esmf_imp_output = .true.
    import_diagnostics = .false.
    import_setting = flexible
    hyc_impexp_file = nems.configure
    espc_show_impexp_minmax = .true.
    ocean_start_dtg = 43702.50000
    start_hour = 0
    start_min = 0
    start_sec = 0
    end_hour = 12

```

(continues on next page)

(continued from previous page)

```

end_min = 0
end_sec = 0
::

DRIVER_attributes::
  start_type = startup
::

ALLCOMP_attributes::
  mediator_read_restart = false
::

# CMEPS cold run sequence

runSeq::
@360
  ATM -> MED :remapMethod=redist
  MED med_phases_post_atm
  OCN -> MED :remapMethod=redist
  MED med_phases_post_ocn
  MED med_phases_prep_atm
  MED med_phases_prep_ocn_accum
  MED med_phases_prep_ocn_avg
  MED -> ATM :remapMethod=redist
  MED -> OCN :remapMethod=redist
  ATM
  OCN
@
::

# HYCOM field coupling configuration (location set by hyc_impexp_file)

ocn_export_fields::
  'sst'      'sea_surface_temperature'  'K'
  'mask'     'ocean_mask'               '1'
::

ocn_import_fields::
  'taux10'   'mean_zonal_moment_flx_atm' 'N_m-2'
  'tauy10'   'mean_merid_moment_flx_atm' 'N_m-2'
  'prcp'     'mean_prec_rate'            'kg_m-2_s-1'
  'swflxd'   'mean_net_sw_flx'           'W_m-2'
  'lwflxd'   'mean_net_lw_flx'           'W_m-2'
  'mslprs'   'inst_pres_height_surface'  'Pa'
  'sensflx'  'mean_sensi_heat_flx'       'W_m-2'
  'latflx'   'mean_laten_heat_flx'       'W_m-2'
::

```

For more HAFS, HAFSW, and HAFS-ALL configurations please see the following nems.configure templates.

- [HAFS ATM-OCN](#)
- [HAFS ATM-WAV](#)

- [HAFS ATM-OCN-WAV](#)
- [HAFS ATM-DOCN](#)

Todo: GOCART information in progress

4.2.5 The SDF (Suite Definition File) file

There are two SDFs currently supported for the UFS Medium Range Weather App configuration: *suite_FV3_GFS_v15p2.xml* and *suite_FV3_GFS_v16beta.xml*.

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

Detailed descriptions of the supported suites can be found with the [CCPP v5.0.0 Scientific Documentation](#).

4.2.6 *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 4.25: *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
```

(continues on next page)

(continued from previous page)

```

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 Faxe_swdn" "DLWRF Faxe_lwdn" "vbdsf_
→ave Faxe_swdn" "vddsfc_ave Faxe_swdn" "nbdsfc_ave Faxe_swdn" "nddsfc_ave Faxe_swdn"
→"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" "precip Faxe_rain" "fprecip Faxe_snow"

```

4.2.7 datm_in

Table 4.26: 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 configuration 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 following link provides an in depth description of the namelist settings:

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

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

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

The following link describes the stochastic physics namelist records:

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

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

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

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

4.2.10 *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 4.27](#).

Table 4.27: Description of the `&fms_io_nml` namelist section.

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

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
/
```

4.2.11 namsvc

The namelist section `&namsvc` 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 `&namsvc` to set the filenames are described in Table 4.28.

Table 4.28: List of common variables in the `*namsvc` 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'
fnmxlc	Climatological maximum ice extent	character*500	'global_maxice.2x2.grb'
fntsf	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'
fnveg	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:

```
&namsvc
  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'
  FNMVNC   = 'global_shdmin.0.144x0.144.grb'
  FNMVXC   = 'global_shdmax.0.144x0.144.grb'
  FNSLPC   = 'global_slope.1x1.grb'
  FNABSC   = 'global_mxsnoalb.uariz.t126.384.190.rg.grb'
```

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

4.2.12 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 4.29](#).

Table 4.29: *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: $((\text{np}x-1)*(\text{np}y-1)/(\text{layout}\backslash_x)*(\text{layout}\backslash_y))$. 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 <code>build.sh</code>
avg_max_length	Forecast interval (in seconds) determining when the maximum values of diagnostic fields in FV3 dynamics are computed.	real	3600.

A sample of this namelist is shown below:

```
&atmos_model_nml
  blocksize = 32
  chksum_debug = .false.
  dycore_only = .false.
  ccpp_suite = 'FV3_GFS_v16beta'
/
```

The namelist section relating to the FMS diagnostic manager `&diag_manager_nml` is described in [Section 4.4.1](#).

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.

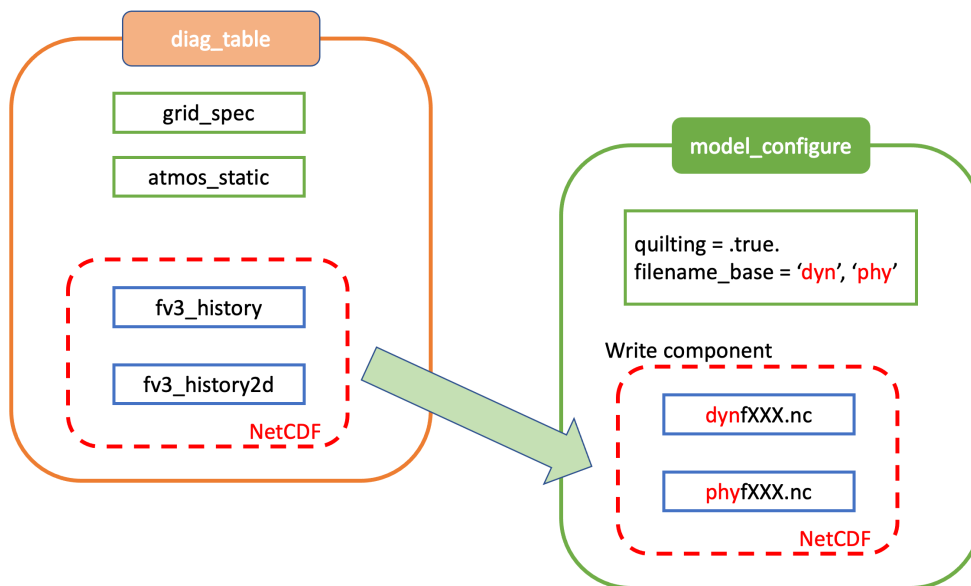


Fig. 4.1: Relationship between *diag_table*, *model_configure* and generated output files

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
"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 4.30

Table 4.30: *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 outputs 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 sell for profiling purposes. Post-processing utilities are available in the [HYCOM-tools](#) repository.

Table 4.31: *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, instantaneous 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 [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. More information about the FMS diagnostic manager can be found at: https://data1.gfdl.noaa.gov/summer-school/Lectures/July16/03_Seth1_DiagManager.pdf

4.4.1 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 4.32. See FMS/diag_manager/diag_manager.F90 for the complete list.

Table 4.32: *Namelist variables used to control the behavior of the diagnostic manager.*

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

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.

AUTOMATED TESTING

The UFS Weather Model repository on GitHub employs two types of automated testing. One is the CI/CD on cloud and the other is the AutoRT on NOAA R&D platforms. Both are application level tests, and utilize the regression testing framework discussed in [Section 3.4.1](#).

5.1 CI/CD

The following summarizes the CI/CD used in the UFS Weather Model:

- GitHub Actions (GHA), a GitHub-hosted continuous integration service is used.
- Build jobs are done on GHA-provided virtual machines.
- Test jobs are performed on Amazon cloud using a number of EC2 instances.
- Builds and tests are carried out using a Docker container.
- Docker container has prerequisite libraries installed via the hpc-stack.
- Input data needed to run tests are stored as a separate Docker container.

When a developer makes a pull request (PR) to the UFS Weather Model repository, and a code manager subsequently adds the *run-ci* label, the CI/CD workflow is triggered:

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 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 Amazon cloud by downloading the hpc-stack Docker container, the executable file from the build job, and the input-data Docker container.
6. When all tests are finished, 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 checking the up-to-dateness of the PR branch, and starting and stopping the EC2 instances. Other CI-related scripts are located in the `tests/ci/` directory. `ci.sh` is the main script that invokes Docker build and run. `Dockerfile` is used to build UFS Weather Model. Other shell and python scripts help with various tasks such as checking the up-to-dateness of the PR branch (`repo_check.sh`), checking the status of EC2 instances (`check_status.py`), and configuring the test cases to carry out in the CI/CD workflow (`setup.py` and `ci.test`).

5.2 Auto RT

The Automated Regression Testing (AutoRT) system:

- Automates the process of regression testing on NOAA HPC platforms.
- Written in python.
- Contains the following files:

Table 5.1: *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

5.2.1 AutoRT Workflow

- Cron-job on supported HPC systems runs start_rt_auto.sh bash script every 15 minutes.
 - This script verifies the HPC name, and sets the python paths. Runs rt_auto.py.
- rt_auto.py: Uses the Github API (Through pyGitHub)
 - Checks the pull requests to ufs-community/ufs-weather-model for labels specific to the HPC name. If no match to HPC name, exits. (i.e. hera-intel-RT or cheyenne-gnu-BL)
 - If the HPC name matches the label in ufs-weather-model pull request, the label provides the HPC with the compiler and job to run on the machine.
 - * For example the label gaea-intel-BL will be recognized by the HPC machine ‘Gaea’, set the RT_COMPILER variable to ‘intel’ and run the baseline creation script (bl.py).
 - Creates a Job class that contains all information from the machine that the job will need to run. That is sent into the jobs/rt[bl].py script.
- rt.py: Sets directories for storage, gets repo information, runs RT, post processes.

```
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.

```
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)
```

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

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

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

The steps are:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
qsub job_card
```

6.2 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.

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

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

6.4 How do I set the output history interval?

The interval at which output (history) files are written is controlled in two places, and depends on whether you are using the write component to generate your output files. [Table 6.1](#) 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 6.1: *Namelist variables used to control the output file frequency.*

Namelist variable	Location	Default Value	Description
fdiag	input.nml	0	Array with dimension <code>maxhr = 4096</code> 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 <code>fhmax</code> , or from <code>fhmaxhf</code> to <code>fhmax</code> if <code>fhmaxf > 0</code> .
fhouthf	input.nml	1	The high frequency output frequency during the forecast time from 0 to <code>fhmaxhf</code> 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

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

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

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

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

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 Prediction
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

GLOSSARY

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

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

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

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

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

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

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

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

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

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

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

INDEX

C

CCPP, [55](#)

CCPP-Framework, [55](#)

CCPP-Physics, [55](#)

F

FMS, [55](#)

N

NEMS, [55](#)

NUOPC, [55](#)

P

Parameterization or physics scheme, [55](#)

S

Suite, [55](#)

Suite Definition File (*SDF*), [55](#)

U

UFS, [55](#)

W

Weather Model, [55](#)