UFS Short-Range Weather App Users Guide

Release v1.0

Mar 03, 2021

CONTENTS

1 Introduction **1**
- 1.1 Pre-processor Utilities and Initial Conditions . . . 1
- 1.2 Forecast Model . . . 2
- 1.3 Post-processor . . . 3
- 1.4 Visualization Example . . . 3
- 1.5 Build System and Workflow . . . 3
- 1.6 User Support, Documentation, and Contributing Development . . . 4
- 1.7 Future Direction . . . 5
- 1.8 How to Use This Document . . . 5

2 Workflow Quick Start **7**
- 2.1 Download the UFS SRW Application Code . . . 7
- 2.2 Set up the Build Environment . . . 8
- 2.3 Build the Executables . . . 8
- 2.4 Generate the Workflow Experiment . . . 8
  - 2.4.1 Set up `config.sh` file . . . 9
  - 2.4.2 Set up the Python and other Environment Parameters . . . 10
  - 2.4.3 Run the `generate_FV3LAM_wflow.sh` script . . . 10
- 2.5 Run the Workflow Using Rocoto . . . 10
- 2.6 Plot the Output . . . 13

3 Code Repositories and Directory Structure **15**
- 3.1 Hierarchical Repository Structure . . . 15
- 3.2 Directory Structure . . . 16
  - 3.2.1 Regional Workflow Sub-Directories . . . 17
- 3.3 Experiment Directory Structure . . . 17

4 Short-Range Weather Application Overview **21**
- 4.1 Download from GitHub . . . 21
- 4.2 External Components . . . 23
- 4.3 Building the Executables for the Application . . . 23
- 4.4 Grid-specific Configuration . . . 24
- 4.5 Case-specific Configuration . . . 25
  - 4.5.1 Default configuration: `config_defaults.sh` . . . 25
  - 4.5.2 User-specific configuration: `config.sh` . . . 27

4.6 Python Environment for Workflow . . . 28
4.7 Generating a Regional Workflow Experiment . . . 29
4.7.1 Steps to a Generate a New Experiment . . . 29
4.7.2 Description of Workflow Tasks . . . 29
4.8 Launch of Workflow . . . 32
4.8.1 Launch with the `launch_FV3LAM_wflow.sh` script . . . 32
4.8.2 Manually launch by calling the `rocotorun` command . . . 34
4.8.3 Run the Workflow Using the Stand-alone Scripts . . . 35

5 Configuring the Workflow: `config.sh` and `config_defaults.sh` 37
5.1 Platform Environment . . . 37
5.2 Parameters for Running Without a Workflow Manager . . . 39
5.3 Cron-Associated Parameters . . . 39
5.4 Directory Parameters . . . 40
5.5 NCO Mode Parameters . . . 40
5.6 Pre-Processing File Separator Parameters . . . 41
5.7 File Name Parameters . . . 41
5.8 Foreast Parameters . . . 42
5.9 Initial and Lateral Boundary Condition Generation Parameters . . . 43
5.10 User-Staged External Model Directory and File Parameters . . . 43
5.11 CCPP Parameter . . . 44
5.12 Grid Generation Parameters . . . 44
5.13 Computational Forecast Parameters . . . 45
5.14 Write-Component (Quilting) Parameters . . . 45
5.15 Predefined Grid Parameters . . . 46
5.16 Pre-existing Directory Parameter . . . 47
5.17 Verbose Parameter . . . 47
5.18 Pre-Processing Parameters . . . 47
5.19 Surface Climatology Parameter . . . 48
5.20 Fixed File Parameters . . . 48
5.21 Workflow Task Parameters . . . 50
5.22 Customized Post Configuration Parameters . . . 52
5.23 Halo Blend Parameter . . . 52
5.24 FVCOM Parameter . . . 53
5.25 Compiler Parameter . . . 53

6 Limited Area Model (LAM) Grids: Predefined and User-Generated Options 55
6.1 Predefined Grids . . . 55
6.2 Creating User-Generated Grids . . . 59

7 Input and Output Files 63
7.1 Input Files . . . 63
7.1.1 Initial and Boundary Condition Files . . . 63
7.1.2 Pre-processing (UFS_UTILS) . . . 63
7.1.3 UFS Weather Model . . . 64
7.1.4 Unified Post Processor (UPP) . . . 64
7.1.5 Workflow . . . 64
7.2 Output Files . . . 67

7.2.1 Initial and boundary condition files . . . 67
7.2.2 Pre-processing (UFS_UTILS) . . . 67
7.2.3 UFS Weather Model . . . 67
7.2.4 Unified Post Processor (UPP) . . . 68
7.3 Downloading and Staging Input Data . . . 69
7.3.1 Static Files . . . 69
7.3.2 Initial Condition Formats and Source . . . 69
7.3.3 Initial and Lateral Boundary Condition Organization . . . 70
7.3.4 Default Initial and Lateral Boundary Conditions . . . 71
7.3.5 Running the App for Different Dates . . . 71
7.3.6 Staging Initial Conditions Manually . . . 71
7.3.7 Coexistence of Multiple Files for the Same Date . . . 72
7.3.8 Best Practices for Conserving Disk Space and Keeping Files Safe . . . 73

8 Configuring a New Platform **75**
8.1 Installing NCEPLIBS-external . . . 76
8.2 Installing NCEPLIBS . . . 77
8.3 Building the UFS Short-Range Weather Application (UFS SRW App) . . . 78
8.4 Setting Up Your Python Environment . . . 79
8.5 Running Without a Workflow Manager: Generic Linux and macOS Platforms . . . 80
8.6 Running on a New Platform with Rocoto Workflow Manager . . . 82
8.7 Software/Operating System Requirements . . . 83

9 Workflow End-to-End (WE2E) Tests **85**

10 Graphics Generation **89**
10.1 Plotting output from one experiment . . . 91
10.2 Plotting differences from two experiments . . . 91
10.3 Submitting plotting scripts through a batch system . . . 92

11 FAQ **95**
11.1 How do I turn On/Off the Cycle-Independent Workflow Tasks . . . 95
11.2 How do I define an experiment name? . . . 95
11.3 How do I change the Suite Definition File (SDF)? . . . 96
11.4 How do I restart a DEAD task? . . . 96
11.5 How do I change the grid? . . . 96

12 Additional Rocoto Information **97**
12.1 rocotorun . . . 97
12.2 rocotostat . . . 98
12.3 rocotocheck . . . 100
12.4 rocotorewind . . . 101
12.5 rocotoboot . . . 102

13 Glossary **103**

Bibliography **105**

Index **107**

CHAPTER

ONE

INTRODUCTION

The Unified Forecast System (*UFS*) is a community-based, coupled, comprehensive Earth modeling system. It is designed to be the source system for NOAA's operational numerical weather prediction applications while enabling research, development, and contribution opportunities for the broader weather enterprise. For more information about the UFS, visit the UFS Portal at https://ufscommunity.org/.

The UFS can be configured for multiple applications (see a complete list at https://ufscommunity.org/science/aboutapps/). The configuration described here is the UFS Short-Range Weather (SRW) Application, which targets predictions of atmospheric behavior on a limited spatial domain and on time scales from less than an hour out to several days. The SRW Application v1.0 release includes a prognostic atmospheric model, pre- and post-processing, and a community workflow for running the system end-to-end, which are documented within the User's Guide and supported through a community forum. Future work will include expanding the capabilities of the application to include data assimilation (DA) and a verification package (e.g. METplus) as part of the workflow. This documentation provides an overview of the release components, a description of the supported capabilities, a quick start guide for running the application, and information on where to find more information and obtain support.

The SRW App v1.0.0 citation is as follows and should be used when presenting results based on research conducted with the App.

UFS Development Team. (2021, March 4). Unified Forecast System (UFS) Short-Range Weather (SRW) Application (Version v1.0.0). Zenodo. https://doi.org/10.5281/zenodo.4534994

1.1 Pre-processor Utilities and Initial Conditions

The SRW Application includes a number of pre-processing utilities to initialize and prepare the model for integration. For the limited area model (LAM), it is necessary to first generate a regional grid `regional_esg_grid/make_hgrid` along with orography `orog` and surface climatology `sfc_climo_gen` files on that grid. There are additional utilities included to handle the correct number of halo `shave` points and topography filtering `filter_topo`. The pre-processing software `chgres_cube` is used to convert the raw external model data into initial and lateral boundary condition files in netCDF format, needed as input to the FV3-LAM. Additional information about the UFS pre-processor utilities can be found in the UFS_UTILS User's Guide.

The SRW Application can be initialized from a range of operational initial condition files. It is possible to initialize the model from GFS, NAM, RAP, and HRRR files in Gridded Binary v2 (GRIB2) format and GFS in NEMSIO format for past dates. Please note, for GFS data, dates prior to 1 January 2018 may work but are not guaranteed. Public archives of model data can be accessed through the National Centers for Environmental Information (NCEI) or through the NOAA Operational Model Archive and Distribution System (NOMADS). Raw external model data may be pre-staged on disk by the user.

1.2 Forecast Model

The prognostic atmospheric model in the UFS SRW Application is the Finite-Volume Cubed-Sphere (*FV3*) dynamical core configured with a Limited Area Model (LAM) capability [BAB+ed]. The dynamical core is the computational part of a model that solves the equations of fluid motion. A User's Guide for the UFS *Weather Model* is here.

Supported model resolutions in this release include a 3-, 13-, and 25-km predefined Contiguous U.S. (CONUS) domain, all with 64 vertical levels. Preliminary tools for users to define their own domain are also available in the release with full, formal support of these tools to be provided in future releases. The Extended Schmidt Gnomonic (ESG) grid is used with the FV3-LAM, which features relatively uniform grid cells across the entirety of the domain. Additional information about the FV3 dynamical core can be found here and on the NOAA Geophysical Fluid Dynamics Laboratory website.

Interoperable atmospheric physics, along with the Noah Multi-parameterization (Noah MP) Land Surface Model options, are supported through the Common Community Physics Package (*CCPP*; described here). Atmospheric physics are a set of numerical methods describing small-scale processes such as clouds, turbulence, radiation, and their interactions. There are two physics options supported for the release. The first is an experimental physics suite being tested for use in the future operational implementation of the Rapid Refresh Forecast System (RRFS) planned for 2023-2024, and the second is an updated version of the physics suite used in the operational Global Forecast System (GFS) v15. A scientific description of the CCPP parameterizations and suites can be found in the CCPP Scientific Documentation, and CCPP technical aspects are described in the CCPP Technical Documentation. The model namelist has many settings beyond the physics options that can optimize various aspects of the model for use with each of the supported suites.

The SRW App supports the use of both GRIB2 and *NEMSIO* input data. The UFS Weather Model ingests initial and lateral boundary condition files produced by *chgres_cube* and outputs files in NetCDF format on a specific projection (e.g., Lambert Conformal) in the horizontal and model levels in the vertical.

1.3 Post-processor

The SRW Application is distributed with the Unified Post Processor (*UPP*) included in the workflow as a way to convert the NetCDF output on the native model grid to GRIB2 format on standard isobaric vertical coordinates. UPP can also be used to compute a variety of useful diagnostic fields, as described in the UPP user's guide.

Output from UPP can be used with visualization, plotting, and verification packages, or for further downstream post-processing, e.g. statistical post-processing techniques.

1.4 Visualization Example

A Python script is provided to create basic visualization of the model output. The script is designed to output graphics in PNG format for 14 standard meteorological variables when using the pre-defined CONUS domain. In addition, a difference plotting script is included to visually compare two runs for the same domain and resolution. These scripts are provided only as an example for users familiar with Python, and may be used to do a visual check to verify that the application is producing reasonable results.

The scripts are available in the regional_workflow repository under ush/Python. Usage information and instructions are described in Chapter 10 and are also included at the top of the script.

1.5 Build System and Workflow

The SRW Application has a portable build system and a user-friendly, modular, and expandable workflow framework.

An umbrella CMake-based build system is used for building the components necessary for running the end-to-end SRW Application: the UFS Weather Model and the pre- and post-processing software. Additional libraries (*NCEPLIBS-external* and *NCEPLIBS*) necessary for the application are not included in the SRW Application build system, but are available pre-built on pre-configured platforms. There is a small set of system libraries and utilities that are assumed to be present on the target computer: the CMake build software, a Fortran, C, and C++ compiler, and MPI library.

Once built, the provided experiment generator script can be used to create a Rocoto-based workflow file that will run each task in the system (see Rocoto documentation) in the proper sequence. If Rocoto and/or a batch system is not present on the available platform, the individual components can be run in a stand-alone, command line fashion with provided run scripts. The generated namelist for the atmospheric model can be modified in order to vary settings such as forecast starting and ending dates, forecast length hours, the CCPP physics suite, integration time step, history file output frequency, and more. It also allows for configuration of other elements of the workflow; for example, whether to run some or all of the pre-processing, forecast model, and post-processing steps.

This SRW Application release has been tested on a variety of platforms widely used by researchers, such as the NOAA Research and Development High-Performance Computing Systems (RDHPCS), including Hera, Orion, and Jet; NOAA's Weather and Climate Operational Supercomputing System

(WCOSS); the National Center for Atmospheric Research (NCAR) Cheyenne system; NSSL's HPC machine, Odin; the National Science Foundation Stampede2 system; and generic Linux and macOS systems using Intel and GNU compilers. Four levels of support have been defined for the SRW Application, including pre-configured (level 1), configurable (level 2), limited test platforms (level 3), and build only platforms (level 4). Each level is further described below.

For the selected computational platforms that have been pre-configured (level 1), all the required libraries for building the SRW Application are available in a central place. That means bundled libraries (NCEPLIBS) and third-party libraries (NCEPLIBS-external) have both been built. The SRW Application is expected to build and run out of the box on these pre-configured platforms and users can proceed directly to the using the workflow, as described in the Quick Start (Chapter 2).

A few additional computational platforms are considered configurable for the SRW Application release. Configurable platforms (level 2) are platforms where all of the required libraries for building the SRW Application are expected to install successfully, but are not available in a central place. Applications and models are expected to build and run once the required bundled libraries (NCEPLIBS) and third-party libraries (NCEPLIBS-external) are built.

Limited-Test (level 3) and Build-Only (level 4) computational platforms are those in which the developers have built the code but little or no pre-release testing has been conducted, respectively. A complete description of the levels of support, along with a list of preconfigured and configurable platforms can be found in the SRW Application wiki page.

1.6 User Support, Documentation, and Contributing Development

A forum-based, online support system with topical sections provides a centralized location for UFS users and developers to post questions and exchange information. The forum complements the formal, written documentation, summarized here for ease of use.

A list of available documentation is shown in Table 1.1.

Table 1.1: Centralized list of documentation

Documentation	Location
UFS SRW Application v1.0 User's Guide	https://ufs-srweather-app.readthedocs.io/en/ufs-v1.0.0
UFS_UTILS v2.0 User's Guide	https://noaa-emcufs-utils.readthedocs.io/en/ufs-v2.0.0/
UFS Weather Model v2.0 User's Guide	https://ufs-weather-model.readthedocs.io/en/ufs-v2.0.0
NCEPLIBS Documentation	https://github.com/NOAA-EMC/NCEPLIBS/wiki
NCEPLIBS-external Documentation	https://github.com/NOAA-EMC/NCEPLIBS-external/wiki
FV3 Documentation	https://noaa-emc.github.io/FV3_Dycore_ufs-v2.0.0/html/index.html
CCPP Scientific Documentation	https://dtcenter.ucar.edu/GMTB/v5.0.0/sci_doc/index.html
CCPP Technical Documentation	https://ccpp-techdoc.readthedocs.io/en/v5.0.0/
ESMF manual	http://earthsystemmodeling.org/docs/release/ESMF_8_0_0/ESMF_usrdoc/
Unified Post Processor	https://upp.readthedocs.io/en/upp-v9.0.0/

The UFS community is encouraged to contribute to the development effort of all related utilities, model code, and infrastructure. Issues can be posted in the GitHub repository for the SRW Application or the relevant subcomponent to report bugs or to announce upcoming contributions to the code base. For code to be accepted in the authoritative repositories, the code management rules of each component (described in the User's Guides listed in Table 1.1 need to be followed.

1.7 Future Direction

Users can expect to see incremental improvements and additional capabilities in upcoming releases of the SRW Application to enhance research opportunities and support operational forecast implementations. Planned advancements include:

- A more extensive set of supported developmental physics suites.
- A larger number of pre-defined domains/resolutions and a fully supported capability to create a user-defined domain.
- Inclusion of data assimilation, cycling, and ensemble capabilities.
- A verification package (i.e., METplus) integrated into the workflow.
- Inclusion of stochastic perturbation techniques.

In addition to the above list, other improvements will be addressed in future releases.

1.8 How to Use This Document

This guide instructs both novice and experienced users on downloading, building and running the SRW Application. Please post questions in the UFS forum at https://forums.ufscommunity.org/.

```
Throughout the guide, this presentation style indicates shell
commands and options, code examples, etc.
```

Note: Variables presented as `AaBbCc123` in this document typically refer to variables in scripts, names of files and directories.

CHAPTER

TWO

WORKFLOW QUICK START

To build and run the out-of-the-box case of the UFS Short-Range Weather (SRW) Application the user must get the source code for multiple components, including: the regional workflow, the UFS_UTILS pre-processor utilities, the UFS Weather Model, and the Unified Post Processor (UPP). Once the UFS SRW Application umbrella repository is cloned, obtaining the necessary external repositories is simplified by the use of `manage_externals`. The out-of-the-box case uses a predefined 25-km CONUS grid (RRFS_CONUS_25km), the GFS version 15.2 physics suite (FV3_GFS_v15p2 CCPP), and FV3-based GFS raw external model data for initialization.

Note: The steps described in this chapter are applicable to preconfigured (Level 1) machines where all of the required libraries for building community releases of UFS models and applications are available in a central place (i.e. the bundled libraries (NCEPLIBS) and third-party libraries (NCEPLIBS-external) have both been built). The Level 1 platforms are listed here. For more information on compiling NCEPLIBS-external and NCEPLIBS, please refer to the NCEPLIBS-external wiki.

2.1 Download the UFS SRW Application Code

The necessary source code is publicly available on GitHub. To clone the release branch of the repository:

```
git clone -b ufs-v1.0.0 https://github.com/ufs-community/ufs-srweather-app.git
cd ufs-srweather-app
```

Then, check out the submodules for the SRW application:

```
./manage_externals/checkout_externals
```

The `checkout_externals` script uses the configuration file `Externals.cfg` in the top level directory and will clone the regional workflow, pre-processing utilities, UFS Weather Model, and UPP source code into the appropriate directories under your `regional_workflow` and `src` directories.

2.2 Set up the Build Environment

Instructions for loading the proper modules and/or setting the correct environment variables can be found in the `env/` directory in files named `build_<platform>_<compiler>.env`. The commands in these files can be directly copy-pasted to the command line or the file can be sourced. You may need to modify certain variables such as the path to NCEP libraries for your individual platform, or use `setenv` rather than `export` depending on your environment:

```
$ ls -l env/
   -rw-rw-r-- 1 user ral 466 Jan  21 10:09 build_cheyenne_intel.env
   -rw-rw-r-- 1 user ral 461 Jan  21 10:09 build_hera_intel.env
   -rw-rw-r-- 1 user ral 543 Jan  21 10:09 build_jet_intel.env
```

2.3 Build the Executables

Build the executables as follows:

```
mkdir build
cd build
```

Run `cmake` to set up the `Makefile`, then run `make`:

```
cmake .. -DCMAKE_INSTALL_PREFIX=..
make -j 4  >& build.out &
```

Output from the build will be in the `ufs-srweather-app/build/build.out` file. When the build completes, you should see the forecast model executable `NEMS.exe` and eleven pre- and post-processing executables in the `ufs-srweather-app/bin` directory which are described in Table 4.2.

2.4 Generate the Workflow Experiment

Generating the workflow experiment requires three steps:

- Set experiment parameters in config.sh
- Set Python and other environment parameters
- Run the `generate_FV3LAM_wflow.sh` script

The first two steps depend on the platform being used and are described here for each Level 1 platform.

2.4.1 Set up config.sh file

The workflow requires a file called config.sh to specify the values of your experiment parameters. Two example templates are provided: config.community.sh and config.nco.sh and can be found in the ufs-srweather-app/regional_workflow/ush directory. The first file is a minimal example for creating and running an experiment in the *community* mode (with RUN_ENVIR set to community), while the second is an example of creating and running an experiment in the *NCO* (operational) mode (with RUN_ENVIR set to nco). The *community* mode is recommended in most cases and will be fully supported for this release while the operational mode will be more exclusively used by NOAA/NCEP Central Operations (NCO) and those in the NOAA/NCEP/Environmental Modeling Center (EMC) working with NCO on pre-implementation testing. Sample config.sh files are discussed in this section for Level 1 platforms.

Make a copy of config.community.sh to get started (under /path-to-ufs-srweather-app/regional_workflow/ush):

```
cd ../regional_workflow/ush
cp config.community.sh config.sh
```

Edit the config.sh file to set the machine you are running on to MACHINE, use an account you can charge for ACCOUNT, and set the name of the experiment with EXPT_SUBDIR. If you have access to the NOAA HPSS from the machine you are running on, those changes should be sufficient; however, if that is not the case (for example, on Cheyenne), or if you have pre-staged the initialization data you would like to use, you will also want to set USE_USER_STAGED_EXTRN_FILES="TRUE" and set the paths to the data for EXTRN_MDL_SOURCE_BASEDIR_ICS and EXTRN_MDL_SOURCE_BASEDIR_LBCS.

At a minimum, the following parameters should be set for the machine you are using:

For Cheyenne:

```
MACHINE="cheyenne"
ACCOUNT="my_account"
EXPT_SUBDIR="my_expt_name"
USE_USER_STAGED_EXTRN_FILES="TRUE"
EXTRN_MDL_SOURCE_BASEDIR_ICS="/glade/p/ral/jntp/UFS_SRW_app/model_data/FV3GFS"
EXTRN_MDL_SOURCE_BASEDIR_LBCS="/glade/p/ral/jntp/UFS_SRW_app/model_data/FV3GFS"
```

For Hera:

```
MACHINE="hera"
ACCOUNT="my_account"
EXPT_SUBDIR="my_expt_name"
```

For Jet:

```
MACHINE="jet"
ACCOUNT="my_account"
EXPT_SUBDIR="my_expt_name"
```

For Orion:

```
MACHINE="orion"
ACCOUNT="my_account"
EXPT_SUBDIR="my_expt_name"
```

For Gaea:

```
MACHINE="gaea"
ACCOUNT="my_account"
EXPT_SUBDIR="my_expt_name"
```

For WCOSS, edit `config.sh` with these WCOSS-specific parameters, and use a valid WCOSS project code for the account parameter:

```
MACHINE="wcoss_cray" or MACHINE="wcoss_dell_p3"
ACCOUNT="my_account"
EXPT_SUBDIR="my_expt_name"
```

2.4.2 Set up the Python and other Environment Parameters

Next, it is necessary to load the appropriate Python environment for the workflow. The workflow requires Python 3, with the packages 'PyYAML', 'Jinja2', and 'f90nml' available. This Python environment has already been set up on Level 1 platforms, and can be activated in the following way (when in /path-to-ufs-srweather-app/regional_workflow/ush):

```
source ../../env/wflow_<platform>.env
```

2.4.3 Run the `generate_FV3LAM_wflow.sh` script

For all platforms, the workflow can then be generated with the command:

```
./generate_FV3LAM_wflow.sh
```

The generated workflow will be in `$EXPTDIR`, where `EXPTDIR=${EXPT_BASEDIR}/${EXPT_SUBDIR}`. A log file called `log.generate_FV3LAM_wflow` is generated by this step and can also be found in `$EXPTDIR`. The settings for these paths can be found in the output from the `./generate_FV3LAM_wflow.sh` script.

2.5 Run the Workflow Using Rocoto

The information in this section assumes that Rocoto is available on the desired platform. If Rocoto is not available, it is still possible to run the workflow using stand-alone scripts described in Section 4.8.3. There are two ways you can run the workflow with Rocoto using either the `./launch_FV3LAM_wflow.sh` or by hand.

An environment variable may be set to navigate to the `$EXPTDIR` more easily. If the login shell is bash, it can be set as follws:

```
export EXPTDIR=/path-to-experiment/directory
```

Or if the login shell is csh/tcsh, it can be set using:

```
setenv EXPTDIR /path-to-experiment/directory
```

To run Rocoto using the script:

```
cd $EXPTDIR
./launch_FV3LAM_wflow.sh
```

Once the workflow is launched with the `launch_FV3LAM_wflow.sh` script, a log file named `log.launch_FV3LAM_wflow` will be created (or appended to it if it already exists) in `EXPTDIR`.

Or to manually call Rocoto:

First load the Rocoto module, depending on the platform used.

For Cheyenne:

```
module use -a /glade/p/ral/jntp/UFS_SRW_app/modules/
module load rocoto
```

For Hera or Jet:

```
module purge
module load rocoto
```

For Orion:

```
module purge
module load contrib rocoto
```

For Gaea:

```
module use /lustre/f2/pdata/esrl/gsd/contrib/modulefiles
module load rocoto/1.3.3
```

For WCOSS_DELL_P3:

```
module purge
module load lsf/10.1
module use /gpfs/dell3/usrx/local/dev/emc_rocoto/modulefiles/
module load ruby/2.5.1 rocoto/1.2.4
```

For WCOSS_DELL_P3:

```
module purge
module load xt-lsfhpc/9.1.3
module use -a /usrx/local/emc_rocoto/modulefiles
module load rocoto/1.2.4
```

Then manually call `rocotorun` to launch the tasks that have all dependencies satisfied and `rocotostat` to monitor the progress:

```
cd $EXPTDIR
rocotorun -w FV3LAM_wflow.xml -d FV3LAM_wflow.db -v 10
rocotostat -w FV3LAM_wflow.xml -d FV3LAM_wflow.db -v 10
```

For automatic resubmission of the workflow (e.g., every 3 minutes), the following line can be added to the user's crontab (use `crontab -e` to edit the cron table).

```
*/3 * * * * cd /glade/p/ral/jntp/$USER/expt_dirs/test_CONUS_25km_GFSv15p2 && ./launch_FV3LAM_
↪wflow.sh
```

Note: Currently cron is only available on the orion-login-1 node, so please use that node.

The workflow run is completed when all tasks have "SUCCEEDED", and the rocotostat command will output the following:

```
CYCLE               TASK                JOBID               STATE          EXIT STATUS  ␣
↪TRIES   DURATION
==========================================================================================================================
201906150000          make_grid           4953154            SUCCEEDED          0          1  ␣
↪       5.0
201906150000          make_orog           4953176            SUCCEEDED          0          1  ␣
↪      26.0
201906150000          make_sfc_climo      4953179            SUCCEEDED          0          1  ␣
↪      33.0
201906150000          get_extrn_ics       4953155            SUCCEEDED          0          1  ␣
↪       2.0
201906150000          get_extrn_lbcs      4953156            SUCCEEDED          0          1  ␣
↪       2.0
201906150000          make_ics            4953184            SUCCEEDED          0          1  ␣
↪      16.0
201906150000          make_lbcs           4953185            SUCCEEDED          0          1  ␣
↪      71.0
201906150000          run_fcst            4953196            SUCCEEDED          0          1  ␣
↪    1035.0
201906150000          run_post_f000       4953244            SUCCEEDED          0          1  ␣
↪       5.0
201906150000          run_post_f001       4953245            SUCCEEDED          0          1  ␣
↪       4.0
...
201906150000          run_post_f048       4953381            SUCCEEDED          0          1  ␣
↪       7.0
```

2.6 Plot the Output

Two python scripts are provided to generate plots from the FV3-LAM post-processed GRIB2 output. Information on how to generate the graphics can be found in Chapter 10.

CHAPTER

THREE

CODE REPOSITORIES AND DIRECTORY STRUCTURE

This chapter describes the code repositories that comprise the UFS SRW Application, without describing any of the components in detail.

3.1 Hierarchical Repository Structure

The umbrella repository for the UFS SRW Application is named ufs-srweather-app and is available on GitHub at https://github.com/ufs-community/ufs-srweather-app. An umbrella repository is defined as a repository that houses external code, called "externals," from additional repositories. The UFS SRW Application includes the `manage_externals` tools along with a configuration file called `Externals.cfg`, which describes the external repositories associated with this umbrella repo (see Table 3.1).

Table 3.1: List of top-level repositories that comprise the UFS SRW Application.

Repository Description	Authoritative repository URL
Umbrella repository for the UFS Short-Range Weather Application	https://github.com/ufs-community/ufs-srweather-app
Repository for the UFS Weather Model	https://github.com/ufs-community/ufs-weather-model
Repository for the regional workflow	https://github.com/NOAA-EMC/regional_workflow
Repository for UFS utilities, including pre-processing, chgres_cube, and more	https://github.com/NOAA-EMC/UFS_UTILS
Repository for the Unified Post Processor (UPP)	https://github.com/NOAA-EMC/EMC_post

The UFS Weather Model contains a number of sub-repositories used by the model as documented here.

Note that the prerequisite libraries (including NCEP Libraries and external libraries) are not included in the UFS SRW Application repository. The source code for these components resides in the repositories NCEPLIBS and NCEPLIBS-external.

These external components are already built on the preconfigured platforms listed here. However, they must be cloned and built on other platforms according to the instructions provided in the wiki

pages of those repositories: https://github.com/NOAA-EMC/NCEPLIBS/wiki and https://github.com/NOAA-EMC/NCEPLIBS-external/wiki.

3.2 Directory Structure

The directory structure for the SRW Application is determined by the `local_path` settings in the `Externals.cfg` file, which is in the directory where the umbrella repository has been cloned. After `manage_externals/checkout_externals` is run, the specific GitHub repositories that are described in Table 3.1 are cloned into the target subdirectories shown below. The directories that will be created later by running the scripts are presented in parentheses. Some directories have been removed for brevity.

```
ufs-srweather-app
├── (bin)
├── (build)
├── docs
│   └── UsersGuide
├── (include)
├── (lib)
├── manage_externals
├── regional_workflow
│   ├── docs
│   │   └── UsersGuide
│   ├── (fix)
│   ├── jobs
│   ├── modulefiles
│   ├── scripts
│   ├── tests
│   │   └── baseline_configs
│   └── ush
│       ├── Python
│       ├── rocoto
│       ├── templates
│       └── wrappers
├── (share)
└── src
    ├── EMC_post
    │   ├── parm
    │   └── sorc
    │       └── ncep_post.fd
    ├── UFS_UTILS
    │   ├── sorc
    │   │   ├── chgres_cube.fd
    │   │   ├── fre-nctools.fd
    │   │   ├── grid_tools.fd
    │   │   ├── orog_mask_tools.fd
    │   │   └── sfc_climo_gen.fd
    │   └── ush
    └── ufs_weather_model
        └── FV3
```

(continues on next page)

(continued from previous page)

```
├── atmos_cubed_sphere
└── ccpp
```

3.2.1 Regional Workflow Sub-Directories

Under the `regional_workflow` directory shown in Section 3.2 there are a number of sub-directories that are created when the regional workflow is cloned. The contents of these sub-directories are described in Table 3.2.

Table 3.2: Sub-directories of the regional workflow.

Directory Name	Description
docs	Users' Guide Documentation
jobs	J-job scripts launched by Rocoto
modulefiles	Files used to load modules needed for building and running the workflow
scripts	Run scripts launched by the J-jobs
tests	Baseline experiment configuration
ush	Utility scripts used by the workflow

3.3 Experiment Directory Structure

When the `generate_FV3LAM_wflow.sh` script is run, the user-defined experimental directory `EXPTDIR=/path-to/ufs-srweather-app/../expt_dirs/${EXPT_SUBDIR}` is created, where `EXPT_SUBDIR` is specified in the `config.sh` file. The contents of the `EXPTDIR` directory, before the workflow is run, is shown in Table 3.3.

Table 3.3: Files and sub-directory initially created in the experimental directory.

File Name	Description
config.sh	User-specified configuration file, see Section 4.5.2
data_table	Cycle-independent input file (empty)
field_table	Tracers in the forecast model
FV3LAM_wflow.xml	Rocoto XML file to run the workflow
input.nml	Namelist for the UFS Weather model
launch_FV3LAM_wflow.sh	Symlink to the shell script of `ufs-srweather-app/regional_workflow/ush/launch_FV3LAM_wflow.sh` that can be used to (re)launch the Rocoto workflow. Each time this script is called, it appends to a log file named `log.launch_FV3LAM_wflow`.
log.generate_FV3LAM_wflow	Log of the output from the experiment generation script `generate_FV3LAM_wflow.sh`
nems.configure	See NEMS configuration file
suite_{CCPP}.xml	CCPP suite definition file used by the forecast model
var_defns.sh	Shell script defining the experiment parameters. It contains all of the primary parameters specified in the default and user-specified configuration files plus many secondary parameters that are derived from the primary ones by the experiment generation script. This file is sourced by various other scripts in order to make all the experiment variables available to these scripts.
YYYYMMDDHH	Cycle directory (empty)

In addition, the *community* mode creates the `fix_am` and `fix_lam` directories in `EXPTDIR`. The `fix_lam` directory is initially empty but will contain some *fix* (time-independent) files after the grid, orography, and/or surface climatology generation tasks are run.

Table 3.4: Description of the fix directories

Directory Name	Description
fix_am	Directory containing the global *fix* (time-independent) data files. The experiment generation script copies these files from a machine-dependent system directory.
fix_lam	Directory containing the regional fix (time-independent) data files that describe the regional grid, orography, and various surface climatology fields as well as symlinks to pre-generated files.

Once the workflow is launched with the `launch_FV3LAM_wflow.sh` script, a log file named `log.launch_FV3LAM_wflow` will be created (or appended to it if it already exists) in `EXPTDIR`. Once the `make_grid`, `make_orog`, and `make_sfc_climo` tasks and the `get_extrn_ics` and `get_extrn_lbc` tasks for the YYYYMMDDHH cycle have completed successfully, new files and sub-directories are created, as described in Table 3.5.

Table 3.5: New directories and files created when the workflow is launched.

Directory/file Name	Description
YYYYMMDDHH	This is updated when the first cycle-specific workflow tasks are run, which are `get_extrn_ics` and `get_extrn_lbcs` (they are launched simultaneously for each cycle in the experiment). We refer to this as a "cycle directory". Cycle directories are created to contain cycle-specific files for each cycle that the experiment runs. If `DATE_FIRST_CYCL` and `DATE_LAST_CYCL` were different, and/or `CYCL_HRS` contained more than one element in the `config.sh` file, then more than one cycle directory would be created under the experiment directory.
grid	Directory generated by the `make_grid` task containing grid files for the experiment
log	Contains log files generated by the overall workflow and its various tasks. Look in these files to trace why a task may have failed.
orog	Directory generated by the `make_orog` task containing the orography files for the experiment
sfc_climo	Directory generated by the `make_sfc_climo` task containing the surface climatology files for the experiment
FV3LAM_wflow.db FV3LAM_wflow_lock.db	Database files that are generated when Rocoto is called (by the launch script) to launch the workflow.
log.launch_FV3LAM_wflow	This is the log file to which the launch script `launch_FV3LAM_wflow.sh` appends its output each time it is called. Take a look at the last 30–50 lines of this file to check the status of the workflow.

The output files for an experiment are described in Section 7.2. The workflow tasks are described in Section 4.7.2).

CHAPTER

FOUR

SHORT-RANGE WEATHER APPLICATION OVERVIEW

The UFS Short-Range Weather Application (SRW App) is an umbrella repository that contains the tool `manage_externals` to check out all of the components required for the application. Once the build process is complete, all the files and executables necessary for a regional experiment are located in the `regional_workflow` and `bin` directories, respectively, under the `ufs-srweather-app` directory. Users can utilize the pre-defined domains or build their own domain (details provided in Chapter 6). In either case, users must create/modify the case-specific (`config.sh`) and/or grid-specific configuration files (`set_predef_grid_params.sh`). The overall procedure is shown in Figure 4.1, with the scripts to generate and run the workflow shown in red. The steps are as follows:

1. Clone the UFS Short Range Weather Application from GitHub.
2. Check out the external repositories.
3. Set up the build environment and build the regional workflow system using `cmake/make`.
4. Optional: Add new grid information to the `set_predef_grid_param.sh` configuration file and update `valid_param_vals.sh`.
5. Modify the case-specific `config.sh` configuration file.
6. Load the python environment for the regional workflow
7. Generate a regional workflow experiment.
8. Run the regional workflow as needed.

Each step will be described in detail in the following sections.

4.1 Download from GitHub

Retrieve the UFS Short Range Weather Application (SRW App) repository from GitHub and check-out the `ufs-v1.0.0` tag:

```
git clone -b ufs-v1.0.0 https://github.com/ufs-community/ufs-srweather-app.git
cd ufs-srweather-app
```

The cloned repository contains the configuration files and sub-directories shown in Table 4.1.

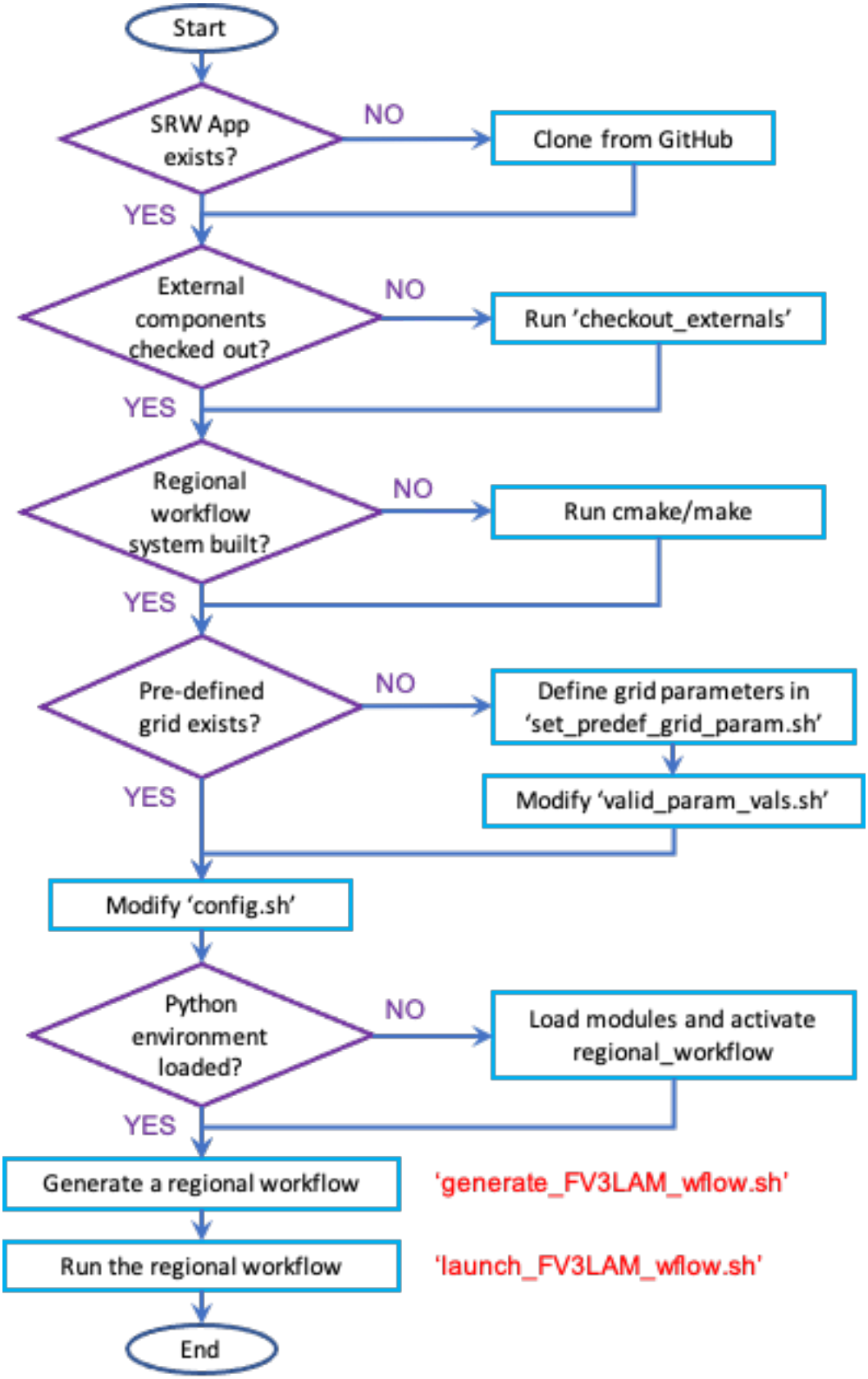

Fig. 4.1: *Overall layout of the SRW App.*

Table 4.1: Files and sub-directories of the ufs-srweather-app repository.

File/directory Name	Description
CMakeLists.txt	Main cmake file for SRW App
Externals.cfg	Tags of the GitHub repositories/branches for the external repositories
LICENSE.md	CC0 license information
README.md	Quick start guide
ufs_srweather_app_meta.h.in	Meta information for SRW App which can be used by other packages
ufs_srweather_app.settings.in	SRW App configuration summary
env	Contains build and workflow environment files
docs	Contains release notes, documentation, and Users' Guide
manage_externals	Utility for checking out external repositories
src	Contains CMakeLists.txt; external repositories will be cloned in this directory.

4.2 External Components

Check out the external repositories, including regional_workflow, ufs-weather-model, ufs_utils, and emc_post for the SRW App.

```
./manage_externals/checkout_externals
```

This step will use the configuration `Externals.cfg` file in the `ufs-srweather-app` directory to clone the specific tags (version of codes) of the external repositories as listed in Section 3.1.

4.3 Building the Executables for the Application

Before building the executables, the build environment must be set up for your specific platform. Instructions for loading the proper modules and/or setting the correct environment variables can be found in the `env/` directory in files named `build_<platform>_<compiler>.env`. For the most part, the commands in those files can be directly copied and pasted, but you may need to modify certain variables such as the path to NCEP libraries for your specific platform. Here is a directory listing example of these kinds of files:

```
$ ls -l env/
   -rw-rw-r-- 1 user ral 1228 Oct  9 10:09 build_cheyenne_intel.env
   -rw-rw-r-- 1 user ral 1134 Oct  9 10:09 build_hera_intel.env
   -rw-rw-r-- 1 user ral 1228 Oct  9 10:09 build_jet_intel.env
   ...
```

The following steps will build the pre-processing utilities, forecast model, and post-processor:

```
make dir
cd build
cmake .. -DCMAKE_INSTALL_PREFIX=..
make -j 4 >& build.out &
```

where `-DCMAKE_INSTALL_PREFIX` specifies the location in which the `bin`, `include`, `lib`, and `share` directories containing various components of the SRW App will be created, and its recommended value `..` denotes one directory up from the build directory. In the next line for the `make` call, `-j 4` indicates the build will run in parallel with 4 threads. If this step is successful, the executables listed in Table 4.2 will be located in the `ufs-srweather-app/bin` directory.

Table 4.2: Names and descriptions of the executables produced by the build step and used by the SRW App.

Executable Name	Description
chgres_cube	Reads in raw external model (global or regional) and surface climatology data to create initial and lateral boundary conditions
filter_topo	Filters topography based on resolution
global_equiv_resol	Calculates a global, uniform, cubed-sphere equivalent resolution for the regional Extended Schmidt Gnomonic (ESG) grid
make_solo_mosaic	Creates mosaic files with halos
ncep_post	Post-processor for the model output
NEMS.exe	UFS Weather Model executable
orog	Generates orography, land mask, and gravity wave drag files from fixed files
regional_esg_grid	Generates an ESG regional grid based on a user-defined namelist
sfc_climo_gen	Creates surface climatology fields from fixed files for use in `chgres_cube`
shave	Shaves the excess halo rows down to what is required for the LBCs in the orography and grid files
vcoord_gen	Generates hybrid coordinate interface profiles

4.4 Grid-specific Configuration

Some SRW App parameters depend on the characteristics of the grid such as resolution and domain size. These include `ESG grid` and `Input configuration` as well as the variables related to the write component (quilting). The SRW App officially supports three different predefined grids as shown in Table 4.3. Their names can be found under `valid_vals_PREDEF_GRID_NAME` in the `valid_param_vals` script, and their grid-specific configuration variables are specified in the `set_predef_grid_params` script. If users want to create a new domain, they should put its name in the `valid_param_vals` script and the corresponding grid-specific parameters in the `set_predef_grid_params` script. More information on the predefined and user-generated options can be found in Chapter 6.

Table 4.3: Predefined grids in the SRW App.

Grid Name	Grid Type	Quilting (write component)
RRFS_CONUS_25km	ESG grid	lambert_conformal
RRFS_CONUS_13km	ESG grid	lambert_conformal
RRFS_CONUS_3km	ESG grid	lambert_conformal

4.5 Case-specific Configuration

4.5.1 Default configuration: `config_defaults.sh`

When generating a new experiment (described in detail in Section 4.7), the `config_defaults.sh` file is read first and assigns default values to the experiment parameters. Important configuration variables in the `config_defaults.sh` file are shown in Table 4.4, with more documentation found in the file itself, and in Chapter 5. Some of these default values are intentionally invalid in order to ensure that the user assigns valid values in the user-specified configuration `config.sh` file. Therefore, any settings provided in `config.sh` will override the default `config_defaults.sh` settings. Note that there is usually no need for a user to modify the default configuration file.

Table 4.4: Configuration variables specified in the config_defaults.sh script.

Group Name	Configuration variables
Experiment mode	RUN_ENVIR
Machine and queue	MACHINE, ACCOUNT, SCHED, PARTITION_DEFAULT, QUEUE_DEFAULT, PARTITION_HPSS, QUEUE_HPSS, PARTITION_FCST, QUEUE_FCST
Cron	USE_CRON_TO_RELAUNCH, CRON_RELAUNCH_INTVL_MNTS
Experiment Dir.	EXPT_BASEDIR, EXPT_SUBDIR
NCO mode	COMINgfs, STMP, NET, envir, RUN, PTMP
Separator	DOT_OR_USCORE
File name	EXPT_CONFIG_FN, RGNL_GRID_NML_FN, DATA_TABLE_FN, DIAG_TABLE_FN, FIELD_TABLE_FN, FV3_NML_BASE_SUITE_FN, FV3_NML_YALM_CONFIG_FN, FV3_NML_BASE_ENS_FN, MODEL_CONFIG_FN, NEMS_CONFIG_FN, FV3_EXEC_FN, WFLOW_XML_FN, GLOBAL_VAR_DEFNS_FN, EXTRN_MDL_ICS_VAR_DEFNS_FN, EXTRN_MDL_LBCS_VAR_DEFNS_FN, WFLOW_LAUNCH_SCRIPT_FN, WFLOW_LAUNCH_LOG_FN
Forecast	DATE_FIRST_CYCL, DATE_LAST_CYCL, CYCL_HRS, FCST_LEN_HRS
IC/LBC	EXTRN_MDL_NAME_ICS, EXTRN_MDL_NAME_LBCS, LBC_SPEC_INTVL_HRS, FV3GFS_FILE_FMT_ICS, FV3GFS_FILE_FMT_LBCS
NOMADS	NOMADS, NOMADS_file_type
External model	USE_USER_STAGED_EXTRN_FILES, EXTRN_MDL_SOURCE_BASEDRI_ICS, EXTRN_MDL_FILES_ICS, EXTRN_MDL_SOURCE_BASEDIR_LBCS, EXTRN_MDL_FILES_LBCS
CCPP	CCPP_PHYS_SUITE
GRID	GRID_GEN_METHOD
ESG grid	ESGgrid_LON_CTR, ESGgrid_LAT_CTR, ESGgrid_DELX, ESGgrid_DELY, ESGgrid_NX, ESGgrid_NY, ESGgrid_WIDE_HALO_WIDTH
Input config-uration	DT_ATMOS, LAYOUT_X, LAYOUT_Y, BLOCKSIZE, QUILTING, PRINT_ESMF, WRTCMP_write_groups, WRTCMP_write_tasks_per_group, WRTCMP_output_grid, WRTCMP_cen_lon, WRTCMP_cen_lat, WRTCMP_lon_lwr_left, WRTCMP_lat_lwr_left, WRTCMP_lon_upr_rght, WRTCMP_lat_upr_rght, WRTCMP_dlon, WRTCMP_dlat, WRTCMP_stdlat1, WRTCMP_stdlat2, WRTCMP_nx, WRTCMP_ny, WRTCMP_dx, WRTCMP_dy
Pre-existing	PREDEF_GRID_NAME, PREEXISTING_DIR_METHOD, VERBOSE

4.5.2 User-specific configuration: `config.sh`

Before generating an experiment, the user must create a `config.sh` file in the `ufs-srweather-app/regional_workflow/ush` directory by copying either of the example configuration files, `config.community.sh` for the community mode or `config.nco.sh` for the NCO mode, or creating their own `config.sh` file. Note that the *community mode* is recommended in most cases and will be fully supported for this release while the operational/NCO mode will be more exclusively used by those at the NOAA/NCEP/Environmental Modeling Center (EMC) and the NOAA/Global Systems Laboratory (GSL) working on pre-implementation testing. Table 4.5 shows the configuration variables, along with their default values in `config_default.sh` and the values defined in `config.community.sh`.

Note: The values of the configuration variables should be consistent with those in the `valid_param_vals script`. In addition, various example configuration files can be found in the `regional_workflow/tests/baseline_configs` directory.

Table 4.5: Configuration variables specified in the config.community.sh script.

Parameter	Default Value	``config.community.sh`` Value
MACHINE	"BIG_COMPUTER"	"hera"
ACCOUNT	"project_name"	"an_account"
EXPT_SUBDIR	""	"test_CONUS_25km_GFSv15p2"
VERBOSE	"TRUE"	"TRUE"
RUN_ENVIR	"nco"	"community"
PREEXISTING_DIR_METHOD	"delete"	"rename"
PREDEF_GRID_NAME	""	"RRFS_CONUS_25km"
GRID_GEN_METHOD	"ESGgrid"	"ESGgrid"
QUILTING	"TRUE"	"TRUE"
CCPP_PHYS_SUITE	"FV3_GSD_V0"	"FV3_GFS_v15p2"
FCST_LEN_HRS	"24"	"48"
LBC_SPEC_INTVL_HRS	"6"	"6"
DATE_FIRST_CYCL	"YYYYMMDD"	"20190615"
DATE_LAST_CYCL	"YYYYMMDD"	"20190615"
CYCL_HRS	("HH1" "HH2")	"00"
EXTRN_MDL_NAME_ICS	"FV3GFS"	"FV3GFS"
EXTRN_MDL_NAME_LBCS	"FV3GFS"	"FV3GFS"
FV3GFS_FILE_FMT_ICS	"nemsio"	"grib2"
FV3GFS_FILE_FMT_LBCS	"nemsio"	"grib2"
WTIME_RUN_FCST	"04:30:00"	"01:00:00"
USE_USER_STAGED_EXTRN_FILES	"FALSE"	"TRUE"
EXTRN_MDL_SOURCE_BASE_DIR_ICS	""	"/scratch2/BMC/det/UFS_SRW_app/v1p0/model_data/FV3GFS
EXTRN_MDL_FILES_ICS	""	"gfs.pgrb2.0p25.f000"
EXTRN_MDL_SOURCE_BASEDIR_LBCS	""	"/scratch2/BMC/det/UFS_SRW_app/v1p0/model_data/FV3GFS
EXTRN_MDL_FILES_LBCS	""	"gfs.pgrb2.0p25.f006"

4.6 Python Environment for Workflow

It is necessary to load the appropriate Python environment for the workflow. The workflow requires Python 3, with the packages 'PyYAML', 'Jinja2', and 'f90nml' available. This Python environment has already been set up on Level 1 platforms, and can be activated in the following way:

```
source ../../env/wflow_<platform>.env
```

when in the `ufs-srweather-app/regional_workflow/ush` directory.

4.7 Generating a Regional Workflow Experiment

4.7.1 Steps to a Generate a New Experiment

Generating an experiment requires running

```
generate_FV3LAM_wflow.sh
```

in the `ufs-srweather-app/regional_workflow/ush` directory. This is the all-in-one script for users to set up their experiment with ease. Figure 4.2 shows the flowchart for generating an experiment. First, it sets up the configuration parameters by running the `setup.sh` script. Second, it copies the time-independent (fix) files and other necessary input files such as `data_table`, `field_table`, `nems.configure`, `model_configure`, and the CCPP suite file from its location in the ufs-weather-model directory to the experiment directory (`EXPTDIR`). Third, it copies the weather model executable (`NEMS.exe`) from the `bin` directory to `EXPTDIR`, and creates the input namelist file `input.nml` based on the `input.nml.FV3` file in the regional_workflow/ush/templates directory. Lastly, it creates the workflow XML file `FV3LAM_wflow.xml` that is executed when running the experiment with the Rocoto workflow manager.

The `setup.sh` script reads three other configuration scripts: (1) `config_default.sh` (Section 4.5.1), (2) `config.sh` (Section 4.5.2), and (3) `set_predef_grid_params.sh` (Section 4.4). Note that these three scripts are read in order: `config_default.sh`, `config.sh`, then `set_predef_grid_params.sh`. If a parameter is specified differently in these scripts, the file containing the last defined value will be used.

4.7.2 Description of Workflow Tasks

The flowchart of the workflow tasks that are specified in the `FV3LAM_wflow.xml` file are illustrated in Figure 4.3, and each task is described in Table 4.6. The first three pre-processing tasks; `MAKE_GRID`, `MAKE_OROG`, and `MAKE_SFC_CLIMO` are optional. If the user stages pre-generated grid, orography, and surface climatology fix files, these three tasks can be skipped by setting `RUN_TASK_MAKE_GRID="FALSE"`, `RUN_TASK_MAKE_OROG="FALSE"`, and `RUN_TASK_MAKE_SFC_CLIMO="FALSE"` in the `regional_workflow/ush/config.sh` file before running the `generate_FV3LAM_wflow.sh` script. As shown in the figure, the `FV3LAM_wflow.xml` file runs the specific j-job scripts in the prescribed order (`regional_workflow/jobs/JREGIONAL_[task name]`) when the `launch_FV3LAM_wflow.sh` is submitted. Each j-job task has its own source script named `exregional_[task name].sh` in the `regional_workflow/scripts` directory. Two database files `FV3LAM_wflow.db` and `FV3LAM_wflow_lock.db` are generated and updated by the Rocoto calls. There is usually no need for users to modify these files. To relaunch the workflow from scratch, delete these two *.db files and then call the launch script repeatedly for each task.

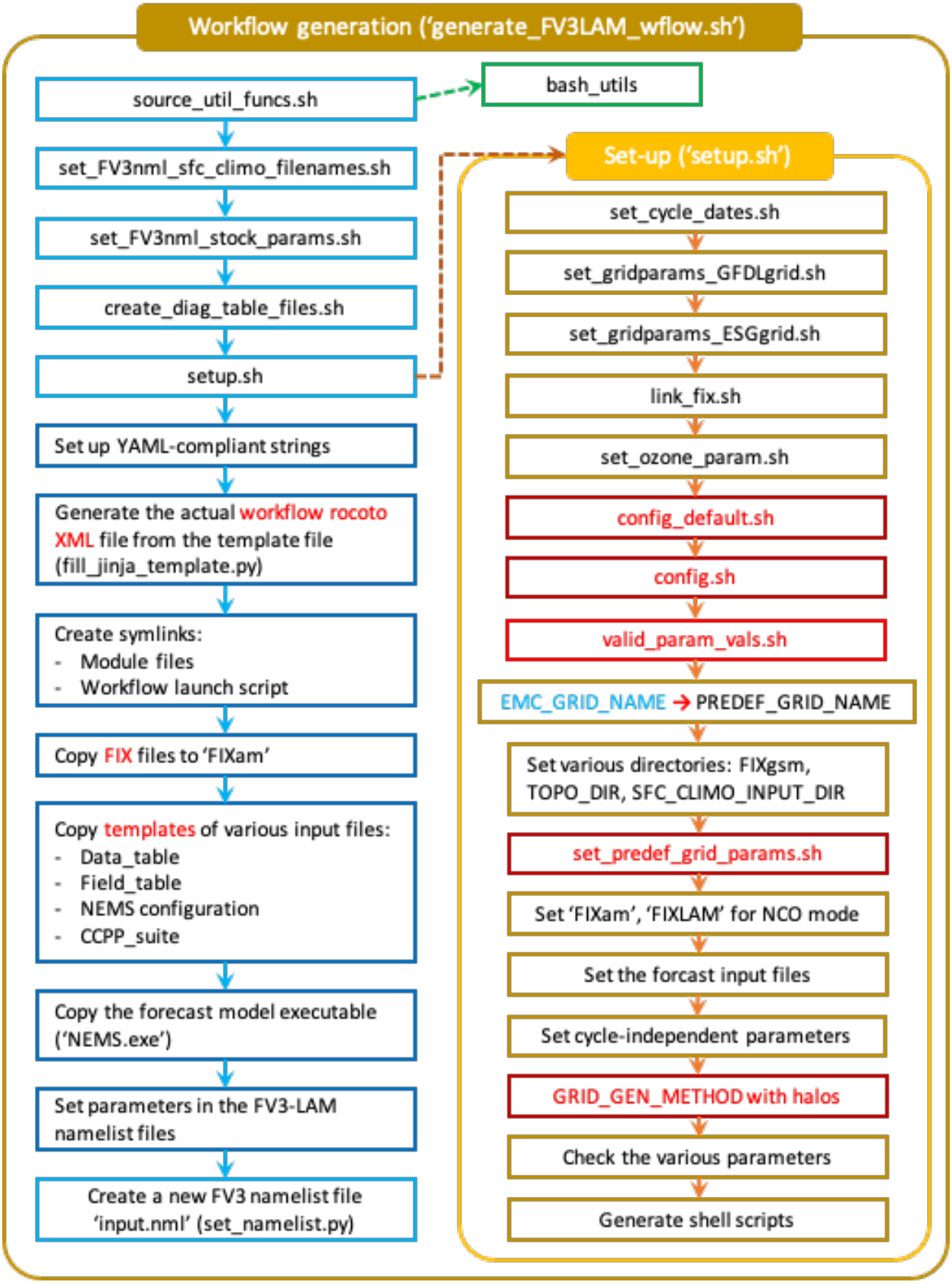

Fig. 4.2: *Experiment generation description*

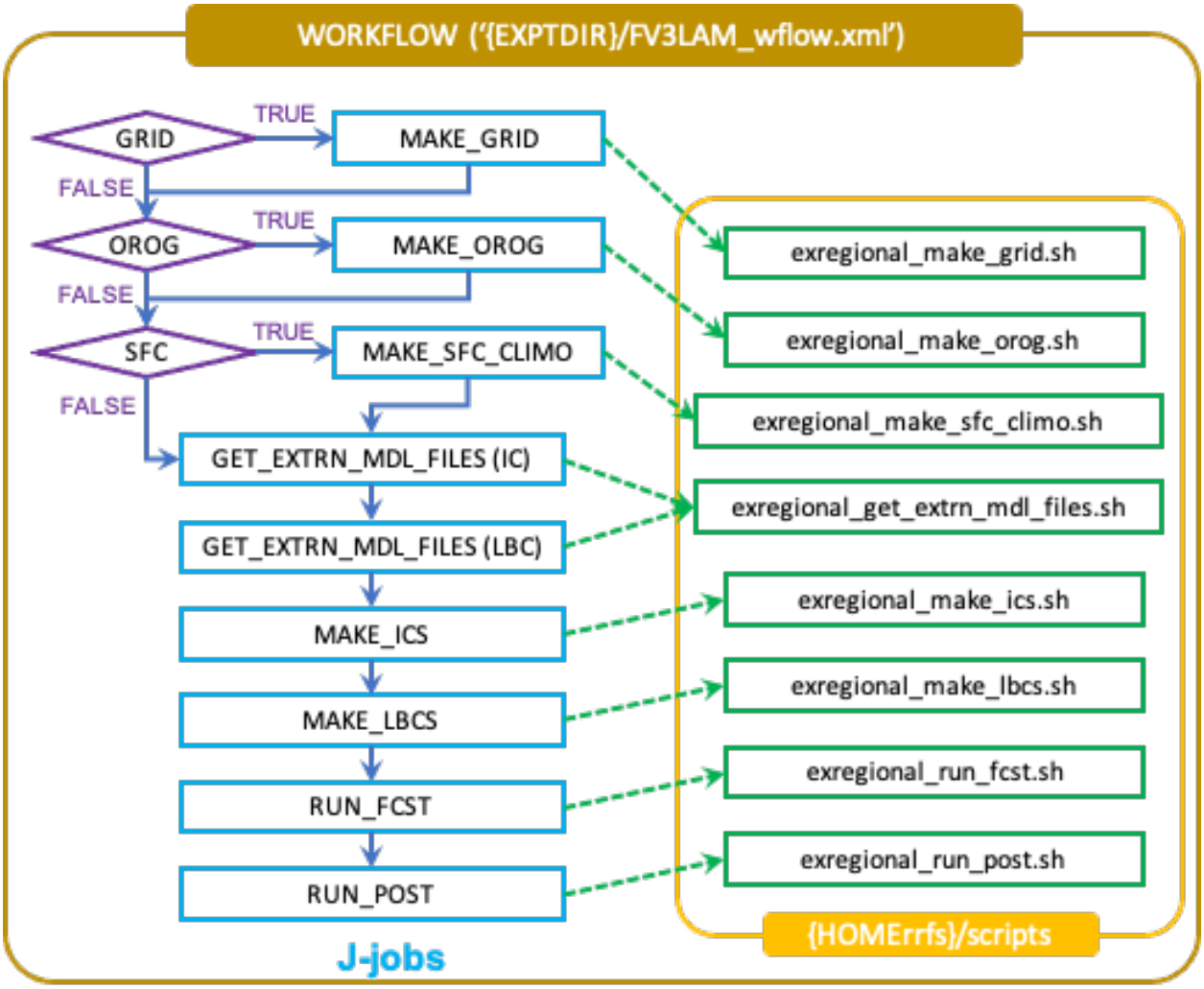

Fig. 4.3: *Flowchart of the workflow tasks*

Table 4.6: Workflow tasks in SRW App

Workflow Task	Task Description
make_grid	Pre-processing task to generate regional grid files. Can be run, at most, once per experiment.
make_orog	Pre-processing task to generate orography files. Can be run, at most, once per experiment.
make_sfc_climo	Pre-processing task to generate surface climatology files. Can be run, at most, once per experiment.
get_extrn_ics	Cycle-specific task to obtain external data for the initial conditions
get_extrn_lbcs	Cycle-specific task to obtain external data for the lateral boundary (LB) conditions
make_ics	Generate initial conditions from the external data
make_lbcs	Generate lateral boundary conditions from the external data
run_fcst	Run the forecast model (UFS weather model)
run_post	Run the post-processing tool (UPP)

4.8 Launch of Workflow

There are two ways to launch the workflow using Rocoto: (1) with the `launch_FV3LAM_wflow.sh` script, and (2) manually calling the `rocotorun` command. Moreover, you can run the workflow separately using stand-alone scripts.

An environment variable may be set to navigate to the `$EXPTDIR` more easily. If the login shell is bash, it can be set as follws:

```
export EXPTDIR=/path-to-experiment/directory
```

Or if the login shell is csh/tcsh, it can be set using:

```
setenv EXPTDIR /path-to-experiment/directory
```

4.8.1 Launch with the `launch_FV3LAM_wflow.sh` script

To launch the `launch_FV3LAM_wflow.sh` script, simply call it without any arguments as follows:

```
cd ${EXPTDIR}
./launch_FV3LAM_wflow.sh
```

This script creates a log file named `log.launch_FV3LAM_wflow` in the EXPTDIR directory (described in Section 3.3) or appends to it if it already exists. You can check the contents of the end of the log file (e.g. last 30 lines) using the command:

```
tail -n 30 log.launch_FV3LAM_wflow
```

This command will print out the status of the workflow tasks as follows:

```
CYCLE                    TASK                       JOBID        STATE    EXIT STATUS   TRIES␣
↪ DURATION
==========================================================================================================
202006170000        make_grid         druby://hfe01:33728   SUBMITTING              -       0␣
↪       0.0
202006170000        make_orog                           -            -              -       -␣
↪         -
202006170000   make_sfc_climo                           -            -              -       -␣
↪         -
202006170000    get_extrn_ics         druby://hfe01:33728   SUBMITTING              -       0␣
↪       0.0
202006170000   get_extrn_lbcs         druby://hfe01:33728   SUBMITTING              -       0␣
↪       0.0
202006170000         make_ics                           -            -              -       -␣
↪         -
202006170000        make_lbcs                           -            -              -       -␣
↪         -
202006170000         run_fcst                           -            -              -       -␣
↪         -
202006170000      run_post_00                           -            -              -       -␣
↪         -
202006170000      run_post_01                           -            -              -       -␣
↪         -
202006170000      run_post_02                           -            -              -       -␣
↪         -
202006170000      run_post_03                           -            -              -       -␣
↪         -
202006170000      run_post_04                           -            -              -       -␣
↪         -
202006170000      run_post_05                           -            -              -       -␣
↪         -
202006170000      run_post_06                           -            -              -       -␣
↪         -

Summary of workflow status:
~~~~~~~~~~~~~~~~~~~~~~~~~~

  0 out of 1 cycles completed.
  Workflow status:  IN PROGRESS
```

Error messages for each task can be found in the task log files located in the `EXPTDIR/log` directory. In order to launch more tasks in the workflow, you just need to call the launch script again as follows:

```
./launch_FV3LAM_wflow
```

If everything goes smoothly, you will eventually get the following workflow status table as follows:

```
CYCLE                    TASK                       JOBID        STATE    EXIT STATUS   TRIES␣
↪ DURATION
==========================================================================================================
202006170000        make_grid                     8854765    SUCCEEDED              0       1␣
↪       6.0
```

(continues on next page)

(continued from previous page)

```
202006170000        make_orog                    8854809    SUCCEEDED                0        1␣
↪      27.0
202006170000   make_sfc_climo                    8854849    SUCCEEDED                0        1␣
↪      36.0
202006170000    get_extrn_ics                    8854763    SUCCEEDED                0        1␣
↪      54.0
202006170000   get_extrn_lbcs                    8854764    SUCCEEDED                0        1␣
↪      61.0
202006170000         make_ics                    8854914    SUCCEEDED                0        1␣
↪     119.0
202006170000        make_lbcs                    8854913    SUCCEEDED                0        1␣
↪      98.0
202006170000         run_fcst                    8854992    SUCCEEDED                0        1␣
↪     655.0
202006170000      run_post_00                    8855459    SUCCEEDED                0        1␣
↪       6.0
202006170000      run_post_01                    8855460    SUCCEEDED                0        1␣
↪       6.0
202006170000      run_post_02                    8855461    SUCCEEDED                0        1␣
↪       6.0
202006170000      run_post_03                    8855462    SUCCEEDED                0        1␣
↪       6.0
202006170000      run_post_04                    8855463    SUCCEEDED                0        1␣
↪       6.0
202006170000      run_post_05                    8855464    SUCCEEDED                0        1␣
↪       6.0
202006170000      run_post_06                    8855465    SUCCEEDED                0        1␣
↪       6.0
```

If all the tasks complete successfully, the workflow status in the log file will include the word "SUCCESS." Otherwise, the workflow status will include the word "FAILURE."

4.8.2 Manually launch by calling the `rocotorun` command

To launch the workflow manually, the `rocoto` module should be loaded:

```
module load rocoto
```

Then, launch the workflow as follows:

```
cd ${EXPTDIR}
rocotorun -w FV3LAM_wflow.xml -d FV3LAM_wflow.db -v 10
```

To check the status of the workflow, issue a `rocotostat` command as follows:

```
rocotostat -w FV3LAM_wflow.xml -d FV3LAM_wflow.db -v 10
```

Wait a few seconds and issue a second set of `rocotorun` and `rocotostat` commands:

```
rocotorun -w FV3LAM_wflow.xml -d FV3LAM_wflow.db -v 10
rocotostat -w FV3LAM_wflow.xml -d FV3LAM_wflow.db -v 10
```

4.8.3 Run the Workflow Using the Stand-alone Scripts

The regional workflow has the capability to be run using standalone shell scripts if the Rocoto software is not available on a given platform. These scripts are located in the `ufs-srweather-app/regional_workflow/ush/wrappers` directory. Each workflow task has a wrapper script to set environment variables and run the job script.

Example batch-submit scripts for Hera (Slurm) and Cheyenne (PBS) are included: `sq_job.sh` and `qsub_job.sh`, respectively. These examples set the build and run environment for Hera or Cheyenne so that run-time libraries match the compiled libraries (i.e. netcdf, mpi). Users may either modify the submit batch script as each task is submitted, or duplicate this batch wrapper for their system settings for each task. Alternatively, some batch systems allow users to specify most of the settings on the command line (with the `sbatch` or `qsub` command, for example). This piece will be unique to your platform. The tasks run by the regional workflow are shown in Table 4.7. Tasks with the same stage level may be run concurrently (no dependency).

Table 4.7: List of tasks in the regional workflow in the order that they are executed. Scripts with the same stage number may be run simultaneously. The number of processors and wall clock time is a good starting point for Cheyenne or Hera when running a 48-h forecast on the 25-km CONUS domain.

Stage/ step	Task Run Script	Number of Processors	Wall clock time (H:MM)
1	run_get_ics.sh	1	0:20 (depends on HPSS vs FTP vs staged-on-disk)
1	run_get_lbcs.sh	1	0:20 (depends on HPSS vs FTP vs staged-on-disk)
1	run_make_grid.sh	24	0:20
2	run_make_orog.sh	24	0:20
3	run_make_sfc_climo.sh	48	0:20
4	run_make_ics.sh	48	0:30
4	run_make_lbcs.sh	48	0:30
5	run_fcst.sh	48	0:30
6	run_post.sh	48	0:25 (2 min per output forecast hour)

The steps to run the standalone scripts are as follows:

1. Clone and build the ufs-srweather-app following the steps here, or in Sections 4.1 to Section 4.6 above.
2. Generate an experiment configuration following the steps here, or in Section 4.7 above.
3. `cd` into the experiment directory
4. Set the environment variable `EXPTDIR` for either csh and bash, respectively:

```
setenv EXPTDIR `pwd`
export EXPTDIR=`pwd`
```

5. COPY the wrapper scripts from the regional_workflow directory into your experiment directory:

```
cp ufs-srweather-app/regional_workflow/ush/wrappers/* .
```

6. RUN each of the listed scripts in order. Scripts with the same stage number may be run simultaneously.
    1. On most HPC systems, you will need to submit a batch job to run multi-processor jobs.
    2. On some HPC systems, you may be able to run the first two jobs (serial) on a login node/command-line
    3. Example scripts for Slurm (Hera) and PBS (Cheyenne) are provided. These will need to be adapted to your system.
    4. This submit batch script is hard-coded per task, so will need to be modified or copied to run each task.

Check the batch script output file in your experiment directory for a "SUCCESS" message near the end of the file.

CHAPTER

FIVE

CONFIGURING THE WORKFLOW: CONFIG.SH AND CONFIG_DEFAULTS.SH

To create the experiment directory and workflow when running the SRW App, the user must create an experiment configuration file named `config.sh`. This file contains experiment-specific information, such as dates, external model data, directories, and other relevant settings. To help the user, two sample configuration files have been included in the `regional_workflow` repository's `ush` directory: `config.community.sh` and `config.nco.sh`. The first is for running experiments in community mode (`RUN_ENVIR` set to "community"; see below), and the second is for running experiments in "nco" mode (`RUN_ENVIR` set to "nco"). Note that for this release, only "community" mode is supported. These files can be used as the starting point from which to generate a variety of experiment configurations in which to run the SRW App.

There is an extensive list of experiment parameters that a user can set when configuring the experiment. Not all of these need to be explicitly set by the user in `config.sh`. In the case that a user does not define an entry in the `config.sh` script, either its value in `config_defaults.sh` will be used, or it will be reset depending on other parameters, e.g. the platform on which the experiment will be run (specified by `MACHINE`). Note that `config_defaults.sh` contains the full list of experiment parameters that a user may set in `config.sh` (i.e. the user cannot set parameters in config.sh that are not initialized in `config_defaults.sh`).

The following is a list of the parameters in the `config_defaults.sh` file. For each parameter, the default value and a brief description is given. In addition, any relevant information on features and settings supported or unsupported in this release is specified.

5.1 Platform Environment

`RUN_ENVIR`: (Default: "nco") This variable determines the mode that the workflow will run in. The user can choose between two modes: "nco" and "community." The "nco" mode uses a directory structure that mimics what is used in operations at NOAA/NCEP Central Operations (NCO) and by those in the NOAA/NCEP/Environmental Modeling Center (EMC) working with NCO on pre-implementation testing. Specifics of the conventions used in "nco" mode can be found in the following WCOSS Implementation Standards document:

NCEP Central Operations
WCOSS Implementation Standards
April 17, 2019

Version 10.2.0

Setting RUN_ENVIR to "community" will use the standard directory structure and variable naming convention and is recommended in most cases for users who are not planning to implement their code into operations at NCO.

MACHINE: (Default: "BIG_COMPUTER") The machine (a.k.a. platform) on which the workflow will run. Currently supported platforms include "WCOSS_CRAY," "WCOSS_DELL_P3," "HERA," "ORION," "JET," "ODIN," "CHEYENNE," "STAMPEDE," "GAEA," "MACOS," and "LINUX."

ACCOUNT: (Default: "project_name") The account under which to submit jobs to the queue on the specified MACHINE.

WORKFLOW_MANAGER: (Default: "none") The workflow manager to use (e.g. "ROCOTO"). This is set to "none" by default, but if the machine name is set to a platform that supports Rocoto, this will be overwritten and set to "ROCOTO."

SCHED: (Default: "") The job scheduler to use (e.g. slurm) on the specified MACHINE. Set this to an empty string in order for the experiment generation script to set it automatically depending on the machine the workflow is running on. Currently, supported schedulers include "slurm," "pbspro," "lsf," "lsfcray," and "none".

PARTITION_DEFAULT: (Default: "") If using the slurm job scheduler (i.e. if SCHED is set to "slurm"), the default partition to which to submit workflow tasks. If a task does not have a specific variable that specifies the partition to which it will be submitted (e.g. PARTITION_HPSS, PARTITION_FCST; see below), it will be submitted to the partition specified by this variable. If this is not set or is set to an empty string, it will be (re)set to a machine-dependent value. This is not used if SCHED is not set to "slurm."

CLUSTERS_DEFAULT: (Default: "") If using the slurm job scheduler (i.e. if SCHED is set to "slurm"), the default clusters to which to submit workflow tasks. If a task does not have a specific variable that specifies the partition to which it will be submitted (e.g. CLUSTERS_HPSS, CLUSTERS_FCST; see below), it will be submitted to the clusters specified by this variable. If this is not set or is set to an empty string, it will be (re)set to a machine-dependent value. This is not used if SCHED is not set to "slurm."

QUEUE_DEFAULT: (Default: "") The default queue or QOS (if using the slurm job scheduler, where QOS is Quality of Service) to which workflow tasks are submitted. If a task does not have a specific variable that specifies the queue to which it will be submitted (e.g. QUEUE_HPSS, QUEUE_FCST; see below), it will be submitted to the queue specified by this variable. If this is not set or is set to an empty string, it will be (re)set to a machine-dependent value.

PARTITION_HPSS: (Default: "") If using the slurm job scheduler (i.e. if SCHED is set to "slurm"), the partition to which the tasks that get or create links to external model files [which are needed to generate initial conditions (ICs) and lateral boundary conditions (LBCs)] are submitted. If this is not set or is set to an empty string, it will be (re)set to a machine-dependent value. This is not used if SCHED is not set to "slurm."

CLUSTERS_HPSS: (Default: "") If using the slurm job scheduler (i.e. if SCHED is set to "slurm"), the clusters to which the tasks that get or create links to external model files [which are needed

to generate initial conditions (ICs) and lateral boundary conditions (LBCs)] are submitted. If this is not set or is set to an empty string, it will be (re)set to a machine-dependent value. This is not used if SCHED is not set to "slurm."

QUEUE_HPSS: (Default: "") The queue or QOS to which the tasks that get or create links to external model files are submitted. If this is not set or is set to an empty string, it will be (re)set to a machine-dependent value.

PARTITION_FCST: (Default: "") If using the slurm job scheduler (i.e. if SCHED is set to "slurm"), the partition to which the task that runs forecasts is submitted. If this is not set or set to an empty string, it will be (re)set to a machine-dependent value. This is not used if SCHED is not set to "slurm."

CLUSTERS_FCST: (Default: "") If using the slurm job scheduler (i.e. if SCHED is set to "slurm"), the clusters to which the task that runs forecasts is submitted. If this is not set or set to an empty string, it will be (re)set to a machine-dependent value. This is not used if SCHED is not set to "slurm."

QUEUE_FCST: (Default: "") The queue or QOS to which the task that runs a forecast is submitted. If this is not set or set to an empty string, it will be (re)set to a machine-dependent value.

5.2 Parameters for Running Without a Workflow Manager

These settings control run commands for platforms without a workflow manager. Values will be ignored unless WORKFLOW_MANAGER="none".

RUN_CMD_UTILS: (Default: "mpirun -np 1") The run command for pre-processing utilities (shave, orog, sfc_climo_gen, etc.). This can be left blank for smaller domains, in which case the executables will run without MPI.

RUN_CMD_FCST: (Default: "mpirun -np ${PE_MEMBER01}") The run command for the model forecast step. This will be appended to the end of the variable definitions file ("var_defns.sh").

RUN_CMD_POST: (Default: "mpirun -np 1") The run command for post-processing (UPP). Can be left blank for smaller domains, in which case UPP will run without MPI.

5.3 Cron-Associated Parameters

USE_CRON_TO_RELAUNCH: (Default: "FALSE") Flag that determines whether or not a line is added to the user's cron table that calls the experiment launch script every CRON_RELAUNCH_INTVL_MNTS minutes.

CRON_RELAUNCH_INTVL_MNTS: (Default: "03") The interval (in minutes) between successive calls of the experiment launch script by a cron job to (re)launch the experiment (so that the workflow for the experiment kicks off where it left off). This is used only if USE_CRON_TO_RELAUNCH is set to "TRUE".

5.4 Directory Parameters

EXPT_BASEDIR: (Default: "") The base directory in which the experiment directory will be created. If this is not specified or if it is set to an empty string, it will default to `${HOMErrfs}/../../expt_dirs`, where `${HOMErrfs}` contains the full path to the `regional_workflow` directory.

EXPT_SUBDIR: (Default: "") The name that the experiment directory (without the full path) will have. The full path to the experiment directory, which will be contained in the variable `EXPTDIR`, will be:

```
EXPTDIR="${EXPT_BASEDIR}/${EXPT_SUBDIR}"
```

This parameter cannot be left as a null string.

5.5 NCO Mode Parameters

These variables apply only when using NCO mode (i.e. when `RUN_ENVIR` is set to "nco").

COMINgfs: (Default: "/base/path/of/directory/containing/gfs/input/files") The beginning portion of the directory which contains files generated by the external model that the initial and lateral boundary condition generation tasks need in order to create initial and boundary condition files for a given cycle on the native FV3-LAM grid. For a cycle that starts on the date specified by the variable YYYYMMDD (consisting of the 4-digit year followed by the 2-digit month followed by the 2-digit day of the month) and hour specified by the variable HH (consisting of the 2-digit hour-of-day), the directory in which the workflow will look for the external model files is:

```
$COMINgfs/gfs.$yyyymmdd/$hh
```

STMP: (Default: "/base/path/of/directory/containing/model/input/and/raw/output/files") The beginning portion of the directory that will contain cycle-dependent model input files, symlinks to cycle-independent input files, and raw (i.e. before post-processing) forecast output files for a given cycle. For a cycle that starts on the date specified by YYYYMMDD and hour specified by HH (where YYYYMMDD and HH are as described above) [so that the cycle date (cdate) is given by `cdate="${YYYYMMDD}${HH}"`], the directory in which the aforementioned files will be located is:

```
$STMP/tmpnwprd/$RUN/$cdate
```

NET, envir, RUN: Variables used in forming the path to the directory that will contain the output files from the post-processor (UPP) for a given cycle (see definition of `PTMP` below). These are defined in the WCOSS Implementation Standards document as follows:

NET: (Default: "rrfs") Model name (first level of com directory structure)

envir: (Default: "para") Set to "test" during the initial testing phase, "para" when running in parallel (on a schedule), and "prod" in production.

RUN: (Default: "experiment_name") Name of model run (third level of com directory structure).

PTMP: (Default: "/base/path/of/directory/containing/postprocessed/output/files") The beginning portion of the directory that will contain the output files from the post-processor (UPP) for a given cycle. For a cycle that starts on the date specified by YYYYMMDD and hour specified by HH (where YYYYMMDD and HH are as described above), the directory in which the UPP output files will be placed will be:

```
$PTMP/com/$NET/$envir/$RUN.$yyyymmdd/$hh
```

5.6 Pre-Processing File Separator Parameters

DOT_OR_USCORE: (Default: "_") This variable sets the separator character(s) to use in the names of the grid, mosaic, and orography fixed files. Ideally, the same separator should be used in the names of these fixed files as the surface climatology fixed files.

5.7 File Name Parameters

EXPT_CONFIG_FN: (Default: "config.sh") Name of the user-specified configuration file for the forecast experiment.

RGNL_GRID_NML_FN: (Default: "regional_grid.nml") Name of the file containing Fortran namelist settings for the code that generates an "ESGgrid" type of regional grid.

FV3_NML_BASE_SUITE_FN: (Default: "input.nml.FV3") Name of the Fortran namelist file containing the forecast model's base suite namelist, i.e. the portion of the namelist that is common to all physics suites.

FV3_NML_YAML_CONFIG_FN: (Default: "FV3.input.yml") Name of YAML configuration file containing the forecast model's namelist settings for various physics suites.

DIAG_TABLE_FN: (Default: "diag_table") Name of the file that specifies the fields that the forecast model will output.

FIELD_TABLE_FN: (Default: "field_table") Name of the file that specifies the tracers that the forecast model will read in from the IC/LBC files.

DATA_TABLE_FN: (Default: "data_table") The name of the file containing the data table read in by the forecast model.

MODEL_CONFIG_FN: (Default: "model_configure") The name of the file containing settings and configurations for the NUOPC/ESMF component.

NEMS_CONFIG_FN: (Default: "nems.configure") The name of the file containing information about the various NEMS components and their run sequence.

FV3_EXEC_FN: (Default: "NEMS.exe") Name of the forecast model executable in the executables directory (`EXECDIR`; set during experiment generation).

WFLOW_XML_FN: (Default: "FV3LAM_wflow.xml") Name of the Rocoto workflow XML file that the experiment generation script creates and that defines the workflow for the experiment.

GLOBAL_VAR_DEFNS_FN: (Default: "var_defns.sh") Name of the file (a shell script) containing the definitions of the primary experiment variables (parameters) defined in this default configuration script and in config.sh as well as secondary experiment variables generated by the experiment generation script. This file is sourced by many scripts (e.g. the J-job scripts corresponding to each workflow task) in order to make all the experiment variables available in those scripts.

EXTRN_MDL_ICS_VAR_DEFNS_FN: (Default: "extrn_mdl_ics_var_defns.sh") Name of the file (a shell script) containing the definitions of variables associated with the external model from which ICs are generated. This file is created by the GET_EXTRN_ICS_TN task because the values of the variables it contains are not known before this task runs. The file is then sourced by the MAKE_ICS_TN task.

EXTRN_MDL_LBCS_VAR_DEFNS_FN: (Default: "extrn_mdl_lbcs_var_defns.sh") Name of the file (a shell script) containing the definitions of variables associated with the external model from which LBCs are generated. This file is created by the GET_EXTRN_LBCS_TN task because the values of the variables it contains are not known before this task runs. The file is then sourced by the MAKE_ICS_TN task.

WFLOW_LAUNCH_SCRIPT_FN: (Default: "launch_FV3LAM_wflow.sh") Name of the script that can be used to (re)launch the experiment's Rocoto workflow.

WFLOW_LAUNCH_LOG_FN: (Default: "log.launch_FV3LAM_wflow") Name of the log file that contains the output from successive calls to the workflow launch script (WFLOW_LAUNCH_SCRIPT_FN).

5.8 Foreast Parameters

DATE_FIRST_CYCL: (Default: "YYYYMMDD") Starting date of the first forecast in the set of forecasts to run. Format is "YYYYMMDD". Note that this does not include the hour-of-day.

DATE_LAST_CYCL: (Default: "YYYYMMDD") Starting date of the last forecast in the set of forecasts to run. Format is "YYYYMMDD". Note that this does not include the hour-of-day.

CYCL_HRS: (Default: ("HH1" "HH2")) An array containing the hours of the day at which to launch forecasts. Forecasts are launched at these hours on each day from DATE_FIRST_CYCL to DATE_LAST_CYCL, inclusive. Each element of this array must be a two-digit string representing an integer that is less than or equal to 23, e.g. "00", "03", "12", "23".

FCST_LEN_HRS: (Default: "24") The length of each forecast, in integer hours.

5.9 Initial and Lateral Boundary Condition Generation Parameters

EXTRN_MDL_NAME_ICS: (Default: "FV3GFS") The name of the external model that will provide fields from which initial condition (IC) files, surface files, and 0-th hour boundary condition files will be generated for input into the forecast model.

EXTRN_MDL_NAME_LBCS: (Default: "FV3GFS") The name of the external model that will provide fields from which lateral boundary condition (LBC) files (except for the 0-th hour LBC file) will be generated for input into the forecast model.

LBC_SPEC_INTVL_HRS: (Default: "6") The interval (in integer hours) at which LBC files will be generated, referred to as the boundary specification interval. Note that the model specified in EXTRN_MDL_NAME_LBCS must have data available at a frequency greater than or equal to that implied by LBC_SPEC_INTVL_HRS. For example, if LBC_SPEC_INTVL_HRS is set to 6, then the model must have data available at least every 6 hours. It is up to the user to ensure that this is the case.

FV3GFS_FILE_FMT_ICS: (Default: "nemsio") If using the FV3GFS model as the source of the ICs (i.e. if EXTRN_MDL_NAME_ICS is set to "FV3GFS"), this variable specifies the format of the model files to use when generating the ICs.

FV3GFS_FILE_FMT_LBCS: (Default: "nemsio") If using the FV3GFS model as the source of the LBCs (i.e. if EXTRN_MDL_NAME_LBCS is set to "FV3GFS"), this variable specifies the format of the model files to use when generating the LBCs.

5.10 User-Staged External Model Directory and File Parameters

USE_USER_STAGED_EXTRN_FILES: (Default: "False") Flag that determines whether or not the workflow will look for the external model files needed for generating ICs and LBCs in user-specified directories (as opposed to fetching them from mass storage like NOAA HPSS).

EXTRN_MDL_SOURCE_BASEDIR_ICS: (Default: "/base/dir/containing/user/staged/extrn/mdl/files/for/ICs") Directory in which to look for external model files for generating ICs. If USE_USER_STAGED_EXTRN_FILES is set to "TRUE", the workflow looks in this directory (specifically, in a subdirectory under this directory named "YYYYMMDDHH" consisting of the starting date and cycle hour of the forecast, where YYYY is the 4-digit year, MM the 2-digit month, DD the 2-digit day of the month, and HH the 2-digit hour of the day) for the external model files specified by the array EXTRN_MDL_FILES_ICS (these files will be used to generate the ICs on the native FV3-LAM grid). This variable is not used if USE_USER_STAGED_EXTRN_FILES is set to "FALSE".

EXTRN_MDL_FILES_ICS: (Default: "ICS_file1" "ICS_file2" "...") Array containing the names of the files to search for in the directory specified by EXTRN_MDL_SOURCE_BASEDIR_ICS. This variable is not used if USE_USER_STAGED_EXTRN_FILES is set to "FALSE".

EXTRN_MDL_SOURCE_BASEDIR_LBCS: (Default: "/base/dir/containing/user/staged/extrn/mdl/files/for/ICs") Analogous to EXTRN_MDL_SOURCE_BASEDIR_ICS but for LBCs instead of ICs.

EXTRN_MDL_FILES_LBCS: (Default: " "LBCS_file1" "LBCS_file2" "...") Analogous to EXTRN_MDL_FILES_ICS but for LBCs instead of ICs.

5.11 CCPP Parameter

CCPP_PHYS_SUITE: (Default: "FV3_GFS_v15p2") The CCPP (Common Community Physics Package) physics suite to use for the forecast(s). The choice of physics suite determines the forecast model's namelist file, the diagnostics table file, the field table file, and the XML physics suite definition file that are staged in the experiment directory or the cycle directories under it. Current supported settings for this parameter are "FV3_GFS_v15p2" and "FV3_RRFS_v1alpha".

5.12 Grid Generation Parameters

GRID_GEN_METHOD: (Default: "") This variable specifies the method to use to generate a regional grid in the horizontal. The only supported value of this parameter is "ESGgrid", in which case the Extended Schmidt Gnomonic grid generation method developed by Jim Purser(1) of EMC will be used.

(1)Purser, R. J., D. Jovic, G. Ketefian, T. Black, J. Beck, J. Dong, and J. Carley, 2020: The Extended Schmidt Gnomonic Grid for Regional Applications. Unified Forecast System (UFS) Users' Workshop. July 27-29, 2020.

Note:

1. If the experiment is using one of the predefined grids (i.e. if PREDEF_GRID_NAME is set to the name of one of the valid predefined grids), then GRID_GEN_METHOD will be reset to the value of GRID_GEN_METHOD for that grid. This will happen regardless of whether or not GRID_GEN_METHOD is assigned a value in the user-specified experiment configuration file, i.e. any value it may be assigned in the experiment configuration file will be overwritten.
2. If the experiment is not using one of the predefined grids (i.e. if PREDEF_GRID_NAME is set to a null string), then GRID_GEN_METHOD must be set in the experiment configuration file. Otherwise, it will remain set to a null string, and the experiment generation will fail because the generation scripts check to ensure that it is set to a non-empty string before creating the experiment directory.

The following parameters must be set if using the "ESGgrid" method of generating a regional grid (i.e. for GRID_GEN_METHOD set to "ESGgrid").

ESGgrid_LON_CTR: (Default: "") The longitude of the center of the grid (in degrees).

ESGgrid_LAT_CTR: (Default: "") The latitude of the center of the grid (in degrees).

ESGgrid_DELX: (Default: "") The cell size in the zonal direction of the regional grid (in meters).

ESGgrid_DELY: (Default: "") The cell size in the meridional direction of the regional grid (in meters).

ESGgrid_NX: (Default: "") The number of cells in the zonal direction on the regional grid.

ESGgrid_NY: (Default: "") The number of cells in the meridional direction on the regional grid.

ESGgrid_WIDE_HALO_WIDTH: (Default: "") The width (in units of number of grid cells) of the halo to add around the regional grid before shaving the halo down to the width(s) expected by the forecast model.

In order to generate grid files containing halos that are 3-cell and 4-cell wide and orography files with halos that are 0-cell and 3-cell wide (all of which are required as inputs to the forecast model), the grid and orography tasks first create files with halos around the regional domain of width ESGgrid_WIDE_HALO_WIDTH cells. These are first stored in files. The files are then read in and "shaved" down to obtain grid files with 3-cell-wide and 4-cell-wide halos and orography files with 0-cell-wide (i.e. no halo) and 3-cell-wide halos. For this reason, we refer to the original halo that then gets shaved down as the "wide" halo, i.e. because it is wider than the 0-cell-wide, 3-cell-wide, and 4-cell-wide halos that we will eventually end up with. Note that the grid and orography files with the wide halo are only needed as intermediates in generating the files with 0-cell-, 3-cell-, and 4-cell-wide halos; they are not needed by the forecast model.

5.13 Computational Forecast Parameters

DT_ATMOS: (Default: "") The main forecast model integration time step. As described in the forecast model documentation, "It corresponds to the frequency with which the top level routine in the dynamics is called as well as the frequency with which the physics is called."

LAYOUT_X, LAYOUT_Y: (Default: "") The number of MPI tasks (processes) to use in the two horizontal directions (x and y) of the regional grid when running the forecast model.

BLOCKSIZE: (Default: "") The amount of data that is passed into the cache at a time.

Here, we set these parameters to null strings. This is so that, for any one of these parameters:

1. If the experiment is using a predefined grid and the user sets the parameter in the user-specified experiment configuration file (EXPT_CONFIG_FN), that value will be used in the forecast(s). Otherwise, the default value of the parameter for that predefined grid will be used.
2. If the experiment is not using a predefined grid (i.e. it is using a custom grid whose parameters are specified in the experiment configuration file), then the user must specify a value for the parameter in that configuration file. Otherwise, the parameter will remain set to a null string, and the experiment generation will fail, because the generation scripts check to ensure that all the parameters defined in this section are set to non-empty strings before creating the experiment directory.

5.14 Write-Component (Quilting) Parameters

QUILTING: (Default: "TRUE") Flag that determines whether or not to use the write-component for writing forecast output files to disk. If set to "TRUE", the forecast model will output files named dynf$HHH.nc and phyf$HHH.nc (where HHH is the 3-hour output forecast hour) containing dynamics and physics fields, respectively, on the write-component grid (the regridding from the native FV3-LAM grid to the write-component grid is done by the forecast model). If QUILTING is set to "FALSE", then the output file names are fv3_history.nc and fv3_history2d.nc and contain fields on the native grid. Note that if QUILTING is set to

"FALSE", then the RUN_POST_TN (meta)task cannot be run because the Unified Post Processor (UPP) code that this task calls cannot process fields on the native grid. In that case, the RUN_POST_TN (meta)task will be automatically removed from the Rocoto workflow XML.

PRINT_ESMF: (Default: "FALSE") Flag for whether or not to output extra (debugging) information from ESMF routines. Must be "TRUE" or "FALSE". Note that the write-component uses ESMF library routines to interpolate from the native forecast model grid to the user-specified output grid (which is defined in the model configuration file (model_configure) in the forecast run directory).

WRTCMP_write_groups: (Default: "1") The number of write groups (i.e. groups of MPI tasks) to use in the write-component.

WRTCMP_write_tasks_per_group: (Default: "20") The number of MPI tasks to allocate for each write group.

5.15 Predefined Grid Parameters

PREDEF_GRID_NAME: (Default: "") This parameter specifies the name of a predefined regional grid.

Note:

- If PREDEF_GRID_NAME is set to a valid predefined grid name, the grid generation method GRID_GEN_METHOD, the (native) grid parameters, and the write-component grid parameters are set to predefined values for the specified grid, overwriting any settings of these parameters in the user-specified experiment configuration file (config.sh). In addition, if the time step DT_ATMOS and the computational parameters LAYOUT_X, LAYOUT_Y, and BLOCKSIZE are not specified in that configuration file, they are also set to predefined values for the specified grid.
- If PREDEF_GRID_NAME is set to an empty string, it implies the user is providing the native grid parameters in the user-specified experiment configuration file (EXPT_CONFIG_FN). In this case, the grid generation method GRID_GEN_METHOD, the native grid parameters, and the write-component grid parameters as well as the main time step (DT_ATMOS) and the computational parameters LAYOUT_X, LAYOUT_Y, and BLOCKSIZE must be set in that configuration file.

Setting PREDEF_GRID_NAME provides a convenient method of specifying a commonly used set of grid-dependent parameters. The predefined grid parameters are specified in the script

```
ush/set_predef_grid_params.sh
```

Currently supported PREDEF_GRID_NAME options are "RRFS_CONUS_25km," "RRFS_CONUS_13km," and "RRFS_CONUS_3km."

5.16 Pre-existing Directory Parameter

PREEXISTING_DIR_METHOD: (Default: "delete") This variable determines the method to deal with pre-existing directories [e.g ones generated by previous calls to the experiment generation script using the same experiment name (EXPT_SUBDIR) as the current experiment]. This variable must be set to one of "delete", "rename", and "quit". The resulting behavior for each of these values is as follows:

- "delete": The preexisting directory is deleted and a new directory (having the same name as the original preexisting directory) is created.
- "rename": The preexisting directory is renamed and a new directory (having the same name as the original pre-existing directory) is created. The new name of the preexisting directory consists of its original name and the suffix "_oldNNN", where NNN is a 3-digit integer chosen to make the new name unique.
- "quit": The preexisting directory is left unchanged, but execution of the currently running script is terminated. In this case, the preexisting directory must be dealt with manually before rerunning the script.

5.17 Verbose Parameter

VERBOSE: (Default: "TRUE") This is a flag that determines whether or not the experiment generation and workflow task scripts print out extra informational messages.

5.18 Pre-Processing Parameters

These parameters set flags (and related directories) that determine whether the grid, orography, and/or surface climatology file generation tasks should be run. Note that these are all cycle-independent tasks, i.e. if they are to be run, they do so only once at the beginning of the workflow before any cycles are run.

RUN_TASK_MAKE_GRID: (Default: "TRUE") Flag that determines whether the grid file generation task (MAKE_GRID_TN) is to be run. If this is set to "TRUE", the grid generation task is run and new grid files are generated. If it is set to "FALSE", then the scripts look for pre-generated grid files in the directory specified by GRID_DIR (see below).

GRID_DIR: (Default: "/path/to/pregenerated/grid/files") The directory in which to look for pre-generated grid files if RUN_TASK_MAKE_GRID is set to "FALSE".

RUN_TASK_MAKE_OROG: (Default: "TRUE") Same as RUN_TASK_MAKE_GRID but for the orography generation task (MAKE_OROG_TN).

OROG_DIR: (Default: "/path/to/pregenerated/orog/files") Same as GRID_DIR but for the orography generation task.

RUN_TASK_MAKE_SFC_CLIMO: (Default: "TRUE") Same as RUN_TASK_MAKE_GRID but for the surface climatology generation task (MAKE_SFC_CLIMO_TN).

SFC_CLIMO_DIR: (Default: "/path/to/pregenerated/surface/climo/files") Same as GRID_DIR but for the surface climatology generation task.

5.19 Surface Climatology Parameter

SFC_CLIMO_FIELDS: (Default: "("facsf" "maximum_snow_albedo" "slope_type" "snowfree_albedo" "soil_type
Array containing the names of all the fields for which the MAKE_SFC_CLIMO_TN task generates files on the native FV3-LAM grid.

5.20 Fixed File Parameters

Set parameters associated with the fixed (i.e. static) files. For the main NOAA HPC platforms, as well as Cheyenne, Odin, and Stampede, fixed files are prestaged with paths defined in the setup.sh script.

FIXgsm: (Default: "") System directory in which the majority of fixed (i.e. time-independent) files that are needed to run the FV3-LAM model are located.

TOPO_DIR: (Default: "") The location on disk of the static input files used by the make_orog task (orog.x and shave.x). Can be the same as FIXgsm.

SFC_CLIMO_INPUT_DIR: (Default: "") The location on disk of the static surface climatology input fields, used by sfc_climo_gen. These files are only used if RUN_TASK_MAKE_SFC_CLIMO=TRUE.

FNGLAC, ..., FNMSKH: (Default: see below)

```
(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"
 FNAISC="CFSR.SEAICE.1982.2012.monthly.clim.grb"
 FNSMCC="global_soilmgldas.t126.384.190.grb"
 FNMSKH="seaice_newland.grb")
```

Names of (some of the) global data files that are assumed to exist in a system directory specified (this directory is machine-dependent; the experiment generation scripts will set it and store it in the variable FIXgsm). These file names also appear directly in the forecast model's input namelist file.

FIXgsm_FILES_TO_COPY_TO_FIXam: (Default: see below)

```
("$FNGLAC" \
 "$FNMXIC" \
 "$FNTSFC" \
 "$FNSNOC" \
 "$FNAISC" \
 "$FNSMCC" \
 "$FNMSKH" \
```

(continues on next page)

(continued from previous page)

```
 "global_climaeropac_global.txt" \
 "fix_co2_proj/global_co2historicaldata_2010.txt" \
 "fix_co2_proj/global_co2historicaldata_2011.txt" \
 "fix_co2_proj/global_co2historicaldata_2012.txt" \
 "fix_co2_proj/global_co2historicaldata_2013.txt" \
 "fix_co2_proj/global_co2historicaldata_2014.txt" \
 "fix_co2_proj/global_co2historicaldata_2015.txt" \
 "fix_co2_proj/global_co2historicaldata_2016.txt" \
 "fix_co2_proj/global_co2historicaldata_2017.txt" \
 "fix_co2_proj/global_co2historicaldata_2018.txt" \
 "global_co2historicaldata_glob.txt" \
 "co2monthlycyc.txt" \
 "global_h2o_pltc.f77" \
 "global_hyblev.l65.txt" \
 "global_zorclim.1x1.grb" \
 "global_sfc_emissivity_idx.txt" \
 "global_solarconstant_noaa_an.txt" \
 "replace_with_FIXgsm_ozone_prodloss_filename")
```

If not running in NCO mode, this array contains the names of the files to copy from the `FIXgsm` system directory to the `FIXam` directory under the experiment directory. Note that the last element has a dummy value. This last element will get reset by the workflow generation scripts to the name of the ozone production/loss file to copy from `FIXgsm`. The name of this file depends on the ozone parameterization being used, and that in turn depends on the CCPP physics suite specified for the experiment. Thus, the CCPP physics suite XML must first be read in to determine the ozone parameterization and then the name of the ozone production/loss file. These steps are carried out elsewhere (in one of the workflow generation scripts/functions).

`FV3_NML_VARNAME_TO_FIXam_FILES_MAPPING`: (Default: see below)

```
("FNGLAC | $FNGLAC" \
 "FNMXIC | $FNMXIC" \
 "FNTSFC | $FNTSFC" \
 "FNSNOC | $FNSNOC" \
 "FNAISC | $FNAISC" \
 "FNSMCC | $FNSMCC" \
 "FNMSKH | $FNMSKH" )
```

This array is used to set some of the namelist variables in the forecast model's namelist file that represent the relative or absolute paths of various fixed files (the first column of the array, where columns are delineated by the pipe symbol "|") to the full paths to these files in the FIXam directory derived from the corresponding workflow variables containing file names (the second column of the array).

`FV3_NML_VARNAME_TO_SFC_CLIMO_FIELD_MAPPING`: (Default: see below)

```
("FNALBC  | snowfree_albedo" \
 "FNALBC2 | facsf" \
 "FNTG3C  | substrate_temperature" \
```

(continues on next page)

(continued from previous page)

```
 "FNVEGC  | vegetation_greenness" \
 "FNVETC  | vegetation_type" \
 "FNSOTC  | soil_type" \
 "FNVMNC  | vegetation_greenness" \
 "FNVMXC  | vegetation_greenness" \
 "FNSLPC  | slope_type" \
 "FNABSC  | maximum_snow_albedo" )
```

This array is used to set some of the namelist variables in the forecast model's namelist file that represent the relative or absolute paths of various fixed files (the first column of the array, where columns are delineated by the pipe symbol "|") to the full paths to surface climatology files (on the native FV3-LAM grid) in the `FIXLAM` directory derived from the corresponding surface climatology fields (the second column of the array).

`CYCLEDIR_LINKS_TO_FIXam_FILES_MAPPING`: (Default: see below)

```
("aerosol.dat                | global_climaeropac_global.txt" \
 "co2historicaldata_2010.txt | fix_co2_proj/global_co2historicaldata_2010.txt" \
 "co2historicaldata_2011.txt | fix_co2_proj/global_co2historicaldata_2011.txt" \
 "co2historicaldata_2012.txt | fix_co2_proj/global_co2historicaldata_2012.txt" \
 "co2historicaldata_2013.txt | fix_co2_proj/global_co2historicaldata_2013.txt" \
 "co2historicaldata_2014.txt | fix_co2_proj/global_co2historicaldata_2014.txt" \
 "co2historicaldata_2015.txt | fix_co2_proj/global_co2historicaldata_2015.txt" \
 "co2historicaldata_2016.txt | fix_co2_proj/global_co2historicaldata_2016.txt" \
 "co2historicaldata_2017.txt | fix_co2_proj/global_co2historicaldata_2017.txt" \
 "co2historicaldata_2018.txt | fix_co2_proj/global_co2historicaldata_2018.txt" \
 "co2historicaldata_glob.txt | global_co2historicaldata_glob.txt" \
 "co2monthlycyc.txt          | co2monthlycyc.txt" \
 "global_h2oprdlos.f77       | global_h2o_pltc.f77" \
 "global_zorclim.1x1.grb     | global_zorclim.1x1.grb" \
 "sfc_emissivity_idx.txt     | global_sfc_emissivity_idx.txt" \
 "solarconstant_noaa_an.txt  | global_solarconstant_noaa_an.txt" \
 "global_o3prdlos.f77        | " )
```

This array specifies the mapping to use between the symlinks that need to be created in each cycle directory (these are the "files" that FV3 looks for) and their targets in the `FIXam` directory. The first column of the array specifies the symlink to be created, and the second column specifies its target file in `FIXam` (where columns are delineated by the pipe symbol "|").

5.21 Workflow Task Parameters

These parameters set the names of the various workflow tasks and usually do not need to be changed. For each task, additional values set the parameters to pass to the job scheduler (e.g. slurm) that will submit a job for each task to be run. Parameters include the number of nodes to use to run the job, the number of MPI processes per node, the maximum walltime to allow for the job to complete, and the maximum number of times to attempt to run each task.

Task names:

MAKE_GRID_TN: (Default: “make_grid”)
MAKE_OROG_TN: (Default: “make_orog”)
MAKE_SFC_CLIMO_TN: (Default: “make_sfc_climo”)
GET_EXTRN_ICS_TN: (Default: “get_extrn_ics”)
GET_EXTRN_LBCS_TN: (Default: “get_extrn_lbcs”)
MAKE_ICS_TN: (Default: “make_ics”)
MAKE_LBCS_TN: (Default: “make_lbcs”)
RUN_FCST_TN: (Default: “run_fcst”)
RUN_POST_TN: (Default: “run_post”)

Number of nodes:

NODES_MAKE_GRID: (Default: “1”)
NODES_MAKE_OROG: (Default: “1”)
NODES_MAKE_SFC_CLIMO: (Default: “2”)
NODES_GET_EXTRN_ICS: (Default: “1”)
NODES_GET_EXTRN_LBCS: (Default: “1”)
NODES_MAKE_ICS: (Default: “4”)
NODES_MAKE_LBCS: (Default: “4”)
NODES_RUN_FCST: (Default: “”) # Calculated in the workflow generation scripts.
NODES_RUN_POST: (Default: “2”)

Number of MPI processes per node:

PPN_MAKE_GRID: (Default: “24”)
PPN_MAKE_OROG: (Default: “24”)
PPN_MAKE_SFC_CLIMO: (Default: “24”)
PPN_GET_EXTRN_ICS: (Default: “1”)
PPN_GET_EXTRN_LBCS: (Default: “1”)
PPN_MAKE_ICS: (Default: “12”)
PPN_MAKE_LBCS: (Default: “12”)
PPN_RUN_FCST: (Default: “24”) # Can be changed depending on the number of threads used.
PPN_RUN_POST: (Default: “24”)

Wall times:

TIME_MAKE_GRID: (Default: “00:20:00”)
TIME_MAKE_OROG: (Default: “00:20:00”)

TIME_MAKE_SFC_CLIMO: (Default: "00:20:00")
TIME_GET_EXTRN_ICS: (Default: "00:45:00")
TIME_GET_EXTRN_LBCS: (Default: "00:45:00")
TIME_MAKE_ICS: (Default: "00:30:00")
TIME_MAKE_LBCS: (Default: "00:30:00")
TIME_RUN_FCST: (Default: "04:30:00")
TIME_RUN_POST: (Default: "00:15:00")

Maximum number of attempts.

MAXTRIES_MAKE_GRID: (Default: "1")
MAXTRIES_MAKE_OROG: (Default: "1")
MAXTRIES_MAKE_SFC_CLIMO: (Default: "1")
MAXTRIES_GET_EXTRN_ICS: (Default: "1")
MAXTRIES_GET_EXTRN_LBCS: (Default: "1")
MAXTRIES_MAKE_ICS: (Default: "1")
MAXTRIES_MAKE_LBCS: (Default: "1")
MAXTRIES_RUN_FCST: (Default: "1")
MAXTRIES_RUN_POST: (Default: "1")

5.22 Customized Post Configuration Parameters

USE_CUSTOM_POST_CONFIG_FILE: (Default: "FALSE") Flag that determines whether a user-provided custom configuration file should be used for post-processing the model data. If this is set to "TRUE", then the workflow will use the custom post-processing (UPP) configuration file specified in CUSTOM_POST_CONFIG_FP. Otherwise, a default configuration file provided in the EMC_post repository will be used.

CUSTOM_POST_CONFIG_FP: (Default: "") The full path to the custom post flat file, including filename, to be used for post-processing. This is only used if CUSTOM_POST_CONFIG_FILE is set to "TRUE".

5.23 Halo Blend Parameter

HALO_BLEND: (Default: "10") Number of rows into the computational domain that should be blended with the LBCs. To shut halo blending off, set this to zero.

5.24 FVCOM Parameter

USE_FVCOM: (Default: "FALSE") Flag that specifies whether or not to update surface conditions in FV3-LAM with fields generated from the Finite Volume Community Ocean Model (FVCOM). If set to "TRUE", lake/sea surface temperatures, ice surface temperatures, and ice placement will be overwritten by data provided by FVCOM. This is done by running the executable `process_FVCOM.exe` in the `MAKE_ICS_TN` task to modify the file `sfc_data.nc` generated by `chgres_cube`. Note that the FVCOM data must already be interpolated to the desired FV3-LAM grid.

FVCOM_DIR: (Default: "/user/defined/dir/to/fvcom/data") User defined directory in which the file `fvcom.nc` containing FVCOM data on the FV3-LAM native grid is located. The file name in this directory must be `fvcom.nc`.

FVCOM_FILE: (Default: "fvcom.nc") Name of file located in `FVCOM_DIR` that has FVCOM data interpolated to FV3-LAM grid. This file will be copied later to a new location and the name changed to `fvcom.nc`.

5.25 Compiler Parameter

COMPILER: (Default: "intel") Type of compiler invoked during the build step. Currently, this must be set manually (i.e. it is not inherited from the build system in the `ufs-srweather-app` directory).

CHAPTER

SIX

LIMITED AREA MODEL (LAM) GRIDS: PREDEFINED AND USER-GENERATED OPTIONS

In order to set up the workflow and experiment generation of the UFS SRW App, the user must choose between three predefined FV3-LAM grids, or generate a user-defined grid. At this time, full support will only be provided to those using one of the three predefined grids supported in this release. However, preliminary information is provided at the end of this chapter that describes how users can leverage the SRW App workflow scripts to generate their own user-defined grid. This feature is not fully supported at this time and is 'use at your own risk'.

6.1 Predefined Grids

The UFS SRW App release includes three predefined LAM grids that users can choose from prior to generating a workflow/experiment configuration. To select a predefined grid, the `PREDEF_GRID_NAME` variable within the `config.sh` script needs to be set to one of the following three options:

- `RRFS_CONUS_3km`
- `RRFS_CONUS_13km`
- `RRFS_CONUS_25km`

The predefined grids are named after the prototype 3-km continental United States (CONUS) grid being tested for the Rapid Refresh Forecast System (RRFS), which will be a convection-allowing, hourly-cycled, FV3-LAM-based ensemble planned for operational implementation in 2024. To allow for use of High Resolution Rapid Refresh (HRRR) data to initialize the SRW App, all three supported grids were created to fit completely within the HRRR domain. Three resolution options were provided for flexibility related to compute resources and physics options. For example, a user may wish to use the 13-km or 25-km domain when running with the `FV3_GFS_v15p2` suite definition file (SDF), since that SDF uses cumulus physics that are not configured to run at 3-km. In addition, users will have much fewer computational constraints when running with the 13-km and 25-km domains.

The boundary of the `RRFS_CONUS_3km` domain is shown in Figure 6.1 (in red). Note that while it is possible to initialize the FV3-LAM with coarser external model data when using the `RRFS_CONUS_3km` domain, it is generally advised to use external model data that has a resolution similar to that of the native FV3-LAM (predefined) grid. In addition, this grid is ideal for running the `FV3_RRFS_v1alpha`

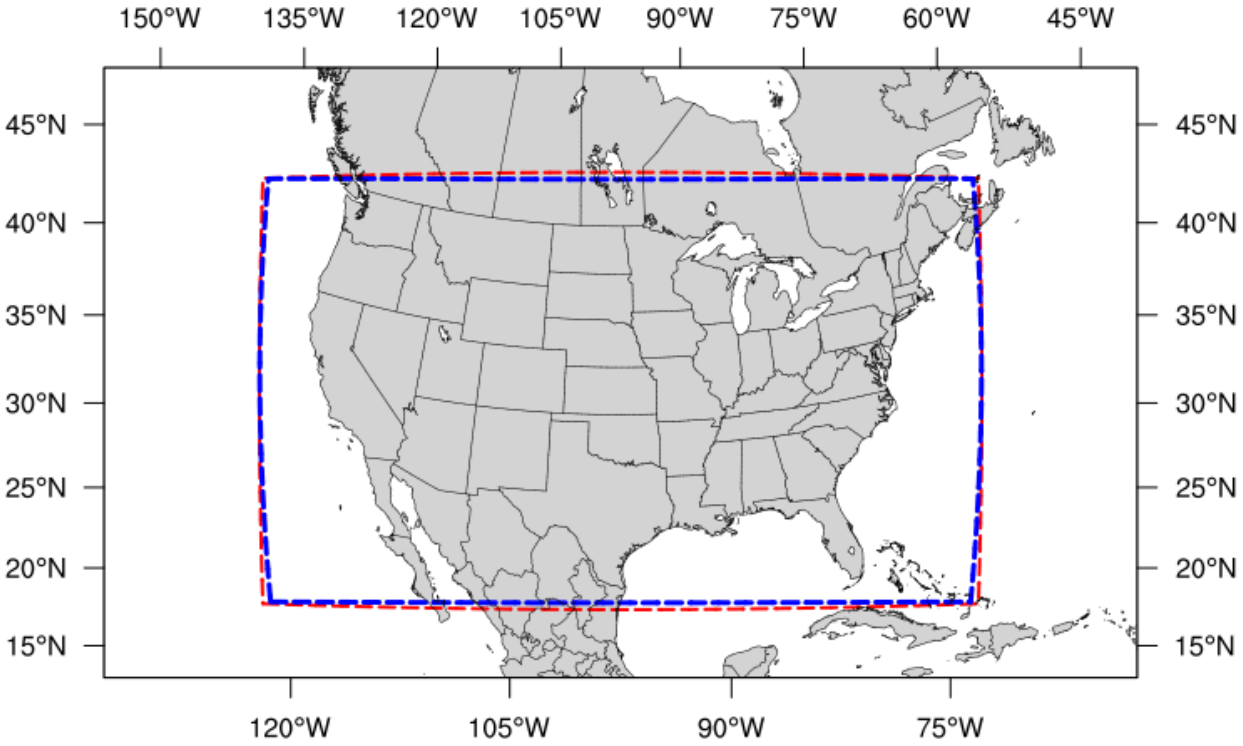

Fig. 6.1: *The boundary of the RRFS_CONUS_3km computational grid (red) and corresponding write-component grid (blue).*

suite definition file (SDF), since this SDF was specifically created for convection-allowing scales, and is the precursor to the operational physics suite that will be used in the RRFS.

As can be seen in Figure 6.1, the boundary of the write-component grid (in blue) sits just inside the computational domain (in red). This extra grid is required because the post-processing utility (UPP) is currently unable to process data on the native FV3 gnomonic grid (in red). Therefore, model data are interpolated to a Lambert conformal grid (the write component grid) in order for UPP to read in and correctly process the data.

The `RRFS_CONUS_13km` grid (Fig. 6.2) also covers the full CONUS, but due to its coarser resolution, and the need to remain within the HRRR domain, areas of the contiguous United States, such as Northern Washington, Southern Texas, and the Florida Keys, are closer to the boundaries of the grid than in the `RRFS_CONUS_3km` grid. This grid is meant to be run with the `FV3_GFS_v15p2` SDF.

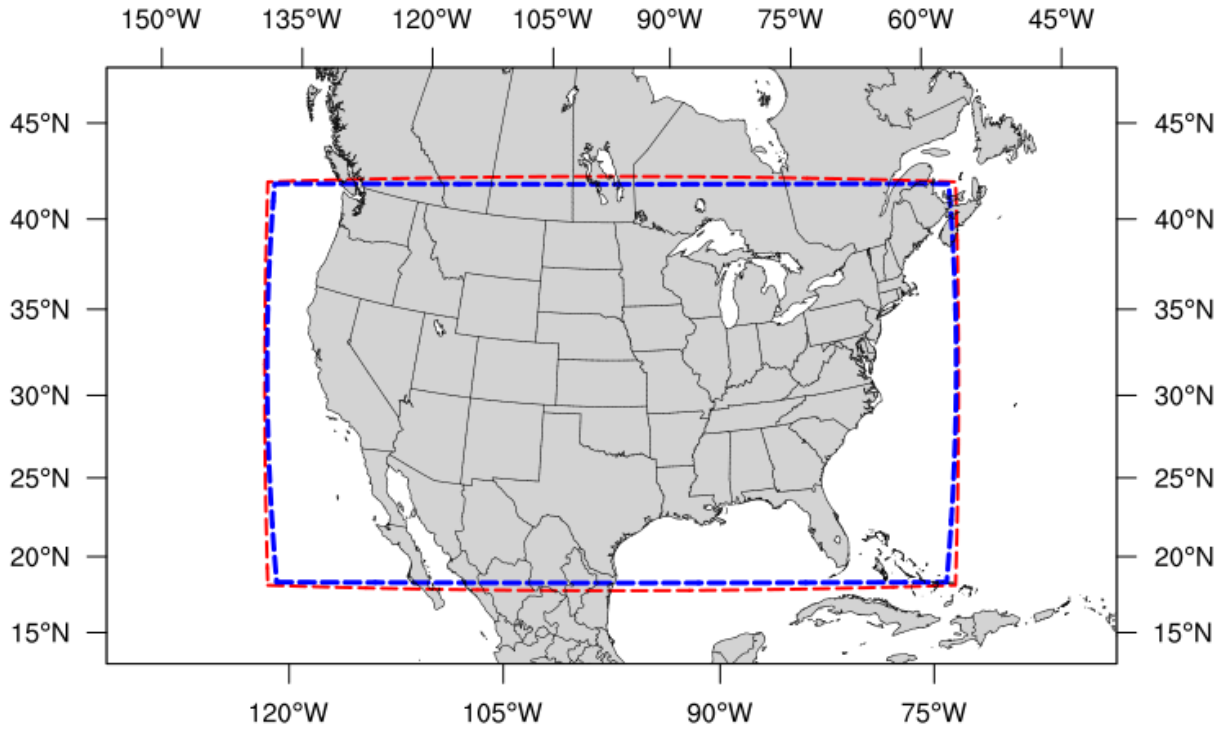

Fig. 6.2: *The boundary of the RRFS_CONUS_13km computational grid (red) and corresponding write-component grid (blue).*

The final predefined CONUS grid (Fig. 6.3) uses a 25-km resolution and is meant mostly for quick testing to ensure functionality prior to using a higher-resolution domain. However, for users who would like to use this domain for research, the `FV3_GFS_v15p2` SDF is recommended.

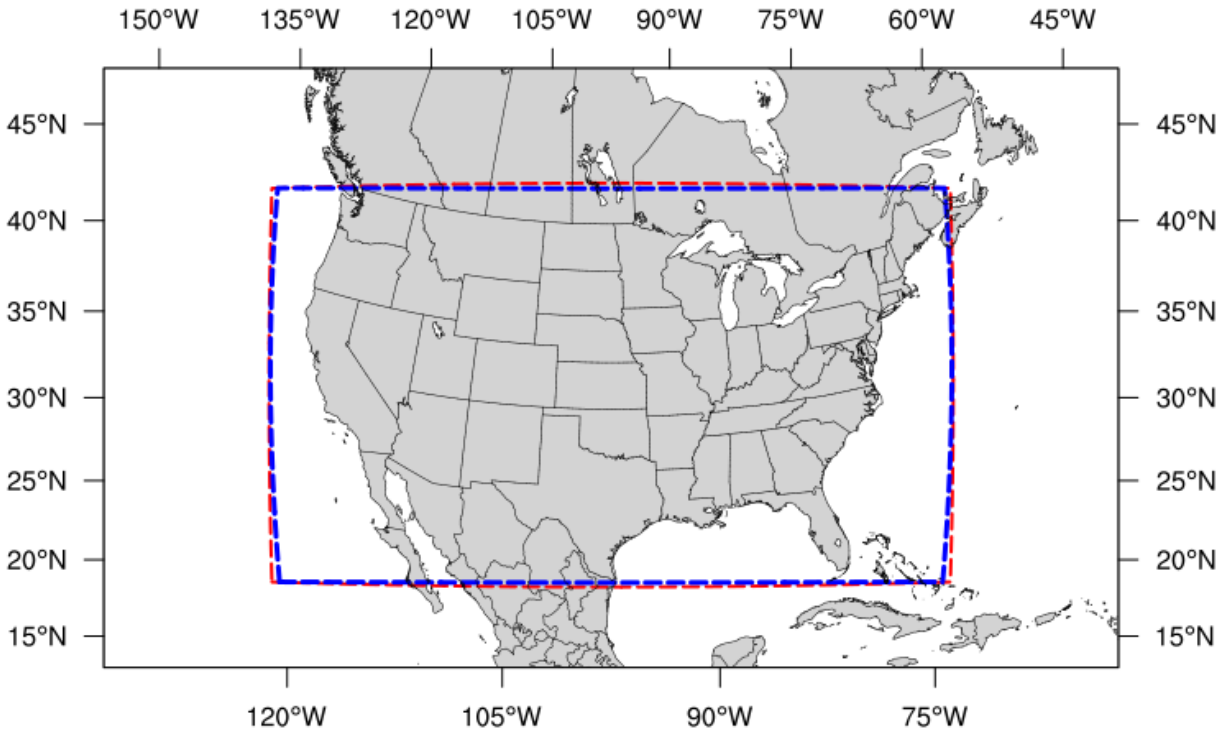

Fig. 6.3: *The boundary of the RRFS_CONUS_25km computational grid (red) and corresponding write-component grid (blue).*

6.2 Creating User-Generated Grids

While the three predefined grids available in this release are ideal for users just starting out with the SRW App, more advanced users may wish to create their own grid for testing over a different region and/or with a different resolution. Creating a user-defined grid requires knowledge of how the SRW App workflow functions, in particular, understanding the set of scripts that handle the workflow and experiment generation. It is also important to note that user-defined grids are not a supported feature of the current release, however information is being provided for the benefit of the FV3-LAM community.

With those caveats in mind, this section provides instructions for adding a new grid to the FV3-LAM workflow that will be generated using the "ESGgrid" method (i.e. using the regional_esg_grid code in the UFS_UTILS repository, where ESG stands for "Extended Schmidt Gnomonic"). We assume here that the grid to be generated covers a domain that (1) does not contain either of the poles and (2) does not cross the -180 deg –> +180 deg discontinuity in longitude near the international date line. Instructions for domains that do not have these restrictions will be provided in a future release.

The steps to add such a grid to the workflow are as follows:

1. Decide on the name of the grid. For the purposes of this documentation, the grid will be called "NEW_GRID".
2. Add NEW_GRID to the array `valid_vals_PREDEF_GRID_NAME` in the `ufs-srweather-app/regional_workflow/ush/valid_param_vals.sh` file.
3. In the file `ufs-srweather-app/regional_workflow/ush/set_predef_grid_params.sh`, add a stanza to the case statement `case ${PREDEF_GRID_NAME} in` for NEW_GRID. An example of such a stanza is given below along with comments describing the variables that need to be set.

To run a forecast experiment on NEW_GRID, start with a workflow configuration file for a successful experiment (this file is named `config.sh` and is located in the directory `ufs-srweather-app/regional_workflow/ush`) and change the line for `PREDEF_GRID_NAME` to the following:

```
PREDEF_GRID_NAME="NEW_GRID"
```

Then, generate a new experiment/workflow using `generate_FV3LAM_wflow.sh` in the usual way.

The following is an example of a stanza for "NEW_GRID" to be added to `set_predef_grid_params.sh`:

```
#
#-----------------------------------------------------------------------
#
# Stanza for NEW_GRID.  This grid covers [provide a description of the
# domain that NEW_GRID covers, its grid cell size, etc].
#
#-----------------------------------------------------------------------
#
"NEW_GRID")
```

(continues on next page)

(continued from previous page)

```
# The method used to generate the grid.  This example is specifically
# for the "ESGgrid" method.
  GRID_GEN_METHOD= "ESGgrid"

# The longitude and latitude of the center of the grid, in degrees.
  ESGgrid_LON_CTR=-97.5
  ESGgrid_LAT_CTR=38.5

# The grid cell sizes in the x and y directions, where x and y are the
# native coordinates of any ESG grid.  The units of x and y are in
# meters. These should be set to the nominal resolution we want the
# grid to have. The cells will have exactly these sizes in xy-space
# (computational space) but will have varying size in physical space.
# The advantage of the ESGgrid generation method over the GFDLgrid
# method is that an ESGgrid will have a much smaller variation in grid
# size in physical space than a GFDLgrid.
  ESGgrid_DELX="25000.0"
  ESGgrid_DELY="25000.0"

# The number of cells along the x and y axes.
  ESGgrid_NX=200
  ESGgrid_NY=112

# The width of the halo (in units of grid cells) that the temporary
# wide-halo grid created during the grid generation task (make_grid)
# will have.  This wide-halo grid gets "shaved" down to obtain the
# 4-cell-wide halo and 3-cell-wide halo grids that the forecast model
# (as well as other codes) will actually use.  Recall that the halo is
# needed to provide lateral boundary conditions to the forecast model.
# Usually, there is no need to modify this parameter.
  ESGgrid_WIDE_HALO_WIDTH=6

# The default physics time step that the forecast model will use. This
# is the (inverse) frequency with which (most of) the physics suite is
# called. The smaller the grid cell size is, the smaller this value
# needs to be in order to avoid numerical instabilities during the
# forecast.  The values specified below are used only if DT_ATMOS is
# not explicitly set in the user-specified experiment configuration
# file config.sh.  Note that this parameter may be suite dependent.
  if [ "${CCPP_PHYS_SUITE}" = "FV3_GFS_v15p2" ]; then
    DT_ATMOS=${DT_ATMOS:-"300"}
  elif [ "${CCPP_PHYS_SUITE}" = "FV3_RRFS_v1alpha" ]; then
    DT_ATMOS=${DT_ATMOS:-"40"}
  else
    DT_ATMOS=${DT_ATMOS:-"40"}
  fi

# Default MPI task layout (decomposition) along the x and y directions and blocksize.
# The values specified below are used only if they are not explicitly set in the user-
↪specified
# experiment configuration file config.sh.
  LAYOUT_X=${LAYOUT_X:-"5"}
```

(continues on next page)

(continued from previous page)

```
  LAYOUT_Y=${LAYOUT_Y:-"2"}
  BLOCKSIZE=${BLOCKSIZE:-"40"}

# The parameters for the write-component (aka "quilting") grid. This
# is the grid to which the output fields from the forecast are
# interpolated. The output fields are not specified on the native grid
# but are instead remapped to this write-component grid because the
# post-processing software (UPP; called during the run_post tasks) is
# not able to process fields on the native grid.  The variable
# "QUILTING", which specifies whether or not to use the
# write-component grid, is by default set to "TRUE".
  if [ "$QUILTING" = "TRUE" ]; then

# The number of "groups" of MPI tasks that may be running at any given
# time to write out the output.  Each write group will be writing to
# one set of output files (a dynf${fhr}.nc and a phyf${fhr}.nc file,
# where $fhr is the forecast hour).  Each write group contains
# WRTCMP_write_tasks_per_group tasks. Usually, it is sufficient to
# have just one write group.  This may need to be increased if the
# forecast is proceeding so quickly that a single write group cannot
# complete writing to its set of files before there is a need/request
# to start writing the next set of files at the next output time (this
# can happen, for instance, if the forecast model is trying to write
# output at every time step).
    WRTCMP_write_groups="1"

# The number of MPI tasks to allocate to each write group.
    WRTCMP_write_tasks_per_group="2"

# The coordinate system in which the write-component grid is
# specified. See the array valid_vals_WRTCMP_output_grid (defined in
# the script valid_param_vals.sh) for the values this can take on.
# The following example is specifically for the Lambert conformal
# coordinate system.
    WRTCMP_output_grid="lambert_conformal"

# The longitude and latitude of the center of the write-component
# grid.
    WRTCMP_cen_lon="${ESGgrid_LON_CTR}"
    WRTCMP_cen_lat="${ESGgrid_LAT_CTR}"

# The first and second standard latitudes needed for the Lambert
# conformal coordinate mapping.
    WRTCMP_stdlat1="${ESGgrid_LAT_CTR}"
    WRTCMP_stdlat2="${ESGgrid_LAT_CTR}"

# The number of grid points in the x and y directions of the
# write-component grid. Note that this xy coordinate system is that of
# the write-component grid (which in this case is Lambert conformal).
# Thus, it is in general different than the xy coordinate system of
# the native ESG grid.
    WRTCMP_nx="197"
```

(continues on next page)

(continued from previous page)

```
    WRTCMP_ny="107"

# The longitude and latitude of the lower-left corner of the
# write-component grid, in degrees.
    WRTCMP_lon_lwr_left="-121.12455072"
    WRTCMP_lat_lwr_left="23.89394570"

# The grid cell sizes along the x and y directions of the
# write-component grid.  Units depend on the coordinate system used by
# the grid (i.e. the value of WRTCMP_output_grid). For a Lambert
# conformal write-component grid, the units are in meters.
    WRTCMP_dx="${ESGgrid_DELX}"
    WRTCMP_dy="${ESGgrid_DELY}"

  fi
  ;;
```

CHAPTER

SEVEN

INPUT AND OUTPUT FILES

This chapter provides an overview of the input and output files needed by the components of the UFS SRW Application (*UFS_UTILS*, the UFS *Weather Model*, and *UPP*). Links to more detailed documentation for each of the components are provided.

7.1 Input Files

The SRW Application requires numerous input files to run: static datasets (fix files containing climatological information, terrain and land use data), initial and boundary conditions files, and model configuration files (such as namelists).

7.1.1 Initial and Boundary Condition Files

The external model files needed for initializing the runs can be obtained in a number of ways, including: pulled directly from NOMADS; limited data availability), pulled from the NOAA HPSS during the workflow execution (requires user access), or obtained and staged by the user from a different source. The data format for these files can be *GRIB2* or *NEMSIO*. More information on downloading and staging the external model data can be found in Section 7.3. Once staged, the end-to-end application will run the system and write output files to disk.

7.1.2 Pre-processing (UFS_UTILS)

When a user runs the SRW Application as described in the quickstart guide Section 2, input data for the pre-processing utilities is linked from a location on disk to your experiment directory by the workflow generation step. The pre-processing utilities use many different datasets to create grids, and to generate model input datasets from the external model files. A detailed description of the input files for the pre-processing utilities can be found here.

7.1.3 UFS Weather Model

The input files for the weather model include both static (fixed) files and grid and date specific files (terrain, initial conditions, boundary conditions, etc). The static fix files must be staged by the user unless you are running on a pre-configured platform, in which case you can link to the existing copy on that machine. See Section 7.3.1 for more information. The static, grid, and date specific files are linked in the experiment directory by the workflow scripts. An extensive description of the input files for the weather model can be found in the UFS Weather Model User's Guide. The namelists and configuration files for the SRW Application are created from templates by the workflow, as described in Section 7.1.5.

7.1.4 Unified Post Processor (UPP)

Documentation for the UPP input files can be found in the UPP User's Guide.

7.1.5 Workflow

The SRW Application uses a series of template files, combined with user selected settings, to create the required namelists and parameter files needed by the Application. These templates can be reviewed to see what defaults are being used, and where configuration parameters are assigned from the `config.sh` file.

List of Template Files

The template files for the SRW Application are located in `regional_workflow/ush/templates` and are shown in Table 7.1.

Table 7.1: Template files for a regional workflow.

File Name	Description
data_table	Cycle-independent file that the forecast model reads in at the start of each forecast. It is an empty file. No need to change.
diag_table_[CCPP]	File specifying the output fields of the forecast model. A different diag_table may be configured for different CCPP suites.
field_table_[CCPP]	Cycle-independent file that the forecast model reads in at the start of each forecast. It specifies the tracers that the forecast model will advect. A different field_table may be needed for different CCPP suites.
FV3.input.yml	YAML configuration file containing the forecast model's namelist settings for various physics suites. The values specified in this file update the corresponding values in the `input.nml` file. This file may be modified for the specific namelist options of your experiment.
FV3LAM_wflow.xml	Rocoto XML file to run the workflow. It is filled in using the `fill_template.py` python script that is called in the `generate_FV3LAM_wflow.sh`.
input.nml.FV3	Namelist file of the weather model.
model_configure	Settings and configurations for the NUOPC/ESMF main component.
nems.configure	NEMS (NOAA Environmental Modeling System) configuration file, no need to change because it is an atmosphere-only model in the SRW Application.
regional_grid.nml	Namelist settings for the code that generates an ESG grid.
README.xml_templating.md	Instructions for Rocoto XML templating with Jinja.

Additional information related to the `diag_table_[CCPP]`, `field_table_[CCPP]`, `input.nml.FV3`, `model_conigure`, and `nems.configure` can be found in the UFS Weather Model User's Guide, while information on the `regional_grid.nml` can be found in the UFS_UTILS User's Guide.

Migratory Route of the Input Files in the Workflow

Figure 7.1 shows how the case-specific input files in the `ufs-srweather-app/regional_workflow/ush/templates/` directory flow to the experiment directory. The value of `CCPP_PHYS_SUITE` is specified in the configuration file `config.sh`. The template input files corresponding to `CCPP_PHYS_SUITE`, such as `field_table` and `nems_configure`, are copied to the experiment directory `EXPTDIR` and the namelist file of the weather model `input.nml` is created from the `input.nml.FV3` and `FV3.input.yml` files by running the script `generate_FV3LAM_wflow.sh`. While running the task `RUN_FCST` in the regional workflow as shown in Figure 4.3, the `field_table`, `nems.configure`, and `input.nml` files, located in `EXPTDIR` are linked to the cycle directory `CYCLE_DIR/`, and `diag_table` and `model_configure` are copied from the `templates` directory. Finally, these files are updated with the variables specified in `var_defn.sh`.

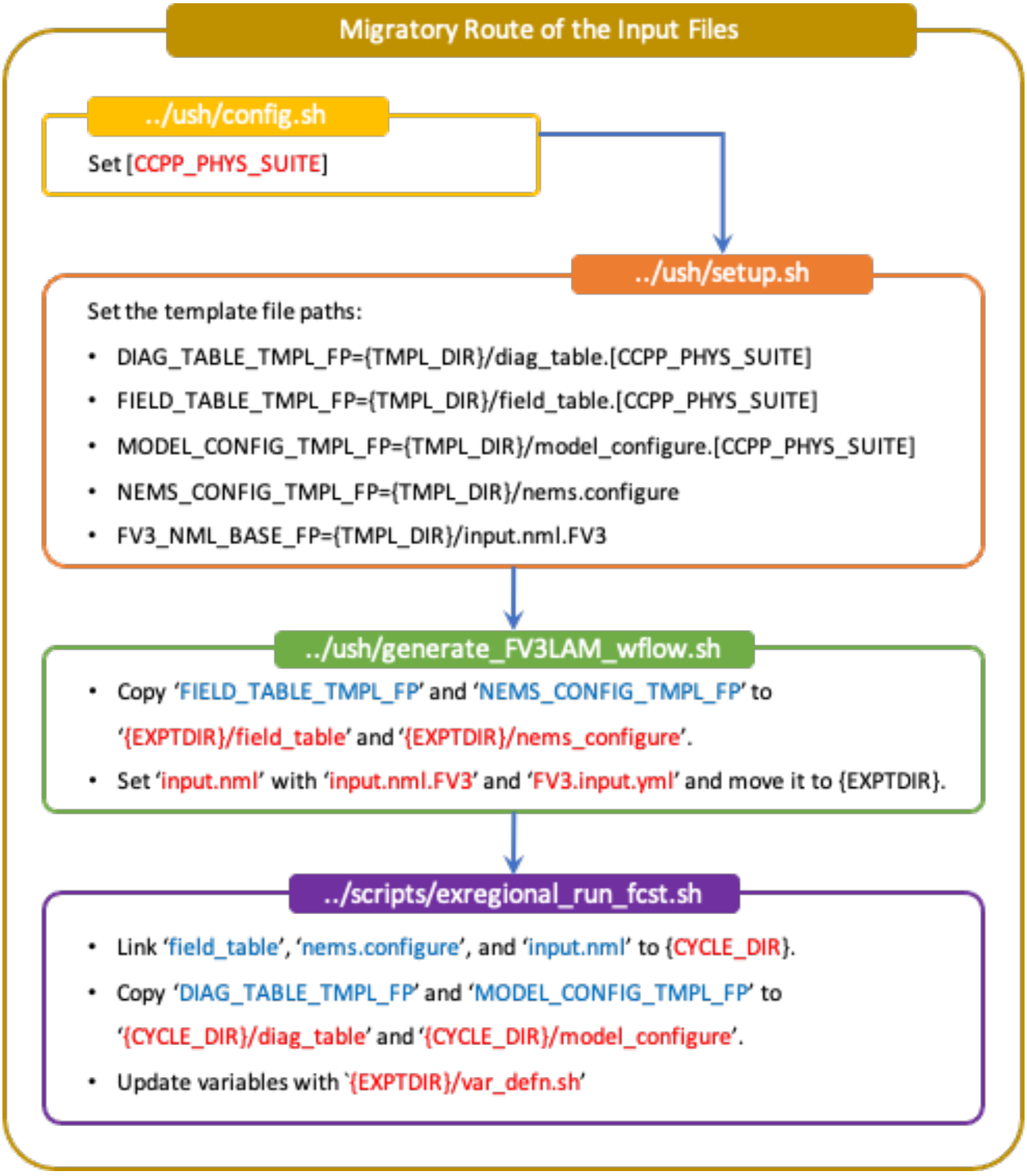

Fig. 7.1: *Migratory route of input files*

7.2 Output Files

The location of the output files written to disk is defined by the experiment directory, `EXPTDIR/YYYYMMDDHH`, as set in `config.sh`.

7.2.1 Initial and boundary condition files

The external model data used by `chgres_cube` (as part of the pre-processing utilities) are located in the experiment run directory under `EXPTDIR/YYYYMMDDHH/{EXTRN_MDL_NAME_ICS/LBCS}`.

7.2.2 Pre-processing (UFS_UTILS)

The files output by the pre-processing utilities reside in the `INPUT` directory under the experiment run directory `EXPTDIR/YYYYMMDDHH/INPUT` and consist of the following:

- `C403_grid.tile7.halo3.nc`
- `gfs_bndy.tile7.000.nc`
- `gfs_bndy.tile7.006.nc`
- `gfs_ctrl.nc`
- `gfs_data.nc -> gfs_data.tile7.halo0.nc`
- `grid_spec.nc -> ../../grid/C403_mosaic.halo3.nc`
- `grid.tile7.halo4.nc -> ../../grid/C403_grid.tile7.halo4.nc`
- `oro_data.nc -> ../../orog/C403_oro_data.tile7.halo0.nc`
- `sfc_data.nc -> sfc_data.tile7.halo0.nc`

These output files are used as inputs for the UFS weather model, and are described in the Users Guide.

7.2.3 UFS Weather Model

As mentioned previously, the workflow can be run in 'community' or 'nco' mode, which determines the location and names of the output files. In addition to this option, output can also be in netCDF or nemsio format. The output file format is set in the `model_configure` files using the `output_file` variable. At this time, due to limitations in the post-processing component, only netCDF format output is recommended for the SRW application.

Note: In summary, the fully supported options for this release include running in 'community' mode with netCDF format output files.

In this case, the netCDF output files are written to the EXPTDIR/YYYYMMDDHH directory. The bases of the file names are specified in the input file model_configure and are set to the following in the SRW Application:

- dynfHHH.nc
- phyfHHH.nc

Additional details may be found in the UFS Weather Model Users Guide.

7.2.4 Unified Post Processor (UPP)

Documentation for the UPP output files can be found here.

For the SRW Application, the weather model netCDF output files are written to the EXPTDIR/YYYYMMDDHH/postprd directory and have the naming convention (file->linked to):

- BGRD3D_{YY}{JJJ}{hh}{mm}f{fhr}00 -> {domain}.t{cyc}z.bgrd3df{fhr}.tmXX.grib2
- BGDAWP_{YY}{JJJ}{hh}{mm}f{fhr}00 -> {domain}.t{cyc}z.bgdawpf{fhr}.tmXX.grib2

The default setting for the output file names uses rrfs for {domain}. This may be overridden by the user in the config.sh settings.

If you wish to modify the fields or levels that are output from the UPP, you will need to make modifications to file fv3lam.xml, which resides in the UPP repository distributed with the UFS SRW Application. Specifically, if the code was cloned in the directory ufs-srweather-app, the file will be located in ufs-srweather-app/src/EMC_post/parm.

Note: This process requires advanced knowledge of which fields can be output for the UFS Weather Model.

Use the directions in the UPP User's Guide for details on how to make modifications to the fv3lam.xml file and for remaking the flat text file that the UPP reads, which is called postxconfig-NT-fv3lam.txt (default).

Once you have created the new flat text file reflecting your changes, you will need to modify your config.sh to point the workflow to the new text file. In your config.sh, set the following:

```
USE_CUSTOM_POST_CONFIG_FILE="TRUE"
CUSTOM_POST_CONFIG_PATH="/path/to/custom/postxconfig-NT-fv3lam.txt"
```

which tells the workflow to use the custom file located in the user-defined path. The path should include the filename. If this is set to true and the file path is not found, then an error will occur when trying to generate the SRW Application workflow.

You may then start your case workflow as usual and the UPP will use the new flat *.txt file.

7.3 Downloading and Staging Input Data

A set of input files, including static (fix) data and raw initial and lateral boundary conditions (IC/LBCs), are needed to run the SRW Application.

7.3.1 Static Files

A set of fix files are necessary to run the SRW Application. Environment variables describe the location of the static files: `FIXgsm`, `TOPO_DIR`, and `SFC_CLIMO_INPUT_DIR` are the directories where the static files are located. If you are on a pre-configured or configurable platform, there is no need to stage the fixed files manually because they have been prestaged and the paths are set in `regional_workflow/ush/setup.sh`. If the user's platform is not defined in that file, the static files can be pulled individually or as a full tar file from the FTP data repository or from Amazon Web Services (AWS) cloud storage and staged on your machine. The paths to the staged files must then be set in `config.sh` as follows:

- `FIXgsm=/path-to/fix/fix_am`
- `TOPO_DIR=/path-to/fix/fix_am/fix_orog`
- `SFC_CLIMO_INPUT_DIR=/path-to/fix_am/fix/sfc_climo/`

7.3.2 Initial Condition Formats and Source

The SRW Application currently supports raw initial and lateral boundary conditions from numerous models (i.e., FV3GFS, NAM, RAP, HRRR). The data can be provided in three formats: *NEMSIO*, netCDF, or *GRIB2*. The SRW Application currently only supports the use of NEMSIO and NetCDF input files from the GFS.

Environment variables describe what IC/LBC files to use (pre-staged files or files to be automatically pulled from the NOAA HPSS) and the location of the and IC/LBC files: `USE_USER_STAGED_EXTRN_FILES` is the `T/F` flag defining what raw data files to use, `EXTRN_MDL_SOURCE_BASEDIR_ICS` is the directory where the initial conditions are located, and `EXTRN_MDL_SOURCE_BASEDIR_LBCS` is the directory where the lateral boundary conditions are located.

If you have access to the NOAA HPSS and want to automatically download the IC/LBC files using the workflow, these environment variables can be left out of the `config.sh` file. However, if you do not have access to the NOAA HPSS and you need to pull and stage the data manually, you will need to set `USE_USER_STAGED_EXTRN_FILES` to `TRUE` and then set the paths to the where the IC/LBC files are located.

A small sample of IC/LBCs is available at the FTP data repository or from AWS cloud storage.

7.3.3 Initial and Lateral Boundary Condition Organization

The suggested directory structure and naming convention for the raw input files is described below. While there is flexibility to modify these settings, this will provide the most reusability for multiple dates when using the SRW Application workflow.

For ease of reusing the `config.sh` for multiple dates and cycles, it is recommended to set up your raw IC/LBC files such that it includes the model name (e.g., FV3GFS, NAM, RAP, HRRR) and `YYYYMMDDHH`, for example: `/path-to/model_data/FV3GFS/2019061518`. Since both initial and lateral boundary condition files are necessary, you can also include an ICS and LBCS directory. The sample IC/LBCs available at the FTP data repository are structured as follows:

- `/path-to/model_data/MODEL/YYYYMMDDHH/ICS`
- `/path-to/model_data/MODEL/YYYYMMDDHH/LBCS`

When files are pulled from the NOAA HPSS, the naming convention looks something like:

- FV3GFS (grib2): `gfs.t{cycle}z.pgrb2.0p25.f{fhr}`
- FV3GFS (nemsio): ICs: `gfs.t{cycle}z.atmanl.nemsio` and `gfs.t{cycle}z.sfcanl.nemsio`; LBCs: `gfs.t{cycle}z.atmf{fhr}.nemsio`
- RAP (grib2): `rap.t{cycle}z.wrfprsf{fhr}.grib2`
- HRRR (grib2): `hrrr.t{cycle}z.wrfprsf{fhr}.grib2`

In order to preserve the original file name, the `f00` files are placed in the `ICS` directory and all other forecast files are placed in the `LBCS` directory. Then, a symbolic link of the original files in the `ICS/LBCS` directory to the `YYYYMMDDHH` directory is suggested with the cycle removed. For example:

```
ln -sf /path-to/model_data/RAP/2020041212/ICS/rap.t12z.wrfprsf00.grib2 /path-to/model_data/
↪RAP/2020041212/rap.wrfprsf00.grib2
```

Doing this allows for the following to be set in the `config.sh` regardless of what cycle you are running:

```
USE_USER_STAGED_EXTRN_FILES="TRUE"
EXTRN_MDL_SOURCE_BASEDIR_ICS="/path-to/model_data/HRRR"
EXTRN_MDL_FILES_ICS=( "hrrr.wrfprsf00.grib2" )
EXTRN_MDL_SOURCE_BASEDIR_LBCS="/path-to/model_data/RAP"
EXTRN_MDL_FILES_LBCS=( "rap.wrfprsf03.grib2" "rap.wrfprsf06.grib2" )
```

If you choose to forgo the extra `ICS` and `LBCS` directory, you may also simply either rename the original files to remove the cycle or modify the `config.sh` to set:

```
EXTRN_MDL_FILES_ICS=( "hrrr.t{cycle}z.wrfprsf00.grib2" )
EXTRN_MDL_FILES_LBCS=( "rap.t{cycle}z.wrfprsf03.grib2" "rap.t{cycle}z.wrfprsf06.grib2" )
```

7.3.4 Default Initial and Lateral Boundary Conditions

The default initial and lateral boundary condition files are set to be a severe weather case from 20190615 at 00 UTC. FV3GFS grib2 files are the default model and file format. A tar file (`gst_model_data.tar.gz`) containing the model data for this case is available on EMC's FTP data repository at https://ftp.emc.ncep.noaa.gov/EIB/UFS/SRW/v1p0/simple_test_case/. It is also available on Amazon Web Services (AWS) at https://ufs-data.s3.amazonaws.com/public_release/ufs-srweather-app-v1.0.0/ic/gst_model_data.tar.gz.

7.3.5 Running the App for Different Dates

If users want to run the SRW Application for dates other than 06-15-2019, you will need to make a change in the case to specify the desired data. This is done by modifying the `config.sh` `DATE_FIRST_CYCL`, `DATE_LAST_CYCL`, and `CYCL_HRS` settings. The forecast length can be modified by changed the `FCST_LEN_HRS`. In addition, the lateral boundary interval can be specified using the `LBC_SPEC_INTVL_HRS` variable.

Users will need to ensure that the initial and lateral boundary condition files are available in the specified path for their new date, cycle, and forecast length.

7.3.6 Staging Initial Conditions Manually

If users want to run the SRW Application with raw model files for dates other than what are currently available on the preconfigured platforms, they need to stage the data manually. The data should be placed in `EXTRN_MDL_SOURCE_BASEDIR_ICS` and `EXTRN_MDL_SOURCE_BASEDIR_LBCS`. Raw model files may be available from a number of sources. A few examples are provided here for convenience.

NOMADS: https://nomads.ncep.noaa.gov/pub/data/nccf/com/{model}/prod, where model may be:

- GFS (grib2 or nemsio) - available for the last 10 days https://nomads.ncep.noaa.gov/pub/data/nccf/com/gfs/prod/
- NAM - available for the last 8 days https://nomads.ncep.noaa.gov/pub/data/nccf/com/nam/prod/
- RAP - available for the last 2 days https://nomads.ncep.noaa.gov/pub/data/nccf/com/rap/prod/
- HRRR - available for the last 2 days https://nomads.ncep.noaa.gov/pub/data/nccf/com/hrrr/prod/

NCDC archive:

- GFS: https://www.ncdc.noaa.gov/data-access/model-data/model-datasets/global-forcast-system-gfs
- NAM: https://www.ncdc.noaa.gov/data-access/model-data/model-datasets/north-american-mesoscale-forecast-system-nam

- RAP: https://www.ncdc.noaa.gov/data-access/model-data/model-datasets/rapid-refresh-rap

AWS S3:

- GFS: https://registry.opendata.aws/noaa-gfs-bdp-pds/
- HRRR: https://registry.opendata.aws/noaa-hrrr-pds/ (necessary fields for initializing available for dates 2015 and newer)

Google Cloud:

- HRRR: https://console.cloud.google.com/marketplace/product/noaa-public/hrrr

Others:

- Univ. of Utah HRRR archive: http://home.chpc.utah.edu/~u0553130/Brian_Blaylock/cgi-bin/hrrr_download.cgi
- NAM nest archive: https://www.ready.noaa.gov/archives.php
- NAM data older than 6 months can be requested through the Archive Information Request System: https://www.ncei.noaa.gov/has/HAS.FileAppRouter?datasetname=NAM218&subqueryby=STATION&applname=&outdest=FILE
- RAP isobaric data older than 6 months can be requested through the Archive Information Request System: https://www.ncei.noaa.gov/has/HAS.FileAppRouter?datasetname=RAP130&subqueryby=STATION&applname=&outdest=FILE

7.3.7 Coexistence of Multiple Files for the Same Date

If you would like to have multiple file formats (e.g., GRIB2, NEMSIO, netCDF) for the same date it is recommended to have a separate directory for each file format. For example, if you have GFS GRIB2 and NEMSIO files your directory structure might look like:

```
/path-to/model_data/FV3GFS/YYYYMMDDHH/ICS and LBCS
/path-to/model_data/FV3GFS_nemsio/YYYYMMDDHH/ICS and LBCS
```

If you want to use GRIB2 format files for FV3GFS you must also set two additional environment variables, including:

```
FV3GFS_FILE_FMT_ICS="grib2"
FV3GFS_FILE_FMT_LBCS="grib2"
```

This is ONLY necessary if you are using FV3GFS GRIB2 files. These settings may be removed if you are initializing from NEMSIO format FV3GFS files.

7.3.8 Best Practices for Conserving Disk Space and Keeping Files Safe

Initial and lateral boundary condition files are large and can occupy a significant amount of disk space. If various users will employ a common file system to conduct runs, it is recommended that the users share the same `EXTRN_MDL_SOURCE_BASEDIR_ICS` and `EXTRN_MDL_SOURCE_BASEDIR_LBCS` directories. That way, if raw model input files are already on disk for a given date they do not need to be replicated.

The files in the subdirectories of the `EXTRN_MDL_SOURCE_BASEDIR_ICS` and `EXTRN_MDL_SOURCE_BASEDIR_LBCS` directories should be write-protected. This prevents these files from being accidentally modified or deleted. The directories should generally be group writable so the directory can be shared among multiple users.

CHAPTER

EIGHT

CONFIGURING A NEW PLATFORM

The UFS SRW Application has been designed to work primarily on a number of Level 1 and 2 support platforms, as specified here. However, it is also designed with flexibility in mind, so that any sufficiently up-to-date machine with a UNIX-based operating system should be capable of running the application. A full list of prerequisites for installing the UFS SRW App and running the Graduate Student Test can be found in Section 8.7.

The first step to installing on a new machine is to install *NCEPLIBS* (https://github.com/NOAA-EMC/NCEPLIBS), the NCEP libraries package, which is a set of libraries created and maintained by NCEP and EMC that are used in many parts of the UFS. NCEPLIBS comes with a large number of prerequisites (see Section 8.7 for more info), but the only required software prior to starting the installation process are as follows:

- Fortran compiler with support for Fortran 2003
  - gfortran v9+ or ifort v18+ are the only ones tested, but others may work.
- C and C++ compilers compatible with the Fortran compiler
  - gcc v9+, ifort v18+, and clang v9+ (macOS, native Apple clang or LLVM clang) have been tested
- Python v3.6+
  - Prerequisite packages must be downloaded: jinja2, yaml and f90nml, as well as a number of additional Python modules (see Section 8.7) if the user would like to use the provided graphics scripts
- Perl 5
- git v1.8+
- CMake v3.12+
  - CMake v3.15+ is needed for building NCEPLIBS, but versions as old as 3.12 can be used to build NCEPLIBS-external, which contains a newer CMake that can be used for the rest of the build.

For both Linux and macOS, you will need to set the stack size to "unlimited" (if allowed) or the largest possible value.

```
# Linux, if allowed
ulimit -s unlimited

# macOS, this corresponds to 65MB
ulimit -S -s unlimited
```

For Linux systems, as long as the above software is available, you can move on to the next step: installing the *NCEPLIBS-external* package.

For macOS systems, some extra software is needed: `wget`, `coreutils`, `pkg-config`, and `gnu-sed`. It is recommended that you install this software using the Homebrew package manager for macOS (https://brew.sh/):

- brew install wget
- brew install cmake
- brew install coreutils
- brew install pkg-config
- brew install gnu-sed

However, it is also possible to install these utilities via Macports (https://www.macports.org/), or installing each utility individually (not recommended).

8.1 Installing NCEPLIBS-external

In order to facilitate the installation of NCEPLIBS (and therefore, the SRW and other UFS applications) on new platforms, EMC maintains a one-stop package containing most of the prerequisite libraries and software necessary for installing NCEPLIBS. This package is known as NCEPLIBS-external, and is maintained in a git repository at https://github.com/NOAA-EMC/NCEPLIBS-external. Instructions for installing these will depend on your platform, but generally so long as all the above-mentioned prerequisites have been installed you can follow the proceeding instructions verbatim (in bash; a csh-based shell will require different commands). Some examples for installing on specific platforms can be found in the *NCEPLIBS-external/doc directory <https://github.com/NOAA-EMC/NCEPLIBS-external/tree/release/public-v2/doc>*.

These instructions will install the NCEPLIBS-external in the current directory tree, so be sure you are in the desired location before starting.

```
export WORKDIR=`pwd`
export INSTALL_PREFIX=${WORKDIR}/NCEPLIBS-ufs-v2.0.0/
export CC=gcc
export FC=gfortran
export CXX=g++
```

The CC, CXX, and FC variables should specify the C, C++, and Fortran compilers you will be using, respectively. They can be the full path to the compiler if necessary (for example, on a machine with multiple versions of the same compiler). It will be important that all libraries and utilities are

built with the same set of compilers, so it is best to set these variables once at the beginning of the process and not modify them again.

```
mkdir -p ${INSTALL_PREFIX}/src && cd ${INSTALL_PREFIX}/src
git clone -b release/public-v2 --recursive https://github.com/NOAA-EMC/NCEPLIBS-external
cd NCEPLIBS-external
mkdir build && cd build
cmake -DCMAKE_INSTALL_PREFIX=${INSTALL_PREFIX} .. 2>&1 | tee log.cmake
make -j4 2>&1 | tee log.make
```

The previous commands go through the process of cloning the git repository for NCEPLIBS-external, creating and entering a build directory, and invoking cmake and make to build the code/libraries. The `make` step will take a while; as many as a few hours depending on your machine and various settings. It is highly recommended you use at least 4 parallel make processes to prevent overly long installation times. The `-j4` option in the make command specifies 4 parallel make processes, `-j8` would specify 8 parallel processes, while omitting the flag all together will run make serially (not recommended).

If you would rather use a different version of one or more of the software packages included in NCEPLIBS-external, you can skip building individual parts of the package by including the proper flags in your call to cmake. For example:

```
cmake -DBUILD_MPI=OFF -DCMAKE_INSTALL_PREFIX=${INSTALL_PREFIX} .. 2>&1 | tee log.cmake
```

will skip the building of MPICH that comes with NCEPLIBS-external. See the readme file `NCEPLIBS-external/README.md` for more information on these flags, or for general troubleshooting.

Once NCEPLIBS-external is installed, you can move on to installing NCEPLIBS.

8.2 Installing NCEPLIBS

Prior to building the UFS SRW Application on a new machine, you will need to install NCEPLIBS. Installation instructions will again depend on your platform, but so long as NCEPLIBS-external has been installed successfully you should be able to build NCEPLIBS. The following instructions will install the NCEPLIBS in the same directory tree as was used for NCEPLIBS-external above, so if you did not install NCEPLIBS-external in the same way, you will need to modify these commands.

```
cd ${INSTALL_PREFIX}/src
git clone -b release/public-v2 --recursive https://github.com/NOAA-EMC/NCEPLIBS
cd NCEPLIBS
mkdir build && cd build
export ESMFMKFILE=${INSTALL_PREFIX}/lib/esmf.mk
cmake -DCMAKE_INSTALL_PREFIX=${INSTALL_PREFIX} -DCMAKE_PREFIX_PATH=${INSTALL_PREFIX} -
↪DOPENMP=ON .. 2>&1 | tee log.cmake
make -j4 2>&1 | tee log.make
make deploy 2>&1 | tee log.deploy
```

As with NCEPLIBS-external, the above commands go through the process of cloning the git repository for NCEPLIBS, creating and entering a build directory, and invoking cmake and make to build

the code. The `make deploy` step created a number of modulefiles and scripts that will be used for setting up the build environment for the UFS SRW App. The `ESMFMKFILE` variable allows NCEPLIBS to find the location where ESMF has been built; if you receive a `ESMF not found, abort` error, you may need to specify a slightly different location:

```
export ESMFMKFILE=${INSTALL_PREFIX}/lib64/esmf.mk
```

Then delete and re-create the build directory and continue the build process as described above.

If you skipped the building of any of the software provided by NCEPLIBS-external, you may need to add the appropriate locations to your `CMAKE_PREFIX_PATH` variable. Multiple directories may be added, separated by semicolons (;) like in the following example:

```
cmake -DCMAKE_INSTALL_PREFIX=${INSTALL_PREFIX} -DCMAKE_PREFIX_PATH="${INSTALL_PREFIX};/
↪location/of/other/software" -DOPENMP=ON .. 2>&1 | tee log.cmake
```

Further information on including prerequisite libraries, as well as other helpful tips, can be found in the `NCEPLIBS/README.md` file.

Once the NCEPLIBS package has been successfully installed, you can move on to building the UFS SRW Application.

8.3 Building the UFS Short-Range Weather Application (UFS SRW App)

Building the UFS SRW App is similar to building NCEPLIBS, in that the code is stored in a git repository and is built using CMake software. The first step is to retrieve the code from Github, using the variables defined earlier:

```
cd ${WORKDIR}
git clone -b release/public-v1 https://github.com/ufs-community/ufs-srweather-app.git
cd ufs-srweather-app/
./manage_externals/checkout_externals
```

Here the procedure differs a bit from NCEPLIBS and NCEPLIBS-external. The UFS SRW App is maintained using an umbrella git repository that collects the individual components of the application from their individual, independent git repositories. This is handled using "Manage Externals" software, which is included in the application; this is the final step listed above, which should output a bunch of dialogue indicating that it is retrieving different code repositories as described in Table 3.1. It may take several minutes to download these repositories.

Once the Manage Externals step has completed, you will need to make sure your environment is set up so that the UFS SRW App can find all of the prerequisite software and libraries. There are a few ways to do this, the simplest of which is to load a modulefile if your machine supports Lua Modules:

```
module use ${INSTALL_PREFIX}/modules
module load NCEPLIBS/2.0.0
```

If your machine does not support Lua but rather TCL modules, see instructions in the `NCEPLIBS/README.md` file for converting to TCL modulefiles.

If your machine does not support modulefiles, you can instead source the provided bash script for setting up the environment:

```
source ${INSTALL_PREFIX}/bin/setenv_nceplibs.sh
```

This script, just like the modulefiles, will set a number of environment variables that will allow CMake to easily find all the libraries that were just built. There is also a csh version of the script in the same directory if your shell is csh-based. If you are using your machine's pre-built version of any of the NCEP libraries (not recommended), reference that file to see which variables should be set to point CMake in the right direction.

At this point there are just a few more variables that need to be set prior to building:

```
export CMAKE_C_COMPILER=mpicc
export CMAKE_CXX_COMPILER=mpicxx
export CMAKE_Fortran_COMPILER=mpifort
```

If you are using your machine's built-in MPI compilers, it is recommended you set the `CMAKE_*_COMPILER` flags to full paths to ensure that the correct MPI aliases are used. Finally, one last environment variable, `CMAKE_Platform`, must be set. This will depend on your machine; for example, on a macOS operating system with GNU compilers:

```
export CMAKE_Platform=macosx.gnu
```

This is the variable used by the weather model to set a few additional flags based on your machine. The available options can be found here.

Now all the prerequisites have been installed and variables set, so you should be ready to build the model!

```
mkdir build && cd build
cmake .. -DCMAKE_INSTALL_PREFIX=.. | tee log.cmake
make -j4 | tee log.make
```

On many platforms this build step will take less than 30 minutes, but for some machines it may take up to a few hours, depending on the system architecture, compiler and compiler flags, and number of parallel make processes used.

8.4 Setting Up Your Python Environment

The regional_workflow repository contains scripts for generating and running experiments, and these require some specific python packages to function correctly. First, as mentioned before, your platform will need Python 3.6 or newer installed. Once this is done, you will need to install several python packages that are used by the workflow: `jinja2` (https://jinja2docs.readthedocs.io/), `pyyaml` (https://pyyaml.org/wiki/PyYAML), and `f90nml` (https://pypi.org/project/f90nml/). These packages can be installed individually, but it is recommended you use a package manager (https://www.datacamp.com/community/tutorials/pip-python-package-manager).

If you have conda on your machine:

```
conda install jinja2 pyyaml f90nml
```

Otherwise you may be able to use pip3 (the Python3 package manager; may need to be installed separately depending on your platform):

```
pip3 install jinja2 pyyaml f90nml
```

Running the graphics scripts in `${WORKDIR}/ufs-srweather-app/regional_workflow/ush/Python` will require the additional packages `pygrib`, `cartopy`, `matplotlib`, `scipy`, and `pillow`. These can be installed in the same way as described above.

For the final step of creating and running an experiment, the exact methods will depend on if you are running with or without a workflow manager (Rocoto).

8.5 Running Without a Workflow Manager: Generic Linux and macOS Platforms

Now that the code has been built, you can stage your data as described in Section 7.3.

Once the data has been staged, setting up your experiment on a platform without a workflow manager is similar to the procedure for other platforms described in earlier chapters. Enter the `${WORKDIR}/ufs-srweather-app/regional_workflow/ush` directory and configure the workflow by creating a `config.sh` file as described in Chapter 5. There will be a few specific settings that you may need change prior to generating the experiment compared to the instructions for pre-configured platforms:

`MACHINE="MACOS" or MACHINE="LINUX"` These are the two `MACHINE` settings for generic, non-Rocoto-based platforms; you should choose the one most appropriate for your machine. `MACOS` has its own setting due to some differences in how command-line utilities function on Darwin-based operating systems.

`LAYOUT_X=2 LAYOUT_Y=2`

These are the settings that control the MPI decomposition when running the weather model. There are default values, but for your machine it is recommended that you specify your own layout to achieve the correct number of MPI processes for your application. In total, your machine should be able to handle `LAYOUT_X×LAYOUT_Y+WRTCMP_write_tasks_per_group` tasks. `WRTCMP_write_tasks_per_group` is the number of MPI tasks that will be set aside for writing model output, and it is a setting dependent on the domain you have selected. You can find and edit the value of this variable in the file `regional_workflow/ush/set_predef_grid_params.sh`.

`RUN_CMD_UTILS="mpirun -np 4"` This is the run command for MPI-enabled pre-processing utilities. Depending on your machine and your MPI installation, you may need to use a different command for launching an MPI-enabled executable.

`RUN_CMD_POST="mpirun -np 1"` This is the same as RUN_CMD_UTILS but for UPP.

**RUN_CMD_FCST='mpirun -np ${PE_MEMBER01}'** This is the run command for the weather model. It is **strongly** recommended that you use the variable `${PE_MEMBER01}` here, which is calculated within the workflow generation script (based on the layout and write tasks described above) and is the number of MPI tasks that the weather model will expect to run with. Running the weather model with a different number of MPI tasks than the workflow has been set up for can lead to segmentation faults and other errors. It is also important to use single quotes here (or escape the "$" character) so that `PE_MEMBER01` is not referenced until runtime, since it is not defined at the beginning of the workflow generation script.

FIXgsm=${WORKDIR}/data/fix_am The location of the `fix_am` static files. This and the following two static data sets will need to be downloaded to your machine, as described in Section 7.3.1.

TOPO_DIR=${WORKDIR}/data/fix_orog Location of `fix_orog` static files

SFC_CLIMO_INPUT_DIR=${WORKDIR}/data/fix_sfc_climo Location of `climo_fields_netcdf` static files

Once you are happy with your settings in `config.sh`, it is time to run the workflow and move to the experiment directory (that is printed at the end of the script's execution):

```
./generate_FV3LAM_wflow.sh
export EXPTDIR="your experiment directory"
cd $EXPTDIR
```

From here, you can run each individual task of the UFS SRW App using the provided run scripts:

```
cp ${WORKDIR}/ufs-srweather-app/regional_workflow/ush/wrappers/*sh .
cp ${WORKDIR}/ufs-srweather-app/regional_workflow/ush/wrappers/README.md .
```

The `README.md` file will contain instructions on the order that each script should be run in. An example of wallclock times for each task for an example run (2017 Macbook Pro, macOS Catalina, 25km CONUS domain, 48hr forecast) is listed in Table 8.1.

Table 8.1: Example wallclock times for each workflow task.

UFS Component	Script Name	Num. Cores	Wall time
UFS_UTILS	./run_get_ics.sh	n/a	3 s
UFS_UTILS	./run_get_lbcs.sh	n/a	3 s
UFS_UTILS	./run_make_grid.sh	n/a	9 s
UFS_UTILS	./run_make_orog.sh	4	1 m
UFS_UTILS	./run_make_sfc_climo.sh	4	27 m
UFS_UTILS	./run_make_ics.sh	4	5 m
UFS_UTILS	./run_make_lbcs.sh	4	5 m
ufs-weather-model	./run_fcst.sh	6	1h 40 m
EMC_post	./run_post.sh	1	7 m

8.6 Running on a New Platform with Rocoto Workflow Manager

All official HPC platforms for the UFS SRW App release make use of the Rocoto workflow management software for running experiments. If you would like to use the Rocoto workflow manager on a new machine, you will have to make modifications to the scripts in the `regional_workflow` repository. The easiest way to do this is to search the files in the `regional_workflow/scripts` and `regional_workflow/ush` directories for an existing platform name (e.g. `CHEYENNE`) and add a stanza for your own unique machine (e.g. `MYMACHINE`). As an example, here is a segment of code from `regional_workflow/ush/setup.sh`, where the highlighted text is an example of the kind of change you will need to make:

```
...
  "CHEYENNE")
    WORKFLOW_MANAGER="rocoto"
    NCORES_PER_NODE=36
    SCHED="${SCHED:-pbspro}"
    QUEUE_DEFAULT=${QUEUE_DEFAULT:-"regular"}
    QUEUE_HPSS=${QUEUE_HPSS:-"regular"}
    QUEUE_FCST=${QUEUE_FCST:-"regular"}
    ;;

  "MYMACHINE")
    WORKFLOW_MANAGER="rocoto"
    NCORES_PER_NODE=your_machine_cores_per_node
    SCHED="${SCHED:-your_machine_scheduler}"
    QUEUE_DEFAULT=${QUEUE_DEFAULT:-"your_machine_queue_name"}
    QUEUE_HPSS=${QUEUE_HPSS:-"your_machine_queue_name"}
    QUEUE_FCST=${QUEUE_FCST:-"your_machine_queue_name"}
    ;;

   "STAMPEDE")
     WORKFLOW_MANAGER="rocoto"
...
```

You will also need to add `MYMACHINE` to the list of valid machine names in `regional_workflow/ush/valid_param_vals.sh`. The minimum list of files that will need to be modified in this way are as follows (all in the `regional_workflow` repository):

- `scripts/exregional_run_post.sh`, line 131
- `scripts/exregional_make_sfc_climo.sh`, line 162
- `scripts/exregional_make_lbcs.sh`, line 114
- `scripts/exregional_make_orog.sh`, line 147
- `scripts/exregional_make_grid.sh`, line 145
- `scripts/exregional_run_fcst.sh`, line 140
- `scripts/exregional_make_ics.sh`, line 114
- `ush/setup.sh`, lines 431 and 742

- `ush/launch_FV3LAM_wflow.sh`, line 104
- `ush/get_extrn_mdl_file_dir_info.sh`, many lines, starting around line 589
- `ush/valid_param_vals.sh`, line 3
- `ush/load_modules_run_task.sh`, line 126
- `ush/set_extrn_mdl_params.sh`, many lines, starting around line 61

The line numbers may differ slightly given future bug fixes. Additionally, you may need to make further changes depending on the exact setup of your machine and Rocoto installation. Information about installing and configuring Rocoto on your machine can be found in the Rocoto Github repository: https://github.com/christopherwharrop/rocoto

8.7 Software/Operating System Requirements

Those requirements highlighted in **bold** are included in the NCEPLIBS-external (https://github.com/NOAA-EMC/NCEPLIBS-external) package.

Minimum platform requirements for the UFS SRW Application and NCEPLIBS:

- POSIX-compliant UNIX-style operating system
- >40 GB disk space
  - 18 GB input data from GFS, RAP, and HRRR for Graduate Student Test
  - 6 GB for NCEPLIBS-external and NCEPLIBS full installation
  - 1 GB for ufs-srweather-app installation
  - 11 GB for 48hr forecast on CONUS 25km domain
- 4GB memory (CONUS 25km domain)
- Fortran compiler with full Fortran 2008 standard support
- C and C++ compiler
- Python v3.6+, including prerequisite packages `jinja2`, `pyyaml` and `f90nml`
- Perl 5
- git v1.8+
- MPI (**MPICH**, OpenMPI, or other implementation)
- wgrib2
- CMake v3.12+
- Software libraries
  - **netCDF (C and Fortran libraries)**
  - **HDF5**
  - **ESMF** 8.0.0

  - **Jasper**
  - **libJPG**
  - **libPNG**
  - **zlib**

macOS-specific prerequisites:

- brew install wget
- brew install cmake
- brew install coreutils
- brew install pkg-config
- brew install gnu-sed

Optional but recommended prerequisites:

- Conda for installing/managing Python packages
- Bash v4+
- Rocoto Workflow Management System (1.3.1)
- **CMake v3.15+**
- Python packages scipy, matplotlib, pygrib, cartopy, and pillow for graphics

CHAPTER

NINE

WORKFLOW END-TO-END (WE2E) TESTS

The SRW Application's experiment generation system contains a set of end-to-end tests that exercise various configurations of that system as well as those of the pre-processing, UFS Weather Model, and UPP post-processing codes. The script to run these tests is named `run_experiments.sh` and is located in the directory `ufs-srweather-app/regional_workflow/tests`. A complete list of the available tests can be found in `baselines_list.txt` in that directory. This list is extensive; it is not recommended to run all of the tests as some are computationally expensive. A subset of the tests supported for this release of the SRW Application can be found in the file `testlist.release_public_v1.txt`.

The base experiment configuration file for each test is located in the `baseline_configs` subdirectory. Each file is named `config.${expt_name}.sh`, where `${expt_name}` is the name of the corresponding test configuration. These base configuration files are subsets of the full `config.sh` experiment configuration file used in Section 2.4.1 and described in Section 4.5.2. For each test that the user wants to run, the `run_experiments.sh` script reads in its base configuration file and generates from it a full `config.sh` file (a copy of which is placed in the experiment directory for the test).

Since `run_experiments.sh` calls `generate_FV3LAM_wflow.sh` for each test to be run, the Python modules required for experiment generation must be loaded before `run_experiments.sh` can be called. See Section 2.4.2 for information on loading the Python environment on supported platforms. Note also that `run_experiments.sh` assumes that all of the executables have been built.

The user specifies the set of test configurations that the `run_experiments.sh` script will run by creating a text file, say `expts_list.txt`, that contains a list of tests (one per line) and passing the name of that file to the script. For each test in the file, `run_experiments.sh` will generate an experiment directory and, by default, will continuously (re)launch its workflow by inserting a new cron job in the user's cron table. This cron job calls the workflow launch script `launch_FV3LAM_wflow.sh` located in the experiment directory until the workflow either completes successfully (i.e. all tasks are successful) or fails (i.e. at least one task fails). The cron job is then removed from the user's cron table.

The script `run_experiments.sh` accepts the command line arguments shown in Table 9.1.

Table 9.1: Command line arguments for the WE2E testing script `run_experiments.sh`.

Command Line Argument	Description	Optional
expts_file	Name of the file containing the list of tests to run. If `expts_file` is the absolute path to a file, it is used as is. If it is a relative path (including just a file name), it is assumed to be given relative to the path from which this script is called.	No
machine	Machine name	No
account	HPC account to use	No
use_cron_to_relaunch	Flag that specifies whether or not to use a cron job to continuously relaunch the workflow	Yes. Default value is `TRUE` (set in `run_experiments.sh`).
cron_relaunch_intvl_mnts	Frequency (in minutes) with which cron will relaunch the workflow	Used only if `use_cron_to_relaunch` is set to `TRUE`. Default value is "02" (set in `run_experiments.sh`).

For example, to run the tests named `grid_RRFS_CONUS_25km_ics_FV3GFS_lbcs_FV3GFS_suite_GFS_v15p2` and `grid_RRFS_CONUS_25km_ics_HRRR_lbcs_RAP_suite_RRFS_v1alpha` on Cheyenne, first create the file `expts_list.txt` containing the following lines:

```
grid_RRFS_CONUS_25km_ics_FV3GFS_lbcs_FV3GFS_suite_GFS_v15p2
grid_RRFS_CONUS_25km_ics_HRRR_lbcs_RAP_suite_RRFS_v1alpha
```

Then, from the `ufs-srweather-app/regional_workflow/tests` directory, issue the following command:

```
./run_experiments.sh expts_file="expts_list.txt" machine=cheyenne account="account_name"
```

where `account_name` should be replaced by the account to which to charge the core-hours used by the tests. Running this command will automatically insert an entry into the user's crontab that regularly (re)launches the workflow. The experiment directories will be created under `ufs-srweather-app/../expt_dirs`, and the name of each experiment directory will be identical to the name of the corresponding test.

To see if a test completed successfully, look at the end of the `log.launch_FV3LAM_wflow` file (which is the log file that `launch_FV3LAM_wflow.sh` appends to every time it is called) located in the experiment directory for that test:

```
Summary of workflow status:
~~~~~~~~~~~~~~~~~~~~~~~~~~

  1 out of 1 cycles completed.
  Workflow status:  SUCCESS
```

(continues on next page)

(continued from previous page)

```
========================================================================
End of output from script "launch_FV3LAM_wflow.sh".
========================================================================
```

Use of cron for all tests to be run by `run_experiments.sh` can be turned off by instead issuing the following command:

```
./run_experiments.sh expts_file="expts_list.txt" machine=cheyenne account="account_name" use_
↪cron_to_relaunch=FALSE
```

In this case, the experiment directories for the tests will be created, but their workflows will not be (re)launched. For each test, the user will have to go into the experiment directory and either manually call the `launch_FV3LAM_wflow.sh` script or use the Rocoto commands described in Chapter 12 to (re)launch the workflow. Note that if using the Rocoto commands directly, the log file `log.launch_FV3LAM_wflow` will not be created; in this case, the status of the workflow can be checked using the `rocotostat` command (see Chapter 12).

CHAPTER

TEN

GRAPHICS GENERATION

Two Python plotting scripts are provided to generate plots from the FV3-LAM post-processed GRIB2 output over the CONUS for a number of variables, including:

- 2-m temperature
- 2-m dew point temperature
- 10-m winds
- 500 hPa heights, winds, and vorticity
- 250 hPa winds
- Accumulated precipitation
- Composite reflectivity
- Surface-based CAPE/CIN
- Max/Min 2-5 km updraft helicity
- Sea level pressure (SLP)

The Python scripts are located under `ufs-srweather-app/regional_workflow/ush/Python`. The script `plot_allvars.py` plots the output from a single cycle within an experiment, while the script `plot_allvars_diff.py` plots the difference between the same cycle from two different experiments (e.g. the experiments may differ in some aspect such as the physics suite used). If plotting the difference, the two experiments must be on the same domain and available for the same cycle starting date/time and forecast hours.

The Python scripts require a cycle starting date/time in YYYYMMDDHH format, a starting forecast hour, an ending forecast hour, a forecast hour increment, the paths to one or two experiment directories, and a path to the directory where the Cartopy Natural Earth shape files are located. The full set of Cartopy shape files can be downloaded at https://www.naturalearthdata.com/downloads/. For convenience, the small subset of files required for these Python scripts can be obtained from the EMC ftp data repository or from AWS cloud storage. In addition, the Cartopy shape files are available on a number of Level 1 platforms in the following locations:

On Cheyenne:

```
/glade/p/ral/jntp/UFS_SRW_app/tools/NaturalEarth
```

On Hera:

```
/scratch2/BMC/det/UFS_SRW_app/v1p0/fix_files/NaturalEarth
```

On Jet:

```
/lfs4/BMC/wrfruc/FV3-LAM/NaturalEarth
```

On Orion:

```
/work/noaa/gsd-fv3-dev/UFS_SRW_App/v1p0/fix_files/NaturalEarth
```

On Gaea:

```
/lustre/f2/pdata/esrl/gsd/ufs/NaturalEarth
```

The medium scale (1:50m) cultural and physical shapefiles are used to create coastlines and other geopolitical borders on the map. Cartopy provides the 'background_img()' method to add background images in a convenient way. The default scale (resolution) of background attributes in the Python scripts is 1:50m Natural Earth I with Shaded Relief and Water, which should be sufficient for most regional applications.

The appropriate environment must be loaded to run the scripts, which require Python 3 with the `scipy`, `matplotlib`, `pygrib`, `cartopy`, and `pillow` packages. This Python environment has already been set up on Level 1 platforms and can be activated as follows:

On Cheyenne:

```
module load ncarenv
ncar_pylib /glade/p/ral/jntp/UFS_SRW_app/ncar_pylib/python_graphics
```

On Hera and Jet:

```
module use -a /contrib/miniconda3/modulefiles
module load miniconda3
conda activate pygraf
```

On Orion:

```
module use -a /apps/contrib/miniconda3-noaa-gsl/modulefiles
module load miniconda3
conda activate pygraf
```

On Gaea:

```
module use /lustre/f2/pdata/esrl/gsd/contrib/modulefiles
module load miniconda3/4.8.3-regional-workflow
```

Note: If using one of the batch submission scripts described below, the user does not need to manually load an environment because the scripts perform this task.

10.1 Plotting output from one experiment

Before generating plots, it is convenient to change location to the directory containing the plotting scripts:

```
cd ufs-srweather-app/regional_workflow/ush/Python
```

To generate plots for a single cycle, the `plot_allvars.py` script must be called with the following six command line arguments:

1. Cycle date/time (`CDATE`) in YYYYMMDDHH format
2. Starting forecast hour
3. Ending forecast hour
4. Forecast hour increment
5. The top level of the experiment directory `EXPTDIR` containing the post-processed data. The script will look for the data files in the directory `EXPTDIR/CDATE/postprd`.
6. The base directory `CARTOPY_DIR` of the cartopy shapefiles. The script will look for the shape files (`*.shp`) in the directory `CARTOPY_DIR/shapefiles/natural_earth/cultural`.

An example of plotting output from a cycle generated using the sample experiment/workflow configuration in the `config.community.sh` script (which uses the GFSv15p2 suite definition file) is as follows:

```
python plot_allvars.py 2019061500 6 48 6 /path-to/expt_dirs/test_CONUS_25km_GFSv15p2 /path-
↪to/NaturalEarth
```

The output files (in .png format) will be located in the directory `EXPTDIR/CDATE/postprd`, where in this case `EXPTDIR` is `/path-to/expt_dirs/test_CONUS_25km_GFSv15p2` and `CDATE` is `2019061500`.

10.2 Plotting differences from two experiments

To generate difference plots, the `plot_allvars_diff.py` script must be called with the following seven command line arguments:

1. Cycle date/time (`CDATE`) in YYYYMMDDHH format
2. Starting forecast hour
3. Ending forecast hour
4. Forecast hour increment
5. The top level of the first experiment directory `EXPTDIR1` containing the first set of post-processed data. The script will look for the data files in the directory `EXPTDIR1/CDATE/postprd`.

6. The top level of the first experiment directory `EXPTDIR2` containing the second set of post-processed data. The script will look for the data files in the directory `EXPTDIR2/CDATE/postprd`.
7. The base directory `CARTOPY_DIR` of the cartopy shapefiles. The script will look for the shape files (`*.shp`) in the directory `CARTOPY_DIR/shapefiles/natural_earth/cultural`.

An example of plotting differences from two experiments for the same date and predefined domain where one uses the "FV3_GFS_v15p2" suite definition file (SDF) and one using the "FV3_RRFS_v1alpha" SDF is as follows:

```
python plot_allvars_diff.py 2019061518 6 18 3 /path-to/expt_dirs1/test_CONUS_3km_GFSv15p2 /
↪path-to/expt_dirs2/test_CONUS_3km_RRFSv1alpha /path-to/NaturalEarth
```

In this case, the output png files will be located in the directory `EXPTDIR1/CDATE/postprd`.

10.3 Submitting plotting scripts through a batch system

If the Python scripts are being used to create plots of multiple forecast lead times and forecast variables, then you may need to submit them to the batch system. Example scripts are provided called `sq_job.sh` and `sq_job_diff.sh` for use on a platform such as Hera that uses the Slurm job scheduler or `qsub_job.sh` and `qsub_job_diff.sh` for use on a platform such as Cheyenne that uses PBS as the job scheduler. Examples of these scripts are located under `ufs-srweather-app/regional_workflow/ush/Python` and can be used as a starting point to create a batch script for your platform/job scheduler of use.

At a minimum, the account should be set appropriately prior to job submission:

```
#SBATCH --account=an_account
```

Depending on the platform you are running on, you may also need to adjust the settings to use the correct Python environment and path to the shape files.

When using these batch scripts, several environment variables must be set prior to submission. If plotting output from a single cycle, the variables to set are `HOMErrfs` and `EXPTDIR`. In this case, if the user's login shell is csh/tcsh, these variables can be set as follows:

```
setenv HOMErrfs /path-to/ufs-srweather-app/regional_workflow
setenv EXPTDIR /path-to/experiment/directory
```

If the user's login shell is bash, they can be set as follows:

```
export HOMErrfs=/path-to/ufs-srweather-app/regional_workflow
export EXPTDIR=/path-to/experiment/directory
```

If plotting the difference between the same cycle from two different experiments, the variables to set are `HOMErrfs`, `EXPTDIR1`. and `EXPTDIR2`. In this case, if the user's login shell is csh/tcsh, these variables can be set as follows:

```
setenv HOMErrfs /path-to/ufs-srweather-app/regional_workflow
setenv EXPTDIR1 /path-to/experiment/directory1
setenv EXPTDIR2 /path-to/experiment/directory2
```

If the user's login shell is bash, they can be set as follows:

```
export HOMErrfs=/path-to/ufs-srweather-app/regional_workflow
export EXPTDIR1=/path-to/experiment/directory1
export EXPTDIR2=/path-to/experiment/directory2
```

In addition, the variables `CDATE`, `FCST_START`, `FCST_END`, and `FCST_INC` in the batch scripts can be modified depending on the user's needs. By default, `CDATE` is set as follows in the batch scripts:

```
export CDATE=${DATE_FIRST_CYCL}${CYCL_HRS}
```

This sets `CDATE` to the first cycle in the set of cycles that the experiment has run. If the experiment contains multiple cycles and the user wants to plot output from a cycle other than the very first one, `CDATE` in the batch scripts will have to be set to the specific YYYYMMDDHH value for that cycle. Also, to plot hourly forecast output, `FCST_INC` should be set to 1; to plot only a subset of the output hours, `FCST_START`, `FCST_END`, and `FCST_INC` must be set accordingly, e.g. to generate plots for every 6th forecast hour starting with forecast hour 6 and ending with the last forecast hour, use

```
export FCST_START=6
export FCST_END=${FCST_LEN_HRS}
export FCST_INC=6
```

The scripts must be submitted using the command appropriate for the job scheduler used on your platform. For example, on Hera, `sq_job.sh` can be submitted as follows:

```
sbatch sq_job.sh
```

On Cheyenne, `qsub_job.sh` can be submitted as follows:

```
qsub qsub_job.sh
```

CHAPTER

ELEVEN

FAQ

11.1 How do I turn On/Off the Cycle-Independent Workflow Tasks

The first three pre-processing tasks `make_grid`, `make_orog`, and `make_sfc_climo` are cycle-independent, meaning that they only need to be run once per experiment. If the grid, orography, and surface climatology files that these tasks generate are already available (e.g. from a previous experiment that used the same grid as the current), then these tasks can be skipped by having the workflow use those pre-generated files. This can be done by adding the following lines to the `config.sh` script before running the `generate_FV3LAM_wflow.sh` script:

```
RUN_TASK_MAKE_GRID="FALSE"
GRID_DIR="/path/to/directory/containing/grid/files"
RUN_TASK_MAKE_OROG="FALSE"
OROG_DIR="/path/to/directory/containing/orography/files"
RUN_TASK_MAKE_SFC_CLIMO="FALSE"
SFC_CLIMO_DIR="/path/to/directory/containing/surface/climatology/files"
```

The `RUN_TASK_MAKE_GRID`, `RUN_TASK_MAKE_OROG`, and `RUN_TASK_MAKE_SFC_CLIMO` flags disable their respective tasks, and `GRID_DIR`, `OROG_DIR`, and `SFC_CLIMO_DIR` specify the directories in which the workflow can find the pre-generated grid, orography, and surface climatology files, respectively (these directories may be the same, i.e. all three sets of files may be placed in the same location). By default, the `RUN_TASK_MAKE_...` flags are set to `TRUE` in `config_defaults.sh`, i.e. the workflow will by default run the `make_grid`, `make_orog`, and `make_sfc_climo` tasks.

11.2 How do I define an experiment name?

The name of the experiment is set in the `config.sh` file using the variable `EXPT_SUBDIR`. See Section 2.4.1 for more details.

11.3 How do I change the Suite Definition File (SDF)?

The SDF is set in the `config.sh` file using the variable `CCPP_PHYS_SUITE`. When the `generate_FV3LAM_wflow.sh` script is run, the SDF file is copied from its location in the forecast model directory to the experiment directory `EXPTDIR`.

11.4 How do I restart a DEAD task?

On platforms that utilize Rocoto workflow software (such as NCAR's Cheyenne machine), sometimes if something goes wrong with the workflow a task may end up in the DEAD state:

```
rocotostat -w FV3SAR_wflow.xml -d FV3SAR_wflow.db -v 10
       CYCLE             TASK        JOBID    STATE    EXIT STATUS  TRIES DURATION
==========================================================================================
201905200000        make_grid      9443237   QUEUED              -      0      0.0
201905200000        make_orog            -        -              -      -        -
201905200000   make_sfc_climo            -        -              -      -        -
201905200000    get_extrn_ics      9443293     DEAD            256      3      5.0
```

This means that the dead task has not completed successfully, so the workflow has stopped. Once the issue has been identified and fixed (by referencing the log files), the failed task can re-run using the `rocotorewind` command:

```
rocotorewind -w FV3SAR_wflow.xml -d FV3SAR_wflow.db -v 10 -c 201905200000 -t get_extrn_ics
```

where `-c` specifies the cycle date (first column of rocotostat output) and `-t` represents the task name (second column of rocotostat output). After using `rocotorewind`, the next time `rocotorun` is used to advance the workflow, the job will be resubmitted.

11.5 How do I change the grid?

To change the predefined grid, you need to modify the `PREDEF_GRID_NAME` variable in the `config.sh` script which the user has created to generate an experiment configuration and workflow. Users can choose from one of three predefined grids for the SRW Application:

```
RRFS_CONUS_3km
RRFS_CONUS_13km
RRFS_CONUS_25km
```

An option also exists to create a user-defined grid, with information available in Chapter 6.

CHAPTER

TWELVE

ADDITIONAL ROCOTO INFORMATION

The tasks in the SRW Application (Table 4.6) are typically run using the Rocoto Workflow Manager. Rocoto is a Ruby program that interfaces with the batch system on an HPC system to run and manage dependencies between the tasks. Rocoto submits jobs to the HPC batch system as the task dependencies allow, and runs one instance of the workflow for a set of user-defined cycles. More information on Rocoto can be found at https://github.com/christopherwharrop/rocoto/wiki/documentation.

The SRW App workflow is defined in a Jinja-enabled Rocoto XML template called `FV3LAM_wflow.xml`, which resides in the `regional_workflow/ufs/templates` directory. When the `generate_FV3LAM_wflow.sh` script is run, the `fill_jinja_template.py` script is called, and the parameters in the template file are filled in. The completed file contains the workflow task names, parameters needed by the job scheduler, and task interdependencies. The generated XML file is then copied to the experiment directory: `$EXPTDIR/FV3LAM_wflow.xml`.

There are a number of Rocoto commands available to run and monitor the workflow and can be found in the complete Rocoto documentation. Descriptions and examples of commonly used commands are discussed below.

12.1 rocotorun

The `rocotorun` command is used to run the workflow by submitting tasks to the batch system. It will automatically resubmit failed tasks and can recover from system outages without user intervention. An example is:

```
rocotorun -w /path/to/workflow/xml/file -d /path/to/workflow/database/file -v 10
```

where

- `-w` specifies the name of the workflow definition file. This must be an XML file.
- `-d` specifies the name of the database file that is to be used to store the state of the workflow. The database file is a binary file created and used only by Rocoto and need not exist prior to the first time the command is run.
- `-v` (optional) specified level of verbosity. If no level is specified, a level of 1 is used.

From the `$EXPTDIR` directory, the `rocotorun` command for the workflow would be:

```
rocotorun -w FV3LAM_wflow.xml -d FV3LAM_wflow.db
```

It is important to note that the `rocotorun` process is iterative; the command must be executed many times before the entire workflow is completed, usually every 2-10 minutes. This command can be placed in the user's crontab and cron will call it with a specified frequency. More information on this command can be found at https://github.com/christopherwharrop/rocoto/wiki/documentation.

The first time the `rocotorun` command is executed for a workflow, the files `FV3LAM_wflow.db` and `FV3LAM_wflow_lock.db` are created. There is usually no need for the user to modify these files. Each time this command is executed, the last known state of the workflow is read from the `FV3LAM_wflow.db` file, the batch system is queried, jobs are submitted for tasks whose dependencies have been satisfied, and the current state of the workflow is saved in `FV3LAM_wflow.db`. If there is a need to relaunch the workflow from scratch, both database files can be deleted, and the workflow can be run using `rocotorun` or the launch script `launch_FV3LAM_wflow.sh` (executed multiple times as described above).

12.2 rocotostat

`rocotostat` is a tool for querying the status of tasks in an active Rocoto workflow. Once the workflow has been started with the `rocotorun` command, Rocoto can also check the status of the workflow using the `rocotostat` command:

```
rocotostat -w /path/to/workflow/xml/file -d /path/to/workflow/database/file
```

Executing this command will generate a workflow status table similar to the following:

```
       CYCLE                    TASK                       JOBID                STATE      ␣
↪EXIT STATUS     TRIES      DURATION
==========================================================================================================================
201907010000               make_grid                      175805               QUEUED       ␣
↪       -         0            0.0
201907010000               make_orog                           -                    -       ␣
↪       -         -              -
201907010000          make_sfc_climo                           -                    -       ␣
↪       -         -              -
201907010000           get_extrn_ics        druby://hfe01:36261          SUBMITTING       ␣
↪       -         0            0.0
201907010000          get_extrn_lbcs        druby://hfe01:36261          SUBMITTING       ␣
↪       -         0            0.0
201907010000                make_ics                           -                    -       ␣
↪       -         -              -
201907010000               make_lbcs                           -                    -       ␣
↪       -         -              -
201907010000                run_fcst                           -                    -       ␣
↪       -         -              -
201907010000           run_post_f000                           -                    -       ␣
↪       -         -              -
201907010000           run_post_f001                           -                    -       ␣
↪       -         -              -
```

(continues on next page)

(continued from previous page)

```
201907010000           run_post_f002                    -                    -        ␣
↪       -         -              -
201907010000           run_post_f003                    -                    -        ␣
↪       -         -              -
201907010000           run_post_f004                    -                    -        ␣
↪       -         -              -
201907010000           run_post_f005                    -                    -        ␣
↪       -         -              -
201907010000           run_post_f006                    -                    -        ␣
↪       -         -              -
```

This table indicates that the `make_grid` task was sent to the batch system and is now queued, while the `get_extrn_ics` and `get_extrn_lbcs` tasks for the `201907010000` cycle are in the process of being submitted to the batch system.

Note that issuing a `rocotostat` command without an intervening `rocotorun` command will not result in an updated workflow status table; it will print out the same table. It is the `rocotorun` command that updates the workflow database file (in this case `FV3LAM_wflow.db`, located in `$EXPTDIR`); the `rocotostat` command reads the database file and prints the table to the screen. To see an updated table, the `rocotorun` command must be executed followed by the `rocotostat` command.

After issuing the `rocotorun` command several times (over the course of several minutes or longer, depending on your grid size and computational resources), the output of the `rocotostat` command should look like this:

```
       CYCLE                    TASK                   JOBID                STATE      ␣
↪EXIT STATUS      TRIES      DURATION
==========================================================================================================================
201907010000               make_grid                  175805            SUCCEEDED      ␣
↪       0          1          10.0
201907010000               make_orog                  175810            SUCCEEDED      ␣
↪       0          1          27.0
201907010000           make_sfc_climo                 175822            SUCCEEDED      ␣
↪       0          1          38.0
201907010000            get_extrn_ics                 175806            SUCCEEDED      ␣
↪       0          1          37.0
201907010000           get_extrn_lbcs                 175807            SUCCEEDED      ␣
↪       0          1          53.0
201907010000                make_ics                  175825            SUCCEEDED      ␣
↪       0          1          99.0
201907010000               make_lbcs                  175826            SUCCEEDED      ␣
↪       0          1          90.0
201907010000                run_fcst                  175937              RUNNING      ␣
↪       -          0           0.0
201907010000            run_post_f000                    -                    -        ␣
↪       -          -              -
201907010000            run_post_f001                    -                    -        ␣
↪       -          -              -
201907010000            run_post_f002                    -                    -        ␣
↪       -          -              -
201907010000            run_post_f003                    -                    -        ␣
↪       -          -              -
```

(continues on next page)

(continued from previous page)

```
201907010000           run_post_f004                    -                   -        ␣
↪      -         -             -
201907010000           run_post_f005                    -                   -        ␣
↪      -         -             -
201907010000           run_post_f006                    -                   -        ␣
↪      -         -             -
```

When the workflow runs to completion, all tasks will be marked as SUCCEEDED. The log files from the tasks are located in `$EXPTDIR/log`. If any tasks fail, the corresponding log file can be checked for error messages. Optional arguments for the `rocotostat` command can be found at https://github.com/christopherwharrop/rocoto/wiki/documentation.

12.3 rocotocheck

Sometimes, issuing a `rocotorun` command will not cause the next task to launch. `rocotocheck` is a tool that can be used to query detailed information about a task or cycle in the Rocoto workflow. To determine the cause of a particular task not being submitted, the `rocotocheck` command can be used from the `$EXPTDIR` directory as follows:

```
rocotocheck -w /path/to/workflow/xml/file -d /path/to/workflow/database/ file -c␣
↪YYYYMMDDHHMM -t taskname
```

where

- `-c` is the cycle to query
- `-t` is the task name

A specific example is:

```
rocotocheck -w FV3LAM_wflow.xml -d FV3LAM_wflow.db -v 10 -c 201907010000 -t run_fcst
```

This will result in output similar to the following:

```
Task: run_fcst
  account: gsd-fv3
  command: /scratch2/BMC/det/$USER/ufs-srweather-app/regional_workflow/ush/load_modules_run_
↪task.sh "run_fcst" "/scratch2/BMC/det/$USER/ufs-srweather-app/regional_workflow/jobs/
↪JREGIONAL_RUN_FCST"
  cores: 24
  final: false
  jobname: run_FV3
  join: /scratch2/BMC/det/$USER/expt_dirs/test_community/log/run_fcst_2019070100.log
  maxtries: 3
  name: run_fcst
  nodes: 1:ppn=24
  queue: batch
  throttle: 9999999
  walltime: 04:30:00
```

(continues on next page)

(continued from previous page)

```
  environment
    CDATE ==> 2019070100
    CYCLE_DIR ==> /scratch2/BMC/det/$USER/UFS_CAM/expt_dirs/test_community/2019070100
    PDY ==> 20190701
    SCRIPT_VAR_DEFNS_FP ==> /scratch2/BMC/det/$USER/expt_dirs/test_community/var_defns.sh
  dependencies
    AND is satisfied
      make_ICS_surf_LBC0 of cycle 201907010000 is SUCCEEDED
      make_LBC1_to_LBCN of cycle 201907010000 is SUCCEEDED

Cycle: 201907010000
  Valid for this task: YES
  State: active
  Activated: 2019-10-29 18:13:10 UTC
  Completed: -
  Expired: -

Job: 513615
  State:  DEAD (FAILED)
  Exit Status:  1
  Tries:  3
  Unknown count:  0
  Duration:  58.0
```

This shows that although all dependencies for this task are satisfied (see the dependencies section, highlighted above), it cannot run because its `maxtries` value (highlighted) is 3. Rocoto will attempt to launch it at most 3 times, and it has already been tried 3 times (the `Tries` value, also highlighted).

The output of the `rocotocheck` command is often useful in determining whether the dependencies for a given task have been met. If not, the dependencies section in the output of `rocotocheck` will indicate this by stating that a dependency "is NOT satisfied".

12.4 rocotorewind

`rocotorewind` is a tool that attempts to undo the effects of running a task and is commonly used to rerun part of a workflow that has failed. If a task fails to run (the STATE is DEAD), and needs to be restarted, the `rocotorewind` command will rerun tasks in the workflow. The command line options are the same as those described in the `rocotocheck` section 12.3, and the general usage statement looks like:

```
rocotorewind -w /path/to/workflow/xml/file -d /path/to/workflow/database/ file -c␣
↪YYYYMMDDHHMM -t taskname
```

Running this command will edit the Rocoto database file `FV3LAM_wflow.db` to remove evidence that the job has been run. `rocotorewind` is recommended over `rocotoboot` for restarting a task, since `rocotoboot` will force a specific task to run, ignoring all dependencies and throttle limits. The throttle limit, denoted by the variable cyclethrottle in the `FV3LAM_wflow.xml` file, limits how many

cycles can be active at one time. An example of how to use this command to rerun the forecast task from `$EXPTDIR` is:

```
rocotorewind -w FV3LAM_wflow.xml -d FV3LAM_wflow.db -v 10 -c 201907010000 -t run_fcst
```

12.5 rocotoboot

`rocotoboot` will force a specific task of a cycle in a Rocoto workflow to run. All dependencies and throttle limits are ignored, and it is generally recommended to use `rocotorewind` instead. An example of how to use this command to rerun the `make_ics` task from `$EXPTDIR` is:

```
rocotoboot -w FV3LAM_wflow.xml -d FV3LAM_wflow.db -v 10 -c 201907010000 -t make_ics
```

CHAPTER

THIRTEEN

GLOSSARY

CCPP A forecast-model agnostic, vetted collection of codes containing atmospheric physical parameterizations and suites of parameterizations for use in Numerical Weather Prediction (NWP) along with a framework that connects the physics to the host forecast model.

chgres_cube The preprocessing software used to create initial and boundary condition files to "coldstart" the forecast model.

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

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

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

NCEPLIBS The software libraries created and maintained by *NCEP* that are required for running *chgres_cube*, the UFS Weather Model, and *UPP*.

NCEPLIBS-external A collection of third-party libraries required to build *NCEPLIBS*, *chgres_cube*, the UFS Weather Model, and *UPP*.

NCL An interpreted programming language designed specifically for scientific data analysis and visualization. More information can be found at https://www.ncl.ucar.edu.

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

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

UFS The Unified Forecast System is a community-based, coupled comprehensive Earth modeling system consisting of several applications (apps). These apps span regional to global domains and sub-hourly to seasonal time scales. The UFS is designed to support the Weather Enterprise and to be the source system for NOAA's operational numerical weather prediction applications. More information can be found at http://ufs-dev.rap.ucar.edu/index.html.

UFS_UTILS A collection of codes used by multiple *UFS* apps (e.g. the UFS Short-Range Weather App, the UFS Medium-Range Weather App). The grid, orography, surface climatology, and

initial and boundary condition generation codes used by the UFS Short-Range Weather App are all part of this collection.

UPP The Unified Post Processor is software developed at *NCEP* and used operationally to post-process raw output from a variety of *NCEP*'s NWP models, including the FV3.

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

BIBLIOGRAPHY

[BAB+ed] T.L. Black, J.A. Abeles, B.T. Blake, D. Jovic, E. Rogers, X. Zhang, E.A. Aligo, L.C. Dawson, Y. Lin, E. Strobach, P.C. Shafran, and J.R. Carley. A limited area modeling capability for the finite-volume cubed-sphere (fv3) dynamical core. *Monthly Weather Review*, Submitted.

INDEX

C

CCPP, 103
chgres_cube, 103

F

FV3, 103

G

GRIB2, 103

N

NCEP, 103
NCEPLIBS, 103
NCEPLIBS-external, 103
NCL, 103
NEMS, 103
NEMSIO, 103

U

UFS, 103
UFS_UTILS, 103
UPP, 104

W

Weather Model, 104