NOAA Technical Memorandum OAR GSD-45
doi:10.7289/V5BC3WG5

COMMUNITY HWRF USERS GUIDE V3.6A

October 2014

THE DEVELOPMENT TESTBED CENTER

C. Holt
L. Bernardet
T. Brown
R. Yablonsky

Earth System Research Laboratory
Global System Division
Boulder, Colorado
October 2014

noaa NATIONAL OCEANIC AND ATMOSPHERIC ADMINISTRATION / Office of Oceanic and Atmospheric Research

NOAA Technical Memorandum OAR GSD-45
doi:10.7289/V5BC3WG5

COMMUNITY HWRF USERS GUIDE V3.6A

Christina Holt[1]
Ligia Bernardet[1]
Timothy Brown[1]
Richard Yablonsky[2]

[1] Cooperative Institute for Research in Environmental Sciences (CIRES), Developmental Testbed Center and NOAA/ESRL/GSD

[2] University of Rhode Island

UNITED STATES DEPARTMENT OF COMMERCE

Penny Pritzker
Secretary

NATIONAL OCEANIC AND ATMOSPHERIC ADMINISTRATION

Dr. Kathryn Sullivan
Acting Under Secretary for Oceans
And Atmosphere/acting Administrator

Office of Oceanic and Atmospheric Research

Dr. Robert Detrick
Assistant Administrator

Community HWRF Users' Guide V3.6a

September 2014
(Revised October 2014)

Christina Holt, Ligia Bernardet, and Timothy Brown
NOAA/ESRL/GSD, Developmental Testbed Center and CIRES/CU

Richard Yablonsky
University of Rhode Island

Please send questions to: wrfhelp@ucar.edu

Contents

Preface

Meaning of typographic changes and symbols

Table 1 describes the type changes and symbols used in this book.

Typeface or Symbol	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.bashrc` Use `ls -a` to list all files. `host$ You have mail!.`
`AaBbCc123`	What you type, contrasted with on-screen computer output	`host$` **`su`**
`AaBbCc123`	Command line placeholder: replace with a real name or value	To delete a file, type `rm` *`filename`*

Table 1: Typographic Conventions

1

HWRF System Introduction

1.1 HWRF System Overview

The Weather Research and Forecast (WRF) system for hurricane prediction (HWRF) is an operational model implemented at the National Centers for Environmental Prediction (NCEP) of the National Weather Service (NWS) to provide numerical guidance to the National Hurricane Center for the forecasting of tropical cyclones' track, intensity, and structure. HWRF v3.6a and this Users' Guide contain the capabilities of the operational 2014 implementation of HWRF.

The HWRF model is a primitive equation non-hydrostatic coupled atmosphere-ocean model with the atmospheric component formulated with 61 levels in the vertical, and a 2 hPa model top. The atmospheric model uses the Non-hydrostatic Mesoscale Model (NMM) dynamic core of the WRF model (WRF-NMM), with a parent and two nest domains. The parent domain covers roughly 80° x 80° on a rotated latitude/longitude E-staggered grid. The location of the parent domain is determined based on the initial position of the storm and on the National Hurricane Center (NHC) forecast of the 72-h position, if available. The middle nest domain, of about 12° x 12°, and the inner nest domain, of about 7.1° x 7.1°, move along with the storm using two-way interactive nesting. The stationary parent domain has a grid spacing of 0.18° (about 27 km) while the middle nest spacing is 0.06° (about 9 km) and the inner nest spacing is 0.02° (about 3 km). The dynamic time steps are 45, 15, and 5 s, respectively, for the parent, middle nest, and inner nest domains.

The model physics originated primarily from the Geophysical Fluid Dynamics Laboratory (GFDL) hurricane model, and includes a simplified Arakawa-Schubert scheme for cumulus parameterization and a Ferrier cloud microphysics package for explicit moist physics. The vertical diffusion scheme is based on Troen and Mahrt's non-local scheme. The Monin-

Obukhov scheme is used for surface flux calculations, which also employs an improved air-sea momentum flux parameterization in strong wind conditions, and a one-layer slab land model. Radiation effects are evaluated by the GFDL scheme, which includes diurnal variations and interactive effects of clouds. The HWRF physics includes parameterization of dissipative heating.

Model initialization is comprised of both a vortex improvement procedure and data assimilation. The NCEP Global Forecast System (GFS) analysis is used to generate the initial conditions (ICs) for the hurricane model parent domain in the operational configuration. On the inner 9-km and 3-km nests, the NCEP Global Data Assimilation System (GDAS) 6-hour forecast initialized 6 hours prior to the HWRF analysis is interpolated to the appropriate grid and is used as the first guess.

The analysis is modified by first separating the vortex and environment fields of the respective first guess on each domain, i.e. the GFS vortex is separated from the environment on the parent domain, the GDAS vortex is removed from the environment on the inner domains. A new vortex is then incorporated onto the environment field. The new vortex that is added to the environment depends on the observed intensity of the cyclone, and on the existence of a previous HWRF forecast. The new vortex may derive from a bogus procedure, from the 6-h forecast of the HWRF model initialized 6-h previously, or from GDAS. In any case, the vortex is modified so that the initial storm position, structure, and intensity conform to the NHC storm message. A new feature for HWRF v3.6a allows for seamlessly cycling from NHC areas of investigation (Invests) to numbered storms.

The first guess with an improved vortex is modified using the HWRF Data Assimilation System (HDAS) by ingesting observations in a three-dimensional (3D) hybrid ensemble-variational (VAR) data assimilation system called Gridpoint Statistical Interpolation (GSI). The ensemble information is obtained from the GFS ensemble. HWRF assimilates conventional observations, reconnaissance dropsondes, tail Doppler Radar, and satellite observations. Satellite observations utilized by HWRF include radiances from infrared instruments (HIRS, AIRS, IASI, and GOES sounders) and microwave instruments (AMSU-A, MHS, and ATMS), satellite-derived wind, and Global Positioning System (GPS) radio occultation bending angle. First Guess at Appropriate Time (FGAT) is used to make sure that GSI uses innovations calculated by comparing observations with corresponding model analysis fields valid at the time when the observations were collected. The GFS forecast fields are used to provide lateral boundary conditions every 6 hours during the forecast.

The time integration is performed with a forward-backward scheme for fast waves, an implicit scheme for vertically propagating sound waves and the Adams-Bashforth scheme for horizontal advection and for the Coriolis force. In the vertical, the hybrid pressure-sigma coordinate is used. Horizontal diffusion in based on a 2nd order Smagorinsky-type, for more details see the HWRF Scientific Documentation at `http:\dtcenter.org/HurrWRF/users/`.

The Community HWRF model can be used for any oceanic basin. In the North Atlantic and Eastern North Pacific basins, for which NHC is responsible, the atmospheric model is coupled with an ocean model, the MPIPOM-TC. This ocean model implements the Message Passaging Interface (MPI) to run a parallel version of the Princeton Ocean Model (POM) for Tropical Cyclones (POM-TC). The POM was developed at Princeton University. At the

University of Rhode Island (URI), the POM was coupled to the GFDL and WRF models. In both basins, MPIPOM-TC is run in three dimensions with 1/12° (approximately 9 km) horizontal grid spacing. The MPIPOM-TC is configured with 23 vertical levels in the North Atlantic and Eastern North Pacific basins. In the other basins, HWRF is configured to run with its atmospheric component only.

The MPIPOM-TC is initialized by a diagnostic and prognostic spin up of the ocean circulations using climatological ocean data. For storms located in the western part of the Atlantic basin, the initial conditions are enhanced with real-time sea surface temperature (SST), sea surface height data, and the assimilation of oceanic "features". During the ocean spin up, realistic representations of the structure and positions of the Loop Current, Gulf Stream, and warm- and cold-core eddies are incorporated using a features-based data assimilation technique developed at URI.

The atmospheric and oceanic components are interactively coupled with a Message Passing Interface (MPI)-based coupler, which was developed at NCEP's Environmental Modeling Center (EMC). The atmospheric and oceanic components exchange information through the coupler; the ocean sends the SST to the atmosphere; the atmosphere receives the SST and sends the surface fluxes, including sensible heat flux, latent heat flux and short-wave radiation to the ocean, and so on. The frequency of information exchange is 9 minutes.

HWRF is suitable for use in tropical applications including real-time NWP, forecast research, physics parameterization research, air-sea coupling research, and teaching. Additionally, HWRF v3.6a includes the capability to perform idealized tropical cyclone simulations. The HWRF system support to the community by the Developmental Testbed Center (DTC) includes the following four main modules.

- HWRF atmospheric components
  - WRF-NMM (which has tropical physics schemes and a vortex-following moving nest)
  - WRF Preprocessing System (WPS)
  - Vortex initialization
  - GSI
  - Unified Post-Processor (UPP)
  - GFDL Vortex Tracker
- HWRF oceanic components
  - MPIPOM-TC model
  - Ocean initialization
- Atmosphere-Ocean Coupler
- HWRF Run Module

New in HWRF v3.6a is a complete rewrite of the scripts used to run HWRF. The new set of scripts, written in the Python language, constitutes unification between the DTC and EMC scripts. They are currently being employed in developmental HWRF configurations undergoing testing at EMC and are scheduled for first operational implementation at NCEP in the 2015 version of HWRF. Due to the novel nature of these scripts, users should expect some additions and changes in the upcoming months. Those will be posted in the Frequently-Asked-Questions (FAQ) or Known Issues sections of the DTC website at `http://dtcenter.org/HurrWRF/users`.

1.2 HWRF Development and Support

All HWRF components are under the Subversion revision control system. The code repositories are hosted and maintained as community codes at the National Center for Atmospheric Research (NCAR), except for GSI, which is housed at the National Oceanic and Atmospheric Administration (NOAA). An HWRF code management protocol has been established for proposing HWRF-related modifications to the software, whether the modifications are simply updates to the current features, bug fixes, or the addition of new features. HWRF code development must be carried out in the branches of the repositories and frequently synchronized with the trunks. Proposed software modifications must be thoroughly tested prior to being committed to the code repository to protect the integrity of the evolving code base.

HWRF is being actively developed and advanced. In the future, more components will be coupled into the HWRF system, including wave, hydrology, storm surge, and inundation components.

The HWRF modeling system software is in the public domain and is freely available for community use. Information about obtaining the codes, datasets, documentations, and tutorials can be found at `http://www.dtcenter.org/HurrWRF/users` and in the following chapters of this Users' Guide. Direct all questions to wrfhelp@ucar.edu. Please also contact this email if you would like more information on the protocols for HWRF development.

1.3 HWRF Source Code Directory Structure

The HWRF system source code has the following nine components.

- WRF Atmospheric Model
- WPS
- UPP
- GSI
- HWRF Utilities
- MPIPOM-TC
- GFDL Vortex Tracker
- NCEP Atmosphere-Ocean Coupler
- HWRF Run Component

The code for all components can be obtained by downloading the following tar files from the DTC website (see Chapter 2 for installation information).

- `HWRF_v3.6a_WRFV3.tar.gz`
- `HWRF_v3.6a_WPSV3.tar.gz`
- `HWRF_v3.6a_UPP.tar.gz`
- `HWRF_v3.6a_GSI.tar.gz`
- `HWRF_v3.6a_hwrf-utilities.tar.gz`
- `HWRF_v3.6a_gfdl-vortextracker.tar.gz`
- `HWRF_v3.6a_ncep-coupler.tar.gz`

- HWRF_v3.6a_pomtc.tar.gz
- HWRF_v3.6a_hwrfrun.tar.gz

First expand HWRF_v3.6a_hwrfrun.tar.gz in a user-defined HWRF top directory. Once completed, change directory to ${SCRATCH}/hwrfrun/sorc to expand the remaining tar files. The following directories should be present once all files are expanded.

- WRFV3 – Weather Research and Forecasting model
- WPSV3 – WRF Preprocessing System
- UPP – Unified Post-Processor
- GSI – Gridpoint Statistical Interpolation 3D-VAR data assimilation
- hwrf-utilities – Vortex initialization, utilities, tools, and supplemental libraries
- gfdl-vortextracker – Vortex tracker
- ncep-coupler – Ocean/atmosphere coupler
- pomtc – Tropical cyclone version of MPIPOM

For the remainder of this document, we assume that the tar files have been expanded under ${SCRATCH}/hwrfrun/sorc, where ${SCRATCH} is an environment variable that describes the location of the directory in which you installed the HWRF components.

The directory trees for these nine components are listed as follows. Note that these are the directories after the code is compiled. Before compilation not all of these directories are present.

1.3.1 HWRF Utilities Programs and Scripts

```
hwrf-utilities/
├── arch/ ..... architecture compiling options
├── clean ..... script to clean created files and executables
├── compile ..... script to compile component
├── configure ..... script to create the configure file
├── exec/ ..... executables
├── libs/ ..... libraries: blas, sp, sfcio, bacio, w3, and bufr
├── makefile ..... top level makefile
├── pure-openmp.inc
└── tools/ ..... source code for tools to run the HWRF system
    ├── Makefile ..... makefile for ocean model code
    ├── bufr_remorest/
    ├── grbindex/
    ├── hwrf_afos/
    ├── hwrf_atcf_to_stats/
    ├── hwrf_aux_rw/
    ├── hwrf_bdy_update/
    ├── hwrf_binary_grads/
    ├── hwrf_bin_io/
    ├── hwrf_blend_gsi/
    └── hwrf_combinetrack/
```

```
├── hwrf_data_flag/
├── hwrf_data_remv/
├── hwrf_gettrk/
├── hwrf_gridgenfine/
├── hwrf_htcfstats/
├── hwrf_netcdf_grads/
├── hwrf_nhc_products/
├── hwrf_prep_hybrid/
├── hwrf_read_indi/
├── hwrf_readtdrstmid/
├── hwrf_readtdrtime/
├── hwrf_supvit/
├── hwrf_swath/
├── hwrf_wrfout_newtime/
├── mdate/
├── mpiserial/
├── ndate/
├── nhour/
├── serpoe/
├── wave_sample/
└── wgrib/
└── vortex_init ..... source code for the HWRF vortex initialization
    ├── Makefile ..... makefile for vortex_init code
    ├── hwrf_anl_bogus/
    ├── hwrf_anl_cs/
    ├── hwrf_anl_step2/
    ├── hwrf_create_nest/
    ├── hwrf_create_trak_fnl/
    ├── hwrf_create_trak_guess/
    ├── hwrf_diffwrf_3dvar/
    ├── hwrf_guess/
    ├── hwrf_pert_ct/
    ├── hwrf_set_ijstart/
    ├── hwrf_split/
    ├── interpolate/
    └── merge_nest/
```

1.3.2 MPIPOM-TC Ocean Model

```
pomtc
├── arch/ ..... architecture compiling options
├── clean ..... script to clean created files and executables
├── compile ..... script to compile component
└── configure ..... script to create the configure file
    ├── makefile ..... makefile for tools code
    ├── ocean_exec/ ..... ocean model executables
    └── ocean_init/ ..... source code for generating ocean model initial conditions
        └── Makefile ..... makefile for the ocean initialization code
```

```
    ├── date2day/
    ├── day2date/
    ├── fbtr/
    ├── gdm3/
    ├── getsst/
    ├── idtr/
    ├── ncda/
    ├── pom/
    ├── sharp_mcs_rf_l2m_rmy5/
    └── tran/
├── ocean_main/ ..... source code for the ocean forecast model
    ├── Makefile ..... makefile for the ocean model code
    └── pom/
└── ocean_plot/ ..... software used to plot ocean output
```

1.3.3 NCEP Coupler

```
ncep-coupler/
├── arch/ ..... architecture compiling options
├── clean ..... script to clean created files and executables
├── compile ..... script to compile component
├── configure ..... script to create the configure file
├── cpl_exec/ ..... coupler executable
├── hwrf_wm3c/ ..... source code for the coupler
└── makefile ..... top level makefile
```

1.3.4 GFDL Vortex Tracker

```
gfdl-vortextracker/
├── arch/ ..... architecture compiling options
├── clean ..... script to clean created files and executables
├── compile ..... script to compile component
├── configure ..... script to create the configure file
├── makefile ..... top level makefile
├── trk_exec/ ..... executables
├── trk_plot/ ..... plot scripts and data
└── trk_src/ ..... source code for the vortex tracker
```

1.3.5 WRFV3 – Atmospheric Model

```
WRFV3/
├── Makefile ..... makefile used to compile WRFV3
├── Registry/ ..... WRFV3 Registry files
├── arch/ ..... architecture compiling options
├── clean ..... script to clean created files and executables
├── compile ..... script to compile component
└── configure ..... script to create the configure file
```

```
|
|__ dyn/_em/ ..... Advanced Research WRF dynamic modules, not used by HWRF
|__ dyn_exp/ ..... 'toy' dynamic core, not used by HWRF
|__ dyn_nmm/ ..... WRF-NMM dynamic modules
|__ external/ ..... external packages including ocean coupler interface
|__ frame/ ..... modules for WRF framework
|__ hydro/ ..... hydrology module, not used by HWRF
|__ inc/ ..... include files
|__ main/ ..... WRF main routines, such as wrf.F
|__ phys/ ..... physics modules
|__ run/ ..... run directory, not used by HWRF
|__ share/ ..... modules for WRF mediation layer and WRF I/O
|__ test/ ..... sub-dirs for specific configurations of WRF, such as idealized HWRF
|__ tools/ ..... tools directory
|__ var/ ..... WRF-Var, not used by HWRF
```

Refer to the WRF-NMM Users' Guide for more information. The WRF-NMM Users' Guide is available online at:
`http://www.dtcenter.org/wrf-nmm/users/docs/user_guide/V3/users_guide_nmm_chap1-7.pdf`.

1.3.6 WPSV3 – WRF Preprocessor

```
WPSV3/
|__ arch/ ..... architecture compiling options
|__ clean ..... script to clean created files and executables
|__ compile ..... script to compile component
|__ configure ..... script to create the configure file
|__ geogrid/ ..... source code for geogrid.exe
|__ link_grib.csh ..... script to link input GRIB files, used in idealized simulations
|__ metgrid/ ..... source code for metgrid.exe
|__ test_suite/ ..... WPS test cases
|__ ungrib/ ..... source code for ungrib.exe
|__ util/ ..... utility programs for WPSV3
```

1.3.7 UPP – Unified Post-Processor

```
UPP/
|__ arch/ ..... architecture compiling options
|__ bin/
|__ clean ..... script to clean created files and executables
|__ compile ..... script to compile component
|__ configure ..... script to create the configure file
|__ makefile ..... top level makefile
|__ include/
|__ lib/
|__ parm/ ..... parameter files to control UPP performed, not used by HWRF
|__ scripts/ ..... sample scripts running UPP, not used by HWRF
```

- src/ ..... UPP and dependent libraries source codes

1.3.8 GSI – Gridpoint Statistical Interpolation

GSI/
- arch/ ..... architecture compiling options
- clean ..... script to clean created files and executables
- compile ..... script to compile component
- configure ..... script to create the configure file
- fix/ ..... fix files for GSI, not used by HWRF
- include/
- lib/
- makefile ..... top level makefile
- run/ ..... executables
- src/ ..... source codes
  - Makefile
  - libs/ ..... dependent libraries
  - main/ ..... GSI source code
- util/ ..... utilities, not used by HWRF

1.3.9 HWRF Run

hwrfrun/
- parm/ ..... files to configure HWRF experiment
- exec/ ..... empty directory where some executables will be linked
- sorc/ ..... empty directory where HWRF components' code should be placed
- scripts/ ..... Python scripts to run the HWRF components
- wrappers/ ..... ksh wrapper scripts used to run the Python scripts
- ush/ ..... Python modules
  - hwrf/ ..... HWRF-specific Python modules
  - produtils/ ..... HWRF-independent Python modules for generalization of platforms and systems
- nwport/ ..... extra utilities for HWRF used in operations.

2

Software Installation

2.1 Introduction

The DTC community HWRF system, which is based on the NOAA operational HWRF, consists of nine components.

- WRF Atmospheric Model
- WPS
- UPP
- GSI
- HWRF-utilities
- MPIPOM-TC
- GFDL Vortex Tracker
- NCEP Atmosphere-Ocean Coupler
- HWRF Run

The first three of these components are the traditional WRF components: WRF, WPS, and UPP. GSI is a 3D variational data assimilation code used for data assimilation, and the remaining four components are specific to the hurricane system itself, and as such are referred to as the hurricane components of the HWRF system.

This chapter discusses how to build the HWRF system. It starts in Section 2.2 by discussing where to find the source code. Section 2.3 covers the preferred directory structure and how to unpack the tar files. Section 2.4 covers the system requirements for building and running the components. Section 2.5 discusses the libraries included in the HWRF-utilities component. Section 2.6 covers building WRF-NMM for HWRF. The remaining sections are devoted to building each of the remaining components of the HWRF system.

2.2 Obtaining the HWRF Source Code

The HWRF hurricane system consists of nine components. All of these are available from the HWRF website. While most of these codes are also available from other community websites, the versions needed for HWRF should be acquired from the DTC HWRF website to ensure they are a consistent set.

All of the HWRF components can be obtained through the HWRF website,

`http://www.dtcenter.org/HurrWRF/users`,

by selecting the Download and HWRF System tabs on the left vertical menu. New users must first register before downloading the source code. Returning users need only provide their registration email address. A successful download produces nine tar files.

- `HWRF_v3.6a_WRFV3.tar.gz`
- `HWRF_v3.6a_WPSV3.tar.gz`
- `HWRF_v3.6a_UPP.tar.gz`
- `HWRF_v3.6a_GSI.tar.gz`
- `HWRF_v3.6a_hwrf-utilities.tar.gz`
- `HWRF_v3.6a_gfdl-vortextracker.tar.gz`
- `HWRF_v3.6a_ncep-coupler.tar.gz`
- `HWRF_v3.6a_pomtc.tar.gz`
- `HWRF_v3.6a_hwrfrun.tar.gz`

After downloading each of the component codes, the user should check the links to known issues and bug fixes to see if any code updates are required. You now have all the HWRF system components as gzipped tar files. The next section describes how to organize them.

2.3 Setting up HWRF

Although the HWRF scripts may be modified for any directory structure, in this discussion it is assumed that the HWRF system will be set up in a single flat directory structure. Because of the storage requirements necessary for the complete HWRF system setup, it typically will need to be located on a computer's "scratch" or "work" partition.

The tar files can be unpacked by use of the GNU gunzip command,

```
gunzip *.tar.gz
```

and the tar files extracted by running **`tar -xvf`** individually on each of the tar files. It is recommended that the User first unpack hwrfrun, which will create a directory `hwrfrun/` in `${SCRATCH}`. Then within `hwrfrun/sorc/` directory, unpack the remaining tar files.

Once unpacked, there should be eight source directories in `sorc/`.

- `WRFV3` – Weather Research and Forecasting model
- `WPSV3` – WRF Preprocessing System
- `UPP` – Unified Post-Processor
- `GSI` – Gridpoint statistical interpolation 3D var data assimilation
- `hwrf-utilities` – Vortex initialization, utilities, tools, and supplemental libraries
- `gfdl-vortextracker` – Vortex tracker
- `ncep-coupler` – Ocean/atmosphere coupler
- `pomtc` – Tropical cyclone version of MPIPOM

The next section covers the system requirements to build the HWRF system.

2.4 System Requirements, Libraries, and Tools

In practical terms, the HWRF system consists of a collection of Python modules, which runs a sequence of serial and parallel code executables. The source code for these executables is in the form of programs written in FORTRAN, FORTRAN 90, and C. In addition, the parallel executables require some flavor of MPI/OpenMP for the distributed memory parallelism, and the I/O relies on the netCDF I/O libraries. Beyond the standard scripts, the build system relies on use of the Perl scripting language, along with GNU make and date.

The basic requirements for building and running the HWRF system are listed below.

- FORTRAN 90+ compiler
- C compiler
- MPI v1.2+
- Perl
- netCDF v3.6+
- LAPACK and BLAS
- Python
- Parallel-netCDF
- PNG
- JasPer
- zlib

Because these tools and libraries are typically the purview of system administrators to install and maintain, they are considered part of the basic system requirements.

2.4.1 Compilers

The DTC community HWRF system has been tested on a variety of computing platforms. Currently the HWRF system is actively supported on Linux computing platforms using both the Intel and PGI Fortran compilers. Unforeseen build issues may occur when using older compiler versions. Typically the best results come from using the most recent version of a compiler. The known issues section of the community website provides the complete list of

compiler versions currently supported.

While the community HWRF build system provides legacy support for the IBM AIX platforms, the unavailability of AIX test platforms means all AIX support is cursory at best.

2.4.2 netCDF, pnetCDF, and MPI

The HWRF system requires a number of support libraries not included with the source code. Many of these libraries may be part of the compiler installation, and are subsequently referred to as system libraries. For our needs, the most important of these libraries are netCDF, pnetCDF, and MPI.

An exception to the rule of using the most recent version of code, libraries, and compilers is the netCDF library. The HWRF system I/O requires the most recent V3 series of the library. Version 4 of netCDF diverges significantly from version 3, and is not supported. The preferred version of the library is netCDF v3.6+. The netCDF libraries can be downloaded from the Unidata website.

`http://www.unidata.ucar.edu`

Typically, the netCDF library is installed in a directory that is included in the users path such as `/usr/local/lib`. When this is not the case, the environment variables `NETCDF` and `PNETCDF` can be set to point to the location of the library.

For csh/tcsh, the path can be set with the following command.

```
setenv NETCDF path_to_netcdf_library/
setenv PNETCDF path_to_pnetcdf_library/
```

For bash/ksh, the equivalent command is as follows.

```
export NETCDF=path_to_netcdf_library/
export PNETCDF=path_to_pnetcdf_library/
```

It is crucial that system libraries, such as netCDF, be built with the same FORTRAN compiler, compiler version, and compatible flags, as used to compile the remainder of the source code. This is often an issue on systems with multiple FORTRAN compilers, or when the option to build with multiple word sizes (e.g. 32-bit vs. 64-bit addressing) is available.

Many default Linux installations include a version of netCDF. Typically this version is only compatible with code compiled using gcc. To build the HWRF system, a version of the library must be built using your preferred compiler and with both C and FORTRAN bindings. If you have any doubts about your installation, ask your system administrator.

Building and running the HWRF distributed memory parallel executables requires that a version of the MPI library be installed. Just as with the netCDF library, the MPI library must be built with the same FORTRAN compiler, and use the same word size option flags,

as the remainder of the source code. Installing MPI on a system is typically a job for the system administrator and will not be addressed here. If you are running HWRF on a computer at a large center, check the machine's documentation before you ask the local system administrator.

2.4.3 LAPACK and BLAS

The LAPACK and BLAS libraries are open source mathematics libraries for solving linear algebra problems. The source code for these libraries is freely available to download from NETLIB at

`http://www.netlib.org/lapack/`.

Most commercial compilers provide their own optimized versions of these routines. These optimized versions of BLAS and LAPACK provide superior performance to the open source versions.

On Linux systems, HWRF supports both the Intel ifort and PGI pgf90 Fortran compilers. The Intel compiler has its own optimized version of the BLAS and LAPACK routines called the Math Kernel Library or MKL. The MKL libraries provide **most** of the LAPACK and BLAS routines needed by the HWRF system. The PGI compiler typically comes with its own version of the BLAS and LAPACK libraries. Again, the PGI version of BLAS and LAPACK contains **most** of the routines needed by HWRF. For PGI these libraries are loaded automatically. Since the vender versions of the libraries are often incomplete, a copy of the full BLAS library is provided with the HWRF-utilities component. The build system links to this version of the libraries last.

On the IBM machines, the AIX compiler is often, but not always, installed with the Engineering and Scientific Subroutine Libraries or ESSL. In part, the ESSL libraries are highly optimized parallel versions of many of the LAPACK and BLAS routines. The ESSL libraries provide all of the necessary linear algebra library routines needed by the HWRF system.

2.5 Included Libraries

For convenience in building HWRF-utilities, the MPIPOM-TC, and the GFDL Vortex Tracker components, the HWRF-utilities component includes a number of libraries in the **`hwrf-utilities/libs/src/`** directory. The following libraries are built automatically when the HWRF-utilities component is built.

- BACIO
- BLAS
- BUFR
- SFCIO

- SIGIO
- SP
- W3
- G2

The other components, WPS, WRF, UPP, and GSI, come with their own versions of many of these libraries, but typically they have been customized for that particular component and should not be used by the other components.

When the HWRF-utilities component is compiled, it starts by first building all the included libraries. The vortex initialization code contained in the HWRF-utilities component requires all of the above libraries except for the SFCIO library. In addition, it requires both the BLAS and LAPACK mathematical libraries when the IBM ESSL library is not included with the compiler installation.

The MPIPOM-TC component requires the SFCIO, SP and W3 libraries. In addition, the local copy of the BLAS library is required when the ESSL library is not included with the compiler installation. This is because the vender-supplied versions of BLAS are typically incomplete, and the local version supplements the vender version. Typically this is for any system other than IBM. The GFDL Vortex Ttracker component requires the BACIO and W3 libraries. The NCEP Coupler does not require any additional libraries.

2.5.1 Component Dependencies

The eight components of the HWRF system that contain source code have certain inter-dependencies. Many of the components depend on libraries produced by other components. For example, four of the components, WPS, UPP, GSI, and the HWRF-utilities, require linking to the WRF I/O API libraries to build. Since these I/O libraries are created as part of the WRF build, the WRF component must be built first. Once WRF is built, WPS, UPP, GSI, or the HWRF-utilities can be built in any order. Since building the HWRF-utilities produces the supplemental libraries needed by MPIPOM-TC and by the GFDL Vortex Tracker, the HWRF-utilities must be built before either of these components. The remaining component, the NCEP Coupler, can be built independently of any of the other components. The main system component dependency is as follows.

- WRF
  - WPS
  - UPP
  - GSI
  - HWRF-utilities
    - MPIPOM-TC (BLAS on Linux, sfcio, sp, w3)
    - GFDL Vortex Tracker (w3, bacio, G2)
- NCEP Coupler

2.6 Building WRF-NMM

The WRF code has a fairly sophisticated build mechanism. The package attempts to determine the machine where the code is being built, and then presents the user with supported build options on that platform. For example, on a Linux machine, the build mechanism determines whether the machine is 32-bit or 64-bit, prompts the user for the desired type of parallelism (such as serial, shared memory, distributed memory, or hybrid), and then presents a selection of possible compiler choices.

In addition, the user may choose to run WRF with either real or idealized input data. The idealized data case requires setting environment flags prior to compiling the code, which creates a unique executable that should only be run with the idealized data. See Section 2.6.3 for compiling WRF for ideal runs.

2.6.1 Set Environment Variables

To correctly configure WRF-NMM for the HWRF system, set the following additional environment variables beyond what WRF typically requires.

In C-Shell use the following commands.

```
setenv HWRF 1
setenv WRF_NMM_CORE 1
setenv WRF_NMM_NEST 1
setenv WRFIO_NCD_LARGE_FILE_SUPPORT 1
```

Add the following command for IBM AIX builds using C-Shell.

```
setenv IBM_REDUCE_BUG_WORKAROUND 1
```

In Bash shell, use the following commands.

```
export HWRF=1
export WRF_NMM_CORE=1
export WRF_NMM_NEST=1
export WRFIO_NCD_LARGE_FILE_SUPPORT=1
```

Add the following command for IBM AIX builds using Bash.

```
export IBM_REDUCE_BUG_WORKAROUND=1
```

These settings produce a version of WRF-NMM compatible with the HWRF system.

There is a bug in the IBM MPI implementation. Some MPI processes will get stuck in MPI_Reduce and not return until the PREVIOUS I/O server group finishes writing. When the environment variable `IBM_REDUCE_BUG_WORKAROUND=1`, a workaround is used that replaces the `MPI_Reduce` call with many `MPI_Send` and `MPI_Recv` calls that perform the

sum on the root of the communicator.

Note that setting the environment variable `WRF_NMM_NEST` to 1 does not preclude running with a single domain.

2.6.2 Configure and Compile WRF-NMM

To configure WRF-NMM, go to the top of the WRF directory (**cd ${SCRATCH}/hwrfrun/sorc/WRFV3**) and use the following command.

./configure

You will be presented with a list of build choices for your computer. These choices may include multiple compilers and parallelism options.

For Linux architectures, there are currently 55 options. For the HWRF system, only the distributed memory (dmpar) builds are recommended. Therefore as an example, the acceptable PGI options are 3, 7, 11, or 50 (shown below).

```
3.  Linux x86_64 i486 i586 i686, PGI compiler with gcc (dmpar)
7.  Linux x86_64, PGI compiler with pgcc, SGI MPT (dmpar)
11.  Linux x86_64, PGI accelerator compiler with gcc (dmpar)
50.  Linux x86_64 i486 i586 i686, PGI compiler with pgcc (dmpar)
```

The configure step for the WRF model is now completed. A file has been created in the WRF directory called `configure.wrf`. The compile options and paths in the `configure.wrf` file can be edited for further customization of the build process.

To build the WRF-NMM component enter the following command.

./compile nmm_real

In general, it is good practice to save the standard out and error to a log file for reference. In the csh/tcsh shell this can be done with the following command.

./compile nmm_real |& tee build.log

For the ksh/bash shell use the following command.

./compile nmm_real 2>& 1 | tee build.log

In both cases, the standard out and the standard error are sent to both the file `build.log` and to the screen. The approximate compile time varies according to the system being used and the aggressiveness of the optimization. On IBM AIX machines, the compiler optimization significantly slows down the build time, and it typically takes at least half an hour to complete. On most Linux systems, the WRF model typically compiles in around 20 minutes.

It is important to note that the commands **./compile -h** and **./compile** produce a listing of all of the available compile options, but only the nmm_real option is relevant to the HWRF system.

A successful compilation produces two executables listed below in the directory `main/`.

`real_nmm.exe`	WRF initialization
`wrf.exe`	WRF model integration

If a recompilation is necessary, a clean to remove all object files (except those in `external/`) should be completed first.

./clean

A complete clean is strongly recommended if the compilation failed, the Registry has been changed, or the configuration file is changed. To conduct a complete clean that removes all built files in all directories, as well as the `configure.wrf` use the "-a" option.

./clean -a

Further details on the HWRF atmospheric model, physics options, and running the model can be found in the Running HWRF chapter of the Users' Guide.

Complete details on building and running the WRF-NMM model are available in the WRF-NMM Users' Guide, which is available here:

`http://www.dtcenter.org/wrf-nmm/users/docs/user_guide/V3/users_guide_nmm_chap1-7.pdf`.

Should you experience difficulty building WRF while using the PGI compiler, a helpful guide for building WRF with PGI compilers on a 32-bit or 64-bit LINUX system can be found at:

`http://www.pgroup.com/resources/tips.htm`.

2.6.3 Configure and Compile: Idealized Tropical Cyclone WRF-NMM

The HWRF idealized tropical cyclone WRF-NMM component requires different executables than for the real case. The following section will describe how to build the executables for the idealized case.

Building the idealized component requires a slightly different configuration than for the standard WRF build outlined in Section 2.6.1. If a user has already built the standard WRFV3 and created `real_nmm.exe` and `wrf.exe`, and now wants to build WRFV3 for idealized tropical cyclone simulations, they first need to completely clean the previous build. This is done by running a

```
./clean -a
```

which removes ALL build files, including the executables, libraries, and the configure.hwrf. To correctly configure WRF-NMM for the HWRF idealized tropical cyclone simulation requires setting the additional environment variable `IDEAL_NMM_TC`. Several other variables must also be set.

In C-Shell use the following commands.

```
setenv WRF_NMM_CORE 1
setenv WRF_NMM_NEST 1
setenv HWRF 1
setenv IDEAL_NMM_TC 1
setenv WRFIO_NCD_LARGE_FILE_SUPPORT 1
```

The following commands should be used for bash/ksh.

```
export WRF_NMM_CORE=1
export WRF_NMM_NEST=1
export HWRF=1
export IDEAL_NMM_TC=1
export WRFIO_NCD_LARGE_FILE_SUPPORT=1
```

To configure WRF-NMM, go to the top of the WRF directory (**cd ${SCRATCH}/hwrfrun/sorc/WRFV3**) before issuing the following command.

```
./configure
```

You will be presented with a list of build choices for your computer. These choices may include multiple compilers and parallel options.

For Linux architectures, there are currently 51 options. For the HWRF system, only the distributed memory (dmpar) builds are recommended. Therefore as an example, the acceptable PGI options are 3, 7, 11, or 50 (shown below).

```
3.  Linux x86_64 i486 i586 i686, PGI compiler with gcc (dmpar)
7.  Linux x86_64, PGI compiler with pgcc, SGI MPT (dmpar)
11.  Linux x86_64, PGI accelerator compiler with gcc (dmpar)
50.  Linux x86_64 i486 i586 i686, PGI compiler with pgcc (dmpar)
```

The configure step for the WRF model is now completed. A file has been created in the WRF directory called `configure.wrf`. The compile options and paths in the `configure.wrf` file can be edited for further customization of the build process.

Once the configure step is complete, the code is compiled by including the target `nmm_tropical_cyclone` to the compile command.

```
./compile nmm_tropical_cyclone
```

A successful compilation produces two executables in the directory `main/`.

`ideal.exe`	WRF initialization
`wrf.exe`	WRF model integration

Note: The only compilation requirements for the idealized capability are WPS and WRF. If wanted, UPP may also be used. The components MPIPOM-TC and coupler, GSI, GFDL Vortex Tracker, and hwrf-utilities are not used in HWRF idealized tropical cyclone simulations.

2.7 Building HWRF-utilities

The `hwrf-utilities/` directory consists of an eclectic collection of source code and libraries. The libraries, which are provided in support of the MPIPOM-TC and the GFDL Vortex Tracker, include the BACIO, BLAS, BUFR, SIGIO, SFCIO, SP, and W3 libraries. In addition to these libraries, this component includes the source code for the vortex initialization routines and software tools such as the grbindex.

2.7.1 Set Environment Variables

The HWRF-utilities build requires that two path variables, `NETCDF` and `WRF_DIR`, be set to the appropriate paths. The netCDF library path `NETCDF` is required for building the WRF-NMM component, and its value should be appropriately set if that component compiled successfully. The `WRF_DIR` path variable should point to the WRF directory compiled in the previous section. You must first build WRF before compiling any of the other components.

If you have followed the directory structure suggested in Section 2.3, the `WRF_DIR` path should be set to `${SCRATCH}/hwrfrun/sorc/WRFV3`. In csh/tcsh, the variables may be set with two commands.

```
setenv NETCDF /absolute_path_to_appropriate_netCDF_library/
setenv WRF_DIR ${SCRATCH}/hwrfrun/sorc/WRFV3
```

For the ksh/bash shells, use the following two commands.

```
export NETCDF=/absolute_path_to_appropriate_netCDF_library/
export WRF_DIR=${SCRATCH}/hwrfrun/sorc/WRFV3
```

It is crucial that the Fortran compiler used to build the libraries (Intel, PGI, XLF, etc.) be the same as the compiler used to compile the source code. Typically, this is only an issue in two situations, on Linux systems having multiple compilers installed, and on systems where there is a choice between building the code with either 32-bit or 64-bit addressing.

2.7.2 Configure and Compile

To configure HWRF-utilities for compilation, from within the `hwrf-utilities` directory, type the following command.

`./configure`

The configure script checks the system hardware, and if the path variables are not set, asks for the correct paths to the netCDF libraries and the WRF build directory. It concludes by asking the user to choose a configuration supported by current machine architecture.

For Linux, seven options are available.

```
1.  Linux x86_64, PGI compiler w/LAPACK (dmpar)
2.  Linux x86_64, PGI compiler w/LAPACK, SGI MPT (dmpar)
3.  Linux x86_64, Intel compiler w/MKL (dmpar)
4.  Linux x86_64, Intel compiler w/MKL, SGI MPT (dmpar)
5.  Linux x86_64, Intel compiler w/MKL, IBM POE (dmpar)
6.  Linux x86_64, Intel compiler w/LAPACK (dmpar)
7.  Linux x86_64, Intel compiler w/LAPACK, SGI MPT (dmpar)
```

For the PGI compiler, pick options 1 or 2. For Intel builds, pick option 3, 4, or 5 if your compiler includes the MKL libraries, and option 6 or 7 if it does not.

If successful, the configure script creates a file called `configure.hwrf` in the `hwrf-utilities/` directory. This file contains compilation options, rules, and paths specific to the current machine architecture, and can be edited to change compilation options, if desired.

In csh/tcsh, compile the HWRF-utilities and save the build output to a log file.

`./compile |& tee build.log`

For the ksh/bash shell, use the following command.

`./compile 2>&1 | tee build.log`

If the compilation is successful, it will create the following executables in the directory `exec/`.

```
bufr_remorest.exe
diffwrf_3dvar.exe
grbindex.exe
grp_atcf_to_stats.exe
grp_getcenter.exe
grp_gridparse.exe
grp_hwrf_atcf_intensity.exe
grp_hwrf_atcf_tracks.exe
grp_inddiag.exe
grp_inddiagnull.exe
grp_nameparse.exe
grp_statsin_domain.exe
grp_statsin_domain_TI.exe
grp_totaldiag.exe
hwrf_afos.exe
hwrf_anl_4x_step2.exe
hwrf_anl_bogus_10m.exe
hwrf_anl_cs_10m.exe
```

```
hwrf_atcf_to_stats.exe
hwrf_aux_rw.exe
hwrf_bdy_update.exe
hwrf_binary_grads.exe
hwrf_bin_io.exe
hwrf_blend_gsi.exe
hwrf_combinetrack.exe
hwrf_create_nest_1x_10m.exe
hwrf_create_trak_fnl.exe
hwrf_create_trak_guess.exe
hwrf_data_flag.exe
hwrf_data_remv.exe
hwrf_gridgenfine.exe
hwrf_guess.exe
hwrf_htcfstats.exe
hwrf_inter_2to1.exe
hwrf_inter_2to2.exe
hwrf_inter_2to6.exe
hwrf_inter_4to2.exe
hwrf_inter_4to6.exe
hwrf_merge_nest_4x_step12_3n.exe
hwrf_netcdf_grads.exe
hwrf_nhc_products.exe
hwrf_old_gettrk.exe
hwrf_pert_ct1.exe
hwrf_prep.exe
hwrf_read_indi_write_all.exe
hwrf_readtdrstmid.exe
hwrf_readtdrtime.exe
hwrf_split1.exe
hwrf_supvit.exe
hwrf_swath.exe
hwrf_swcorner_dynamic.exe
hwrf_wrfout_newtime.exe
mdate.exe
mpiserial.exe
ndate.exe
nhour.exe
serpoe.exe
wave_sample.exe
wgrib.exe
```

In addition, it will create twelve libraries in the directory `libs/`.

`libbacio.a` — BACIO library
`libblas.a` — BLAS library
`libbufr_i4r4.a` — BUFR library built with -i4 Đr4 flags
`libbufr_i4r8.a` — BUFR library built with -i4 -r8 flags
`libg2.a` — GRIB2 library
`libhwrfutil_i4r4.a` — Miscellaneous data manipulation utilities
`libsfcio_i4r4.a` — SFCIO library built with -i4 -r4 flags
`libsigio_i4r4.a` — SIGIO library built with Đi4 Đr4 flags
`libsp_i4r8.a` — SP library built with -i4 -r8 flags
`libsp_i4r4.a` — SP library built with -i4 -r4 flags
`libw3_i4r8.a` — W3 library built with -i4 -r8 flags
`libw3_i4r4.a` — W3 library built with -i4 -r4 flags

These libraries will be used by the GFDL Vortex Tracker and the MPIPOM-TC ocean model. The configuration step for these components will require setting a path variable to point to the `hwrf-utilities/libs/` directory in the HWRF-utilities directory.

If a recompilation is necessary, a clean to remove all object files (except those in `external/`) should be completed first.

`./clean`

A complete clean is strongly recommended if the compilation failed, the Registry has been

changed, or the configuration file is changed. To conduct a complete clean that removes all built files in all directories, as well as the `configure.hwrf`, use the "-a" option.

```
./clean -a
```

The HWRF-utilities can be compiled to produce only the libraries by typing the command below.

```
./compile library
```

This is useful for users that do not intend to use the entire HWRF system, but just need the libraries to build the tracker.

2.8 Building MPIPOM-TC

2.8.1 Set Environment Variables

The Tropical Cyclone version of the MPIPOM-TC requires four external libraries: SFCIO, SP, W3, and PnetCDF. On platforms that lack the ESSL mathematical libraries, typically anything other than IBM AIX machines, a fifth library (BLAS) is required. The first three of these libraries are located in the `hwrf-utilities/libs/` directory and should be available if the HWRF-utilities component has been built successfully. You must first build them before building MPIPOM-TC.

Set the library paths (assuming the directory structure proposed in Section 2.3) using C-Shell.

```
setenv LIB_W3_PATH ${SCRATCH}/hwrfrun/sorc/hwrf-utilities/libs/
setenv LIB_SP_PATH ${SCRATCH}/hwrfrun/sorc/hwrf-utilities/libs/
setenv LIB_SFCIO_PATH ${SCRATCH}/hwrfrun/sorc/hwrf-utilities/libs/
setenv PNETCDF PATH_TO_PNETCDF
```

Similarly, the libraries can be set using the ksh/bash shell.

```
export LIB_W3_PATH=${SCRATCH}/hwrfrun/sorc/hwrf-utilities/libs/
export LIB_SP_PATH=${SCRATCH}/hwrfrun/sorc/hwrf-utilities/libs/
export LIB_SFCIO_PATH=${SCRATCH}/hwrfrun/sorc/hwrf-utilities/libs/
export PNETCDF=PATH_TO_PNETCDF
```

In addition to these libraries, MPIPOM-TC requires linear algebra routines from the BLAS library. When building MPIPOM-TC on an IBM platform, the build will automatically use the ESSL library, which includes highly optimized versions of some of the BLAS routines. When building MPIPOM-TC in a platform without ESSL (such as Linux), the build system uses the BLAS mathematical library provided with the hwrf-utilities component. In such a case, the fifth and final path must be set.

```
setenv LIB_BLAS_PATH ${SCRATCH}/hwrfrun/sorc/hwrf-utilities/libs/
```

For the ksh/bash shells the path can be set similarly.

```
export LIB_BLAS_PATH=${SCRATCH}/hwrfrun/sorc/hwrf-utilities/libs/
```

2.8.2 Configure and Compile

Configure MPIPOM-TC for compilation from within the `pomtc/` directory.

```
./configure
```

The configure script checks the system hardware, and if the path variables are not set, asks for software paths to the W3, SP, SFCIO, and PnetCDF, and for Linux, the BLAS libraries. It concludes by asking the user to choose a configuration supported by current machine architecture.

For the IBM, only one choice is available.

```
1.  AIX (dmpar)
```

The following options exist for Linux.

```
1.  Linux x86_64, PGI compiler (dmpar)
2.  Linux x86_64, PGI compiler, SGI MPT (dmpar)
3.  Linux x86_64, Intel compiler (dmpar)
4.  Linux x86_64, Intel compiler, SGI MPT (dmpar)
5.  Linux x86_64, Intel compiler, IBM POE (dmpar)
```

After selecting the desired compiler option, the configure script creates a file called configure.pom. This file contains compilation options, rules, and paths specific to the current machine architecture, and can be edited to change compilation options, if desired.

Compile the MPIPOM-TC and save the build output to a log file with csh/tcsh.

```
./compile |& tee ocean.log
```

Simarly, for ksh, use the following syntax.

```
./compile 2>&1 | tee ocean.log
```

If the compilation is successful, eleven executables are created in `ocean_exec/`.

```
gfdl_date2day.exe
gfdl_day2date.exe
gfdl_getsst.exe
gfdl_sharp_mcs_rf_l2m_rmy5.exe
hwrf_ocean_fcst.exe
hwrf_ocean_init.exe
pomprep_fbtr.xc
```

```
pomprep_gdm3.xc
pomprep_idtr.xc
pomprep_ncda.xc
transatl06prep.xc
```

If a recompilation is necessary, a clean to remove all object files should be completed.

```
./clean
```

A complete clean is strongly recommended if the compilation failed, the configuration file has been changed, or the configuration file is changed. To conduct a complete clean that removes all built files in all directories, as well as the `configure.pom`, use the "-a" option.

```
./clean -a
```

2.9 Building GFDL Vortex Tracker

2.9.1 Set Environment Variables

The GFDL Vortex Tracker requires two external libraries, W3 and BACIO. These libraries are located in the `hwrf-utility/libs/` directory and should be available if the HWRF-utilities are successfully built. You must build the HWRF-utilities before building the GFDL Vortex Tracker.

Again, assuming that the directory structure is the same as that proposed in Section 2.3, set the library paths.

```
setenv LIB_W3_PATH ${SCRATCH}/hwrfrun/sorc/hwrf-utilities/libs/
setenv LIB_BACIO_PATH ${SCRATCH}/hwrfrun/sorc/hwrf-utilities/libs/
setenv LIB_G2_PATH ${SCRATCH}/hwrfrun/sorc/hwrf-utilities/libs/
setenv LIB_Z_PATH SYSTEM_LOCATION
setenv LIB_PNG_PATH SYSTEM_LOCATION
setenv LIB_JASPER_PATH SYSTEM_LOCATION
```

Similarly, the syntax for the ksh/bash shell can be used.

```
export LIB_W3_PATH=${SCRATCH}/hwrfrun/sorc/hwrf-utilities/libs/
export LIB_BACIO_PATH=${SCRATCH}/hwrfrun/sorc/hwrf-utilities/libs/
export LIB_G2_PATH=${SCRATCH}/hwrfrun/sorc/hwrf-utilities/libs/
export LIB_Z_PATH=SYSTEM_LOCATION
export LIB_PNG_PATH=SYSTEM_LOCATION
export LIB_JASPER_PATH=SYSTEM_LOCATION
```

Where *SYSTEM_LOCATION* should be replaced with the full path to the installed library. On

many systems, these libraries reside in `/usr/lib` or `/usr/lib64`.

2.9.2 Configure and Compile

Configure the GFDL Vortex Tracker for compilation from within the `gfdl-vortextracker` directory.

`./configure`

The configure script checks the system hardware, and if the path variables are not set, asks for software paths to the W3 and BACIO libraries. It concludes by asking the user to choose a configuration supported by current machine architecture.

For Linux, there are six options.

```
1.  Linux x86_64, PGI compiler (serial)
2.  Linux x86_64, Intel compiler (serial)
3.  Linux x86_64, Intel compiler super debug (serial)
4.  Linux x86_64, PGI compiler, SGI MPT (serial)
5.  Linux x86_64, Intel compiler, SGI MPT (serial)
6.  Linux x86_64, Intel compiler, IBM POE (serial)
```

The configure script creates a file called `configure.trk`. This file contains compilation options, rules, and paths specific to the current machine architecture.

The configure file can be edited to change compilation options, if desired.

Compile the vortex tracker and save the build output to a log file.

`./compile |& tee tracker.log`

The command for the ksh/bash shell follows.

`./compile 2>&1 | tee tracker.log`

If the compilation was successful, three executables are created in the directory `trk_exec/`.

```
hwrf_gettrk.exe
hwrf_tave.exe
hwrf_vint.exe
```

If a recompilation is necessary, a clean to remove all object files should be completed.

`./clean`

A complete clean is strongly recommended if the compilation failed, the configuration file has been changed, or the configuration file is changed. To conduct a complete clean that

removes all built files in all directories, as well as the `configure.trk`, use the "-a" option.

```
./clean -a
```

2.10 Building the NCEP Coupler

2.10.1 Configure and Compile

Configure the NCEP Coupler for compilation from within the `ncep-coupler/` directory.

```
./configure
```

The configure script checks the system hardware, asks the user to choose a configuration supported by current machine architecture, and creates a configure file called `configure.cpl`.

There are five dmpar options for Linux.

```
1.  Linux x86_64, PGI compiler (dmpar)
2.  Linux x86_64, PGI compiler, SGI MPT (dmpar)
3.  Linux x86_64, Intel compiler (dmpar)
4.  Linux x86_64, Intel compiler, SGI MPT (dmpar)
5.  Linux x86_64, Intel compiler, IBM POE (dmpar)
```

The configure file `configure.cpl` contains compilation options, rules, and paths specific to the current machine architecture, and can be edited to change compilation options if desired.

Compile the coupler and save the build output to a log file.

```
./compile |& tee coupler.log
```

For the ksh/bash shell, use the following command.

```
./compile 2>&1 | tee coupler.log
```

If the compilation is successful, it will create the single executable `hwrf_wm3c.exe` in the `cpl_exec/` directory.

If a recompilation is necessary, a clean to remove all object files should be completed.

```
./clean
```

A complete clean is strongly recommended if the compilation failed, the configuration file has been changed, or the configuration file is changed. To conduct a complete clean that removes all built files in all directories, as well as the `configure.cpl`, use the "-a" option.

```
./clean -a
```

2.11 Building WPS

2.11.1 Set Environment Variables

The WRF WPS requires the same build environment as the WRF-NMM model, including the netCDF libraries and MPI libraries. Since the WPS makes direct calls to the WRF I/O API libraries included with the WRF model, the WRF-NMM model must be built prior to building the WPS.

Set up the build environment for WPS by setting the `WRF_DIR` environment variable.

```
setenv WRF_DIR ${SCRATCH}/hwrfrun/sorc/WRFV3/
```

For bash/ksh, use the command that follows.

```
export WRF_DIR=${SCRATCH}/hwrfrun/sorc/WRFV3/
```

Further details on using the WPS to create HWRF input data can be found in Chapter 4 of the HWRF Users' Guide.

Complete details on building and running the WPS are available from the WRF-NMM Users' Guide, and can be downloaded from:

`http://www.dtcenter.org/wrf-nmm/users/docs/overview.php`.

2.11.2 Configure and Compile

Following the compilation of the WRF-NMM executables, change to the WPS directory and issue the configure command.

```
./configure
```

Select the appropriate *dmpar* option for your architecture and compiler choice. If you plan to use GRIB2 data, you will also need to select a build option that supports GRIB2 I/O. This will generate the configure resource file.

On Linux computers, there are 38 listed options. The first 28 are the most relevant to HWRF. Select if you want GRIB2 support, or if you don't.

```
1.  Linux x86_64, gfortran (serial)
2.  Linux x86_64, gfortran (serial_NO_GRIB2)
3.  Linux x86_64, gfortran (dmpar)
```

```
4.  Linux x86_64, gfortran (dmpar_NO_GRIB2)
5.  Linux x86_64, PGI compiler (serial)
6.  Linux x86_64, PGI compiler (serial_NO_GRIB2)
7.  Linux x86_64, PGI compiler (dmpar)
8.  Linux x86_64, PGI compiler (dmpar_NO_GRIB2)
9.  Linux x86_64, PGI compiler, SGI MPT (serial)
10.  Linux x86_64, PGI compiler, SGI MPT (serial_NO_GRIB2)
11.  Linux x86_64, PGI compiler, SGI MPT (dmpar)
12.  Linux x86_64, PGI compiler, SGI MPT (dmpar_NO_GRIB2)
13.  Linux x86_64, IA64 and Opteron (serial)
14.  Linux x86_64, IA64 and Opteron (serial_NO_GRIB2)
15.  Linux x86_64, IA64 and Opteron (dmpar)
16.  Linux x86_64, IA64 and Opteron (dmpar_NO_GRIB2)
17.  Linux x86_64, Intel compiler (serial)
18.  Linux x86_64, Intel compiler (serial_NO_GRIB2)
19.  Linux x86_64, Intel compiler (dmpar)
20.  Linux x86_64, Intel compiler (dmpar_NO_GRIB2)
21.  Linux x86_64, Intel compiler, SGI MPT (serial)
22.  Linux x86_64, Intel compiler, SGI MPT (serial_NO_GRIB2)
23.  Linux x86_64, Intel compiler, SGI MPT (dmpar)
24.  Linux x86_64, Intel compiler, SGI MPT (dmpar_NO_GRIB2)
25.  Linux x86_64, Intel compiler, IBM POE (serial)
26.  Linux x86_64, Intel compiler, IBM POE (serial_NO_GRIB2)
27.  Linux x86_64, Intel compiler, IBM POE (dmpar)
28.  Linux x86_64, Intel compiler, IBM POE (dmpar_NO_GRIB2)
```

Select the appropriate "dmpar" option for your choice of compiler.

Compile the coupler and save the build output to a log file.

```
./compile |& tee wps.log
```

For the ksh/bash shell, use the equivalent command.

```
./compile 2>&1 | tee wps.log
```

After issuing the compile command, a successful compilation of WPS produces the three symbolic links: `geogrid.exe`, `ungrib.exe`, and `metgrid.exe` in the `WPSV3/` directory, and several symbolic links in the `util/` directory.

```
avg_tsfc.exe
calc_ecmwf_p.exe
g1print.exe
g2print.exe
height_ukmo.exe
int2nc.exe
mod_levs.exe
rd_intermediate.exe
```

If any of these links do not exist, check the compilation log file to determine what went wrong.

A complete clean is strongly recommended if the compilation failed or if the configuration file is changed. To conduct a complete clean that removes ALL build files, including the executables, libraries, and the `configure.wps`, use the "-a" option to clean.

```
./clean -a
```

For full details on the operation of WPS, see the WPS chapter of the WRF-NMM Users' Guide.

2.12 Building UPP

The NCEP Unified Post-Processor was designed to interpolate WRF output from native coordinates and variables to coordinates and variables more useful for analysis. Specifically, UPP de-staggers the HWRF output, interpolates the data from its native vertical grid to standard levels, and creates additional diagnostic variables.

The UPP requires the same Fortran and C compilers used to build the WRF model. In addition, UPP requires the netCDF library and the WRF I/O API libraries (the latter is included with the WRF build). The UPP build requires a number of support libraries (IP, SP, W3), which are provided with the source code and are located in the `UPP/lib/` directory. These libraries are for the UPP build only. They should not be confused with the libraries of the same name located in the `hwrf-utilities/libs/` directory.

2.12.1 Set Environment Variables

The UPP requires the WRF I/O API libraries to successfully build. These are created when the WRF model is built. If the WRF model has not yet been compiled, it must first be built before compiling UPP.

Since the UPP build requires linking to the WRF-NMM I/O API libraries, it must be able to find the WRF directory. The UPP build uses the `WRF_DIR` environment variable to define the path to WRF. The path variable `WRF_DIR` must therefore be set to the location of the WRF directory.

In addition to setting the path variable, building UPP for use with HWRF requires setting the environment variable HWRF. This is the same variable set when building WRF-NMM for HWRF.

Set up the environment for UPP.

```
setenv HWRF 1
setenv WRF_DIR ${SCRATCH}/hwrfrun/sorc/WRFV3/
setenv JASPERLIB SYSTEM_LOCATION_SO_FILE
setenv JASPERINC SYSTEM_LOCATION_HEADER_FILE
```

The syntax for bash/ksh is as follows.

```
export HWRF=1
export WRF_DIR=${SCRATCH}/hwrfrun/sorc/WRFV3/
export JASPERLIB=SYSTEM_LOCATION_SO_FILE
export JASPERINC=SYSTEM_LOCATION_HEADER_FILE
```

2.12.2 Configure and Compile

UPP uses a build mechanism similar to that used by the WRF model. Type

```
./configure
```

to generate the UPP configure file. The configure script will complain if the `WRF_DIR` path has not been set. You will then be given a list of configuration choices tailored to your computer.

For the LINUX operating systems, there are 10 options. Select the appropriate `dmpar` option compatible with your system.

```
1.  Linux x86_64, PGI compiler (serial)
2.  Linux x86_64, PGI compiler (dmpar)
3.  Linux x86_64, Intel compiler (serial)
4.  Linux x86_64, Intel compiler (dmpar)
5.  Linux x86_64, Intel compiler, SGI MPT (serial)
6.  Linux x86_64, Intel compiler, SGI MPT (dmpar)
7.  Linux x86_64, gfortran compiler (serial)
8.  Linux x86_64, gfortran compiler (dmpar)
9.  Linux x86_64, Intel compiler, IBM POE (serial)
10.  Linux x86_64, Intel compiler, IBM POE (dmpar)
```

The configuration script will generate the configure file `configure.upp`. If necessary, the `configure.upp` file can be modified to change the default compile options and paths.

To compile UPP, enter the following command (csh/tsch).

```
./compile |& tee build.log
```

Alternatively, the ksh/bash command can be used.

```
./compile 2>&1 | tee build.log
```

This command should create 13 UPP libraries in `lib/`.

```
libbacio.a
libCRTM.a
libg2.a
libg2tmpl.a
libgfsio.a
libip.a
libnemsio.a
libsfcio.a
libsigio.a
libsp.a
libw3emc.a
libw3nco.a
libxmlparse.a
```

Four UPP executables are produced in `bin/`.

```
cnvgrib.exe
copygb.exe
ndate.exe
unipost.exe
```

Once again, these libraries are for the UPP only, and should not be used by the other components.

A complete clean is strongly recommended if the compilation failed, or if the configuration file or source code is changed. Conduct a complete clean that removes ALL build files, including the executables, libraries, and the configure.upp.

`./clean`

For full details on the operation of UPP, see the UPP chapter of the HWRF Users' Guide, and for complete details on building and running the UPP, see the WRF-NMM Users' Guide, which can be downloaded at:

`http://www.dtcenter.org/wrf-nmm/users/docs/overview.php`.

2.13 Building GSI

The community GSI requires the same build environment as the WRF-NMM model, including the netCDF, MPI, and LAPACK libraries. In addition, GSI makes direct calls to the WRF I/O API libraries included with the WRF model. Therefore the WRF model must be built prior to building the GSI.

Further details on using the GSI with HWRF can be found in later chapters of this HWRF Users' Guide.

2.13.1 Set Environment Variables

Building GSI for use with HWRF requires setting three environmental variables. The first, HWRF indicates to turn on the HWRF options in the GSI build. This is the same flag set when building WRF-NMM for HWRF. The second is a path variable pointing to the root of the WRF build directory. The third is the variable `LAPACK_PATH`, which indicates the location of the LAPACK library on your system.

Set up the environment for GSI.

```
setenv HWRF 1
setenv WRF_DIR ${SCRATCH}/hwrfrun/sorc/WRFV3/
```

You may use bash/ksh instead.

```
export HWRF=1
export WRF_DIR=${SCRATCH}/hwrfrun/sorc/WRFV3/
```

The additional environment variable `LAPACK_PATH` may be needed on some systems. Typically, the environment variable `LAPACK_PATH` needs only to be set on Linux systems without a vender provided version of LAPACK. IBM systems usually have the ESSL library installed and therefore do not need the LAPACK. Likewise, the PGI compiler often comes with a vender-provided version of LAPACK that links automatically with the compiler. Problems with the vender-supplied LAPACK library are more likely to occur with the Intel compiler. While the Intel compilers typically have the MKL libraries installed, the ifort compiler does not automatically load the library. It is therefore necessary to set the `LAPACK_PATH` variable to the location of the MKL libraries when using the Intel compiler.

Supposing that the MKL library path is set to the environment variable `MKL`, then the LAPACK environment may be set in terms of `MKL`.

```
setenv LAPACK_PATH $MKL
```

Alternatively, the bash/ksh option is as follows.

```
export LAPACK_PATH=$MKL
```

2.13.2 Configure and Compile

To build GSI for HWRF, change into the GSI directory and issue the configure command.

```
./configure
```

Choose one of the configure options listed. On Linux computers, the listed options are as follows.

```
1.  Linux x86_64, PGI compilers (pgf90 & pgcc) (dmpar,optimize)
```

```
2.  Linux x86_64, PGI compilers (pgf90 & pgcc) w/ Vender supplied MPI (dmpar,optimize)
3.  Linux x86_64, PGI compilers (pgf90 & gcc) (dmpar,optimize)
4.  Linux x86_64, PGI compilers (pgf90 & gcc) w/ Vender supplied MPI (dmpar,optimize)
5.  Linux x86_64, GNU compilers (gfortran & gcc) (dmpar,optimize)
6.  Linux x86_64, Intel compiler (ifort & icc) (dmpar,optimize)
7.  Linux x86_64, Intel compiler (ifort & icc) w/ Vender supplied MPI (dmpar,optimize)
8.  Linux x86_64, Intel/gnu compiler (ifort & gcc) (dmpar,optimize)
9.  Linux x86_64, Intel/gnu compiler (ifort & gcc) w/ Vender supplied MPI (dmpar,optimize)
```

Select the appropriate `dmpar` option for your platform and compiler. For a generic Linux machine, choose option (1) or (3) for a PGI build, or option (6) or (8) for an Intel build. On Jet, select option (6) for an Intel build. For a SGI Linux installation, select (2) for PGI or (6) for Intel. For an IBM Linux installation with an Intel compiler, select option (6).

After selecting the proper option, run the compile script.

```
./compile |& tee build.log
```

For the ksh/bash shell, use the following command.

```
./compile 2>&1 | tee build.log
```

The successful completion of the compile will place the GSI executable `gsi.exe` in the `run/` directory. If the executable is not found, check the compilation log file to determine what went wrong.

A complete clean is strongly recommended if the compilation failed or if the configuration file is changed. To conduct a complete clean that removes ALL build files, including the executables, libraries, and the configure.gsi, use the "-a" option with clean.

```
./clean -a
```

For details on using GSI with HWRF, see the GSI chapter in the HWRF Users' Guide. For full details on the operation of GSI, see the DTC Community GSI Users' Guide.

`http://www.dtcenter.org/com-GSI/users/docs/index.php`

3

Running HWRF

3.1 HWRF Scripts Overview

HWRF v3.6a is run by a series of high-level ksh scripts in the `wrappers/` directory, whose primary duty is to set environment variables and call the mid-level Python scripts in the `scripts/` directory. The mid-level Python scripts call HWRF-specific Python modules in the `ush/` directory and subdirectories. Many of the experiment configuration parameters and variables are set by files in the `parm/` directory.

```
hwrfrun/
├── exec/
├── nwport/
├── parm/
├── scripts/
├── sorc/
├── ush/
│   ├── hwrf/
│   ├── pom/
│   ├── produtil/
│   └── hwrf_*.py
└── wrappers/
```

3.2 Defining an Experiment

Typically, there are five files responsible for configuring an experiment. Within the `parm/` directory, there are four configure files.

```
hwrf.conf
hwrf_basic.conf
hwrf_input.conf
system.conf
```

In `wrappers/`, there is `global_vars.ksh`.

This section gives an overview of each of these files and how they should be edited to configure an experiment.

3.2.1 Overview of `hwrf.conf`

The purpose of `hwrf.conf` is to define the directory structure for the source code, and many of the more common namelist options for the components of HWRF. If the source code directory structure has been set up as defined in Section 2.3, and HWRF will be run with the 2014 operational configuration, `hwrf.conf` will point to the correct executable files. Make sure that the executable variables that point to paths including `$HOMEhwrf` are uncommented.

Note that not all of the namelist parameters have been included in this file. Those found in this file will override those set in the template namelists found in the `hwrf-utilities/parm/` directory.

3.2.2 Overview of `hwrf_basic.conf`

The purpose of `hwrf_basic.conf` is to define some basic aspects of the HWRF configuration. In this file, users can choose whether or not the data assimilation, vortex relocation, and coupling with the ocean will be turned on. Additionally, variable `use_spectral` controls whether the HWRF will use spectral GFS and GDAS files to create initial conditions. When variable `use_spectral` is set to no, initialization is done using only GRIB files. Note that the GSI still needs spectral input from GDAS and the GFS Ensemble to run in hybrid mode. The `use_spectral` flag only controls the input to ungrib or prep_hybrid, it does not control the input to GSI.

Another important variable in this configuration file is `allow_fallbacks`. Most community users will have it set to `no`, which is the default. In operations, this variable is set to `yes`, which enables alternate paths in the HWRF run in case a component fails. For instance, when `allow_fallbacks` is set to `yes` and GSI fails, the run does not stop but instead the initial conditions for the main forecast are obtained directly from the output of the vortex

relocation.

Finally, `hwrf_basic.conf` is also used to set variables defining the directories containing the HWRF source code (`HOMEhwrf`) and various output directories. Some of the output directories are scrubbed by default after each run to avoid filling the disk (directories under `CDSCRUB`), while others are not scrubbed (those under `CDNOSCRUB`). To turn off scrubbing for a component of HWRF, go to that section of `hwrf_basic.conf` and add `SCRUB=NO`. One of those directories, where most of the output is written to, is named `WORKhwrf`.

3.2.3 Overview of `hwrf_input.conf`

This file defines the input data directory structure. It currently contains input data locations for users on three platforms: those on the NOAA research and development machines Zeus or Jet with access to the EMC input data directory; and community users that stage the input data using the directory structure described in Section 3.3. These three sets of input data are described in the sections labeled `[zeus_hist]`, `[jet_hist]`, and `[comm_hist]`, respectively.

Users who wish to adopt a different input data directory structure may define it within this file by adding an additional section, or by editing the existing `[comm_hist]` section. While the input data can be placed anywhere that is locally available to the compute nodes being used, users are not advised to change the input file naming convention.

The choice of which set of input data will actually be used in an experiment is determined by variable `input_catalog` in file `system.conf`. In order to use the test datasets provided by DTC, users should set this variable to `comm_hist`. The user must also set the path to the location of the input data within the `[comm_hist]` section of `hwrf_input.py` by editing the variable `inputroot`.

3.2.4 Overview `system.conf`

This file defines the top-level output directory structure and a handful of other variables used for automation. Community users will need to copy the example file `system.conf.jet` to `system.conf` and customize the following variables.

`disk_project`	The name of the project on the computational platform (used to compose the output directory name).
`input_catalog`	The choice of the input file location. This location must exist in the file `hwrf_input.conf`. Community users should set it to `comm_hist`.
`[dir]` section	`CDNOSCRUB`, `CDSCRUB`, and `CDSAVE` set the location of the HWRF install and output files.
`syndat`	The location of input TC Vitals. Community users should set it to the location of staged SYNDAT-PLUS dataset, as explained later in this chapter.

The sections [wrfexe] and [runwrf] describe the mapping of the processor distribution for the WRF runs and should not be altered.

3.2.5 Overview of global_vars.ksh

The file global_vars.ksh is used for setting a few environment variables to define the case to be run by the wrapper scripts. The user should set the first five variables.

export START_TIME=*YYYYMMDDHH*	Initial time of the forecast
export SID=18L	Storm ID, e.g. 18L is Sandy – the 18th storm of the Atlantic season; Invest 99 in the East Pacific would be 99E
export CASE=HISTORY	For most users, this will be HISTORY. FORECAST is reserved for real-time mode.
export HOMEhwrf=	Full path to the directory where HWRF has been installed
export expt=test1	A name chosen by the user for the specific test being conducted, e.g. test1, Phystest, etc.

3.3 Input Data and Fix Directory Structure

Users will need the datasets below as input to HWRF components.

The following lists outline the files needed to initialize forecasts using the operational configuration. Analysis times are indicated by capital letters, such as *YYYYMMDDHH* or *HH*, and forecast hours are indicated by lower-case letters, i.e. *hhh*. For example, gfs.2012102806.pgrbf024 is a GFS 24-h forecast in GRIB format whose initial time is October 28 06Z 2012 and would be indicated as gfs.*YYYYMMDDHH*.pgrbf*hhh*. The storm identifier, *sid* is the storm number and a lower case letter corresponding to the basin (e.g., 18l).

```
./
├── DATA/
│   ├── OCEAN/
│   │   └── gfs.YYYYMMDDHH/YYYYMMDDHH/
│   └── WINDS/
│       └── gfs.YYYYMMDDHH/
├── enkf_files/
│   └── YYYYMMDDHH/
├── RECON/
│   └── gdas.YYYYMMDD/
└── SYNDAT-PLUS/
    ├── tcvitals.txt.prev.YYYY
    ├── syndat_tcvitals.YYYY
    └── investnamed/
```

```
    ├── old/
    ├── renumbered/
    └── work/
├── TDR/
    └── YYYY/YYYYMMDDHH/sid/
└── fix/
    ├── bogus/
    ├── hwrf_cpat.ice
    ├── hwrf_cpat.moddef
    ├── hwrf_cpat.wind
    ├── hwrf-crtm/ ..... Coefficients for Radiative Transfer
    ├── hwrf-crtm-2.1.3/ ..... Updated Coefficients for Radiative Transfer
    ├── hwrf_disclaimer.txt
    ├── hwrf_eta_micro_lookup.dat
    ├── hwrf_fix_datestamp
    ├── hwrf-gsi/ ..... GSI fixed files
    ├── hwrf-pom/ ..... Ocean init climate data
    ├── hwrf_storm_20
    ├── hwrf_storm_25
    ├── hwrf_storm_cyn_axisy_47
    ├── hwrf_storm_cyn_axisy_50_ep
    ├── hwrf_storm_radius
    ├── hwrf_track
    ├── hwrf_wps_geo/ ..... Land use fixed files
    ├── hwrf-wrf/ ..... Fixed files for WRF
    └── loop_curr/ ..... Loop current initialization data
```

The `OCEAN/gfs.YYYYMMDDHH/YYYYMMDDHH` directory contains loop current and warm core ring initialization data, as well as spectral atmospheric analyses in the following four files.

`gfdl_loop_current_rmy5.dat`	Loop current data
`gfdl_loop_current_wc_ring_rmy5.dat`	Warm core ring data
`gfs.tHH.sanl`	GFS analysis for ocean initialization
`gfs.tHH.sfcanl`	GFS sfc analysis for ocean initialization

The `WINDS/gfs.YYYYMMDDHH/` directory contains data for the initialization of the atmosphere. These fall into four categories: observations, text data, gridded data, and spectral data. The prefix for each file denotes the NWP system from which the data originates.

The following files are considered observations, and are in either BUFR or prepBURF formats. *HH* is the analysis time and *HH−6* is the analysis time six hours prior.

gfs.t*HH*.syndata.tcvitals.tm00	Tropical cylone obs used by GFS
gfs.t*HH*.prepbufr.nr	Conventional obs
gfs.t*HH*.adpupa.tm00.bufr_d	Upper-air obs
gfs.t*HH*.proflr.tm00.bufr_d	Profiler obs
gfs.t*HH*.*SATELLITE*.tm00.bufr_d	Satellite obs
gdas1.t{*HH-6*}.abias	Time-dep. sat. bias correction
gdas1.t{*HH-6*}.satang	Angle-dep. sat. bias correction

In the list above, *SATELLITE* can be any of the following, but this is not an all-inclusive list.

1bamua	airsev	gome
1bamub	goesfv	sevcsr
1bhrs3	ascatt	satwind
1bhrs4	ascatw	rassda
1bmhs	atms	
esamua	avcsam	
esamub	avcspm	

The following file is in text (ascii) format.

avn.t*HH*z.cyclone.trackatcfunix	forecast track from GFS

The following files are in gridded binary (GRIB) format.

gfs.t*HH*.pgrbf00	GFS analysis
gdas1.t*HH-6*.pgrbh03	3-h forecast from previous 6-h GDAS analysis
gdas1.t*HH-6*.pgrbh06	6-h forecast from previous 6-h GDAS analysis
gdas1.t*HH-6*.pgrbh09	9-h forecast from previous 6-h GDAS analysis

The remainder of the files are spectral files in binary format.

gfs.t*HH*.sfcanl	GFS surface analysis
gfs.t*HH*.sanl	GFS atmospheric analysis
gfs.t*HH*.sf*hhh*	(one for each fcst hour from 0 to 126 in 3 h increments)
gdas1.t*HH-6*.sf03	3-h forecast from previous 6-h GDAS analysis
gdas1.t*HH-6*.sf06	6-h forecast from previous 6-h GDAS analysis
gdas1.t*HH-6*.sf09	9-h forecast from previous 6-h GDAS analysis

The enkf_files/*YYYYMMDDHH*/ directory contains the GFS Ensemble spectral data to provide ensemble information for data assimilation. There are eighty files named,

sfg_*YYYYMMDDHH*_fhr*hh*s_mem{*mmm*},

where *mmm* is the 3-digit ensemble member ID and ranges from 1 to 80.

The SYNDAT-PLUS directory contains the TC Vitals files for several years.

The RECON/gdas.*YYYYMMDD* directory contains aircraft reconnaissance data in BUFR format and some additional files with metadata. Files are named by the following conventions.

gdas1.t*HH.SID*.hdob.tm00.bufr_d
gdas1.t*HH*z.status_hdob.tm00.bufr_d

The TDR/*YYYY/YYYYMMDDHH/SID* directory contains tail Doppler radar data in BUFR format in a file named as follows.

gdas1.t*HH*z.tldplr.tm00.bufr_d

The fix files are time-independent and are included in the following directories.

- bogus/

  hwrf_ofs_atl.A12grid.dat
  hwrf_ofs_atl.intp_pars.dat
  hwrf_ofs_atl.ismus_msk1152x576.dat
  hwrf_ofs_atl.ismus_msk384x190.dat
  hwrf_ofs_atl.ismus_msk512x256.dat
  hwrf_ofs_atl.ismus_msk768x384.dat
  hwrf_ofs_atl.ncep1_12.regional.depth.a
  hwrf_ofs_atl.ncep1_12.regional.depth.b
  hwrf_ofs_atl.ncep1_12.regional.grid.a
  hwrf_ofs_atl.ncep1_12.regional.grid.b
  hwrf_ofs_atl.ncep1_12.regional.mask.a
  hwrf_ofs_atl.ncep1_12.regional.mask.b

- hwrf-crtm-2.1.3/

  AerosolCoeff/
  CloudCoeff/
  EmisCoeff/
  SpcCoeff/
  TauCoeff/

- hwrf-gsi/

  anavinfo_hwrf
  anavinfo_hwrf_d2
  atms_beamwidth.txt
  bufrtab.012
  global_ozinfo.txt
  global_scaninfo.txt
  hwrf_convinfo.txt
  hwrf_hybens_d01_locinfo
  hwrf_hybens_d01_locinfo_l76
  hwrf_nam_errtable.r3dv
  hwrf_satinfo.txt
  nam_glb_berror.f77.gcv
  nam_global_pcpinfo.txt
  nam_global_satangbias.txt
  prepobs_errtable.hwrf
  prepobs_prep.bufrtable

- hwrf-pom/ – [#-#] represents multiple files numbered consecutively, e.g. [01-12] means there are twelve files with a two-digit number ranging from 01-12 replacing the string within the brackets.

  gfdl_albedo.fall
  gfdl_albedo.spring
  gfdl_albedo.summer
  gfdl_albedo.winter
  gfdl_datainp1
  gfdl_datainp1.142
  gfdl_datainp2
  gfdl_disclaimer.txt

```
gfdl_fildef.afos
gfdl_fildef.sdm
gfdl_fort.7
gfdl_fort.7.l42
gfdl_gdem.[00-13].ascii
gfdl_Hdeepgsu.eastatl
gfdl_Hdeepgsu.united
gfdl_height
gfdl_huranal.data
gfdl_initdata.eastatl.[01-12]
gfdl_initdata.gdem3.united.[05-12]
gfdl_initdata.gdem.united.[01-12]
gfdl_initdata.levit.united.[05-12]
gfdl_initdata.united.[01-12]
gfdl_limit_2nest_dat_x.1
gfdl_limit_2nest_dat_x.5
gfdl_limit_2nest_dat_x.6
gfdl_limit_2nest_dat_y.1
gfdl_limit_2nest_dat_y.12
gfdl_limit_2nest_dat_y.15
gfdl_limit_2nest_dat_y.16
gfdl_limit_2nest_dat_y.6
gfdl_ocean_dat
gfdl_ocean_readu.dat.[01-12]
gfdl_ocean_spinup.BAYuf
gfdl_ocean_spinup.FSgsuf
gfdl_ocean_spinup_gdem3.dat.
 [01-12]
gfdl_ocean_spinup_gspath.[01-12]
gfdl_ocean_spinup.SGYREuf
gfdl_ocean_topo_and_mask.eastpac_x.
 lores
gfdl_ocean_topo_and_mask.transatl.
 lores
gfdl_ocean_topo_and_mask.united
gfdl_pctwat
gfdl_raw_temp_salin.eastpac.
 [04-12]
gfdl_wetness
gfdl_znot
sgdemv3s[01-12].nc
tgdemv3s[01-12].nc
```

- hwrf_wps_geo

```
albedo_ncep/
greenfrac/
hangl/
hanis/
hasynw/
hasys/
hasysw/
hasyw/
hcnvx/
hlennw/
hlens/
hlensw/
hlenw/
hslop/
hstdv/
hzmax/
islope/
landuse_10m/
landuse_2m/
landuse_30s/
landuse_30s_with_lakes/
landuse_5m/
maxsnowalb/
modis_landuse_20class_30s/
modis_landuse_21class_30s/
orogwd_10m/
orogwd_1deg/
orogwd_20m/
orogwd_2deg/
orogwd_30m/
soiltemp_1deg/
soiltype_bot_10m/
soiltype_bot_2m/
soiltype_bot_30s/
soiltype_bot_5m/
soiltype_top_10m/
soiltype_top_2m/
soiltype_top_30s/
soiltype_top_5m/
ssib_landuse_10m/
ssib_landuse_5m/
topo_10m/
topo_2m/
topo_30s/
topo_5m/
varsso/
```

- hwrf-wrf

```
aerosol.formatted
aerosol_lat.formatted
aerosol_lon.formatted
aerosol_plev.formatted
co2_trans
ETAMPNEW_DATA
ETAMPNEW_DATA_DBL
ETAMPNEW_DATA.expanded_rain
ETAMPNEW_DATA.expanded_rain_DBL
GENPARM.TBL
LANDUSE.TBL
MPTABLE.TBL
```

```
ozone.formatted                    RRTMG_SW_DATA_DBL
ozone_lat.formatted                SOILPARM.TBL
ozone_plev.formatted               tr49t67
README.fix                         tr49t85
RRTM_DATA                          tr67t85
RRTM_DATA_DBL                      URBPARM.TBL
RRTMG_LW_DATA                      URBPARM_UZE.TBL
RRTMG_LW_DATA_DBL                  VEGPARM.TBL
RRTMG_SW_DATA
```

- loop_curr
  hwrf_gfdl_loop_current_rmy5.dat.*YYYYMMDD*
  hwrf_gfdl_loop_current_wc_ring_rmy5.dat.*YYYYMMDD*

Sample fix files and datasets for running two consecutive forecasts of Hurricane Sandy (October 28, 2012 06 and 12 UTC) can be obtained from the DTC website: http://www.dtcenter.org/HurrWRF/users. In order to use the DTC-supported scripts for running HWRF, these datasets should be stored following the directory structure described above, and must be on a disk accessible by the HWRF scripts and executables.

The following files are available for download.
HWRF_v3.6a_datasets_ensemble.tar.gz
HWRF_v3.6a_datasets_gfs_2012102800.tar.gz
HWRF_v3.6a_datasets_gfs_2012102806.tar.gz
HWRF_v3.6a_datasets_gfs_2012102812.tar.gz
HWRF_v3.6a_datasets_gfsgrib_2012102806.tar.gz
HWRF_v3.6a_datasets_gfsgrib_2012102812.tar.gz
HWRF_v3.6a_datasets_TDR-RECON.tar.gz
HWRF_v3.6a_datasets_SYNDAT-PLUS.tar.gz
HWRF_v3.6a_datasets_ocean.tar.gz
HWRF_v3.6a_fix.tar.gz

3.4 Production Directory Structure

The top production directory is ${WORKhwrf}/*YYYYMMDDHH*/*SID* (where ${WORKhwrf} is an environment variable defined by the scripts, *SID* is storm ID (e.g., 09L), and *YYYYMMDDHH* is the forecast initial time. The following sub-directories will be present for the default operational configuration.

```
${WORKhwrf}/YYYYMMDDHH/SID
├── fgat.tYYYYMMDDHHHH/ ..... Production dir for processing GDAS at each of the fgat hours: -3, 0 +3
│   ├── ghost/
│   ├── prep_hybrid/
│   ├── realinit/
│   ├── relocate/
│   ├── regribber/
│   ├── tracker/
│   ├── wps/
│   └── wrfanl/
├── gdas1.YYYYMMDDHH/ ..... Production dir for merging grids after GSI
├── gfsinit/ ..... Production dir for processing GFS input
│   ├── ghost/
│   ├── prep_hybrid/
│   ├── realfcst/
│   ├── realinit/
│   ├── relocate/
│   ├── regribber/
│   ├── tracker/
│   ├── wps/
│   └── wrfanl/
├── gsi_d02/ ..... Production dir for data assimilation on 9 km domain
├── gsi_d03/ ..... Production dir for data assimilation on 3 km domain
├── intercom/ ..... Dir containing files needed for subsequent processes
├── pom/ ..... Production dir for the ocean initialization
├── regribber/ ..... Working dir for regribbing process. All files are delivered to other locations
├── runwrf/† ..... Production dir for coupled or uncoupled forecast
├── tracker/ ..... Production dir for the GFDL Vortex Tracker
├── storm1.vitals .... Text file containing TC Vitals for forecast storm, including only the current storm label, i.e. 18L for Sandy
├── storm1.vitals.allids Text file containing TC Vitals for forecast storm, including its Invest labels, i.e. 99L and 18L for Sandy
├── storm1.vitals.oldid ... Text file containing TC Vitals for forecast storm, including only its Invest labels, i.e. 99L for Sandy
├── storm1.vitals.renumberlog ..... Log file for vitals messages processed
├── tmpvit ..... Text file containing only the analysis TC Vital message
├── oldvit ..... Text file containing only the 6-h previous TC Vital message
├── PDY ..... Text file with date information
├── hwrf_state.sqlite3 ..... Datastore file
└── hwrf_state.sqlite3.lock
```

† When the forecast is uncoupled, the production directory for the forecast is named `wrfatmos/`.

The purpose of `intercom/` is to store the files that are used for subsequent processes, separating them from the working directory. Within the `intercom/` directory, the structure is similar to that in the `${WORKhwrf}` directory, except each subdirectory contains only the files that will be used in subsequent steps. The following outlines the structure of `intercom/` when running all components of HWRF as in operations.

```
intercom/
└── fgat.tYYYYMMDDHHHH/
```

```
├── gdas_merge/
├── gfsinit/
├── gsi_d02/
├── gsi_d03/
├── nonsatpost-f{hh}h00m/
│   ├── nonsatpost-fhhh00m-moad.egrb
│   ├── nonsatpost-fhhh00m-storm1inner.egrb
│   └── nonsatpost-fhhh00m-storm1outer.egrb
├── pom/
├── regribber/
├── {STORMNAME}{SID}.YYYYMMDDHH.hwrfprs_GRID.grbfhh
├── {STORMNAME}{SID}.YYYYMMDDHH.hwrfsat_GRID.grbfhh
├── {STORMNAME}{SID}.YYYYMMDDHH.hwrftrk.grbfhh
├── {STORMNAME}{SID}.YYYYMMDDHH.hwrftrk.grbfhh.grbindex
└── satpost-f{hh}h00m/
    ├── satpost-fhhh00m
    ├── satpost-fhhh00m-moad.egrb
    ├── satpost-fhhh00m-storm1inner.egrb
    └── satpost-fhhh00m-storm1outer.egrb
```

Additionally, some output files are transferred to the `com/` directory, which is reserved for transfer of files between cycled and for delivery of final products (in an operational setting). This is discussed in Chapter 11.

In the list above, *GRID* is a single letter, c, i, m, n, or p, which are grids with various domains and grid spacings used for regribbing the HWRF forecast for delivery and product generation. In this context, `moad` stands for ***Mother Of All Domains***, or the HWRF parent grid. Conversely, `outer` and `inner` refer to the 9-km and 3-km nests. More information about the contents of each grid is described in Section 11.2.1.

3.5 Scripts for Running HWRF

It is recommended that HWRF v3.6a be run using the wrapper and Python scripts provided with the HWRF v3.6a release. In `scripts/`, users can find mid-level Python scripts that call HWRF-specific Python utilities located in `ush/`. The user is encouraged to run the scripts in the `scripts/` directory using the wrapper scripts located in `wrappers/`. The wrapper scripts set the proper environment variables to run each Python script, as well as execute multiple iterations as needed.

3.5.1 Submitting a Job

Some of the executables are parallel code and can only run on the computation nodes. We recommend that users first connect to the computer's remote computation nodes. To do this on Linux machines that run the MOAB/Torque, such as NOAA's Jet, users can use the `qsub`

command. For example, the command below requests a two-hour connection of 24 cores on the "sJet" nodes using the account "dtc-hurr".

```
qsub -X -I -l procs=24,walltime=2:00:00,partition=sjet -A dtc-hurr
```

The user should seek assistance from the system administrator on how to connect to the computation nodes on the machine used to run HWRF.

Parallel code can also be submitted to the computation nodes using a batch system. For a platform that uses the batch system Load Sharing Facility (LSF), the beginning of each wrapper script should be edited to add the LSF options listed below.

```
#BSUB -P 99999999                  # Project 99999999
#BSUB -a poe                       # select poe
#BSUB -n 202                       # number of total (MPI) tasks
#BSUB -R "span[ptile=32]"          # run a max of 16 tasks per node
#BSUB -J hwrf                      # job name
#BSUB -o hwrf.%J.out               # output filename
#BSUB -e hwrf.%J.out               # error filename
#BSUB -W 2:30                      # wallclock time
#BSUB -q debug                     # queue
#BSUB -K                           # Don't return prompt until the job is finished
```

For a platform that uses the MOAB/Torque batch system, the beginning of each wrapper script should be edited to add the PBS options listed:

```
#PBS -A project                    # Project name
#PBS -l procs=202                  # number of total (MPI) tasks
#PBS -o stdout.txt                 # Output filename
#PBS -e stder.txt                  # Error filename
#PBS -N hwrf                       # Job name
#PBS -l walltime=02:30:00          # Wallclock time
#PBS -q batch                      # Queue name
#PBS -d .                          # Working directory of the job
```

After the batch system options and environment variables are defined, run the wrapper scripts using the command:

- On machines with LSF:
  `bsub < sample_wrapper`

- On machines with MOAB/Torque:
  `qsub sample_wrapper`

The wrapper script `sample_wrapper` will be submitted to the computation nodes and, once it starts, will call the low-level script from the `scripts/` directory. Appendix A contains the guidelines for resources used to run HWRF at near operational efficiency.

3.5.2 Running HWRF End-to-End

Once all configure files have been edited to define an experiment, the wrapper scripts should be submitted. Some of the wrappers have dependencies on previous wrappers, while others can be run simultaneously. In the following list, the items under the same number can be submitted together in any order, but only after the previous numbered item(s) runs to completion.

1. `launcher_wrapper`
2. `init_gdas_wrapper`
   `init_gfs_wrapper`
   `init_ocean_wrapper`
3. `relocate_wrapper`
4. `gsi_d02_wrapper`
   `gsi_d03_wrapper`
5. `merge_wrapper`
6. `unpost_wrapper`
7. `forecast_wrapper`
   `post_wrapper`†
   `products_wrapper`†

†Wrapper can run at the same time as `forecast_wrapper`, but should only be submitted after the forecast job has started, i.e. no loner in queue.

Wrappers with the same number may be run sequentially or simultaneously. Because the forecast job often waits in the queue before it starts, a post job submitted at the same time as the forecast job will have nothing to do for quite a while and will use wallclock time waiting on output from the forecast. Therefore, it is suggested to submit the post and products wrappers after the forecast job has started running.

By default, the `launcher_wrapper` reads in the configure files `system.conf`, `hwrf.conf`, `hwrf_input.conf`, and `hwrf_basic.conf`. Additional variables and configure files can be passed to `exhwrf_launcher.py` within the `launcher_wrapper` by following the syntax documented within the Python script. If anything is defined more than once, the variable will take the value that was last passed to `exhwrf_launcher.py`.

3.6 Running HWRF in Non-operational Configurations

Users may wish to run HWRF with a subset of the components used in operations. This section will describe a few of the alternative options that are supported using HWRF v3.6a.

3.6.1 Running without Spectral Files (GRIB Only)

In order to run with GRIB input files only, the `hwrf_basic.conf` file should be edited to set the following variables.

```
use_spectral=no
run_gsi=no
```

When running with GRIB files only, GSI data assimilation is not supported and should also be turned off in the configure file. Before submitting `launcher_wrapper`, edit the file to add the `parm/hwrf_gfs_pgrb2.conf` configure file as the last argument to the line calling the `exhwrf_launcher.py` script. The wrapper scripts `init_gdas_wrapper`, `gsi_d02_wrapper`, `gsi_d03_wrapper`, and `merge_wrapper` should be skipped. The relocation of the vortex will take place on the GFS input only. Note that the output will be altered to reflect these changes and not all files present in the operational configuration documented in later chapters will be produced.

3.6.2 Running an Uncoupled Forecast

Coupling with the ocean is only supported on the North Atlantic and Eastern North Pacific basins. In order to run an uncoupled forecast (no ocean) in any basin, you must first edit `hwrf_basic.conf` to set `run_ocean=no`. For uncoupled forecasts, the `init_ocean_wrapper` should not be run. This will change the output so that not all the files documented in later chapters will be produced.

3.6.3 Running without GSI

Running without GSI is nearly identical to the option presented in Section 3.6.1 for running with only GRIB input files, with the exception that `use_spectral` can be set to "yes" in `hwrf_basic.conf`. The following wrappers need not be run: `init_gdas`, `gsi_d02_wrapper`, `gsi_d03_wrapper`, and `merge_wrapper`. The output from this configuration will not match that documented in later chapters.

3.6.4 Running without Relocation

HWRF v3.6a is not configured to run with GSI and without relocation, i.e. the `hwrf_basic.conf` variables cannot be set as follows.

```
run_gsi=yes
run_relocate=no
```

If the user would prefer to run HWRF without vortex relocation, GSI must also be turned off.

See Section 3.6.3 for specifics on running without GSI; in addition, the `relocate_wrapper` should be skipped.

4

HWRF Preprocessing System

4.1 Introduction

HWRF needs data from the operational GFS and GDAS for its initialization procedures. Ultimately, the GFS dataset is used to create initial and boundary conditions for the 27-km outer domain, while the GDAS dataset is used to initialize the inner 9- and 3-km domains. However, as will be explain later, the GDAS analysis and forecast is also used in the 27-km domain for an intermediate step.

The GFS analysis and forecast employed are from the same cycle as the HWRF initialization (e.g., to initialize a HWRF forecast at 12 UTC, the 12 UTC run of GFS is used). However, the GDAS analysis and forecast used by HWRF is from the 6-h previous cycle (e.g., to initialize a HWRF forecast at 12 UTC, the 06 UTC run of GDAS is used). This differentiation is related to the data assimilation procedures (Chapter 6). First of all, data assimilation in HWRF is only conducted in the inner 9- and 3-km domains, and a forecast from the GDAS initialized 6-h before the HWRF analysis provides a better first guess than the GDAS analyses, in which observations have already been included as part of the global data assimilation procedures. Second, the HWRF Data Assimilation System (HDAS) ingests observations in a 3-h time window centered in the HWRF analysis time. This requires the availability of three first-guess files, one at the HWRF analysis time, one before and one after. By using the GDAS forecast initialized 6-h before the HWRF analysis, HWRF can make use of the GDAS 3-, 6-, and 9-h forecast lead times, which are valid at 3 h before the HWRF initialization, at the time of the HWRF initialization, and 3 h after the HWRF initialization, respectively. This procedure is termed FGAT, or First Guess at Appropriate Time.

HWRF employs the WRF model to downscale the GDAS forecasts to the 9- and 3-km grids for the vortex relocation and data assimilation procedures. This is done by using preprocess-

	D01	**D02**	**D03**
Grid spacing (deg)	0.18 (27 km)	0.06 (9 km)	0.02 (3 km)
HWRF Forecast	216 x 432 80°x 80°	106 x 204 12°x 12°	198 x 354 7.1°x 7.1°
Analysis run	216 x 432 80°x 80°	106 x 204 12°x 12°	198 x 354 7.1°x 7.1°
Ghost run	216 x 432 80°x 80°	166 x 336 24°x 24°	250 x 500 20°x 20°
3X domain			748 x 1504 30°x 30°

Table 4.1: Resolution (first row), number of grid points (top number in cell), and size (bottom number in cell) of the HWRF atmospheric grids.

ing utilities to interpolate the GDAS datasets to the HWRF 27-km domain, and then running two uncoupled 90-s WRF runs with three domains. These runs output "analysis" files, which are WRF restart files at t=0 (analysis time), and are referred to as the WRF Analysis run and the WRF Ghost run.

In HWRF operations at the National Weather Service, it is important that safeguards are put in place to prevent model failure. To account for the possibility that the GDAS dataset may be unavailable, WRF Ghost and WRF Analysis runs using the GFS dataset as initial conditions are also performed and can be used as a backup. In general, GDAS data is available, and the WRF Ghost and WRF Analysis runs initialized from the GFS are not ultimately used in the forecast process.

The WRF Analysis run has the main purpose of downscaling the global information to the HWRF high-resolution grids for use in the vortex relocation procedure discussed in further detail in Chapter 5. The WRF Ghost run downscales the global information to high-resolution grids that are slightly larger than their WRF forecast domain counterparts to provide first-guesses for the data assimilation procedure discussed further in Chapter 6. Depending on the domain, the WRF Ghost run is referred to as Ghost domain 2 (ghost_d02, the d02 counterpart) or Ghost domain 3 (ghost_d03, the d03 counterpart). Table **??** describes the grid spacing and size of the domains used in the HWRF main forecast run, the WRF Analysis, and the WRF Ghost runs. These domains are shown in Figure 4.1.

The HWRF includes two preprocessing packages to generate input files for WRF, the WRF Preprocessing System (WPS) and prep_hybrid. WPS consists of three programs to process input: geogrid interpolates static geographical data to the three HWRF domains; ungrib extracts meteorological fields from GRIB-formatted files and writes the fields to intermediate files; and metgrid horizontally interpolates the meteorological fields extracted by ungrib to the parent HWRF grid. The prep_hybrid utility horizontally interpolates the atmospheric fields represented as spectral coefficients in the global model files in binary format and native sigma vertical levels to the parent HWRF grid. The output of prep_hybrid and metgrid are utilized by the program real_nmm to vertically interpolate meteorological information to the HWRF vertical levels, resulting in a full set of 3D initial conditions that constitute the required WRF input. Both prep_hybrid and WPS are required because prep_hybrid

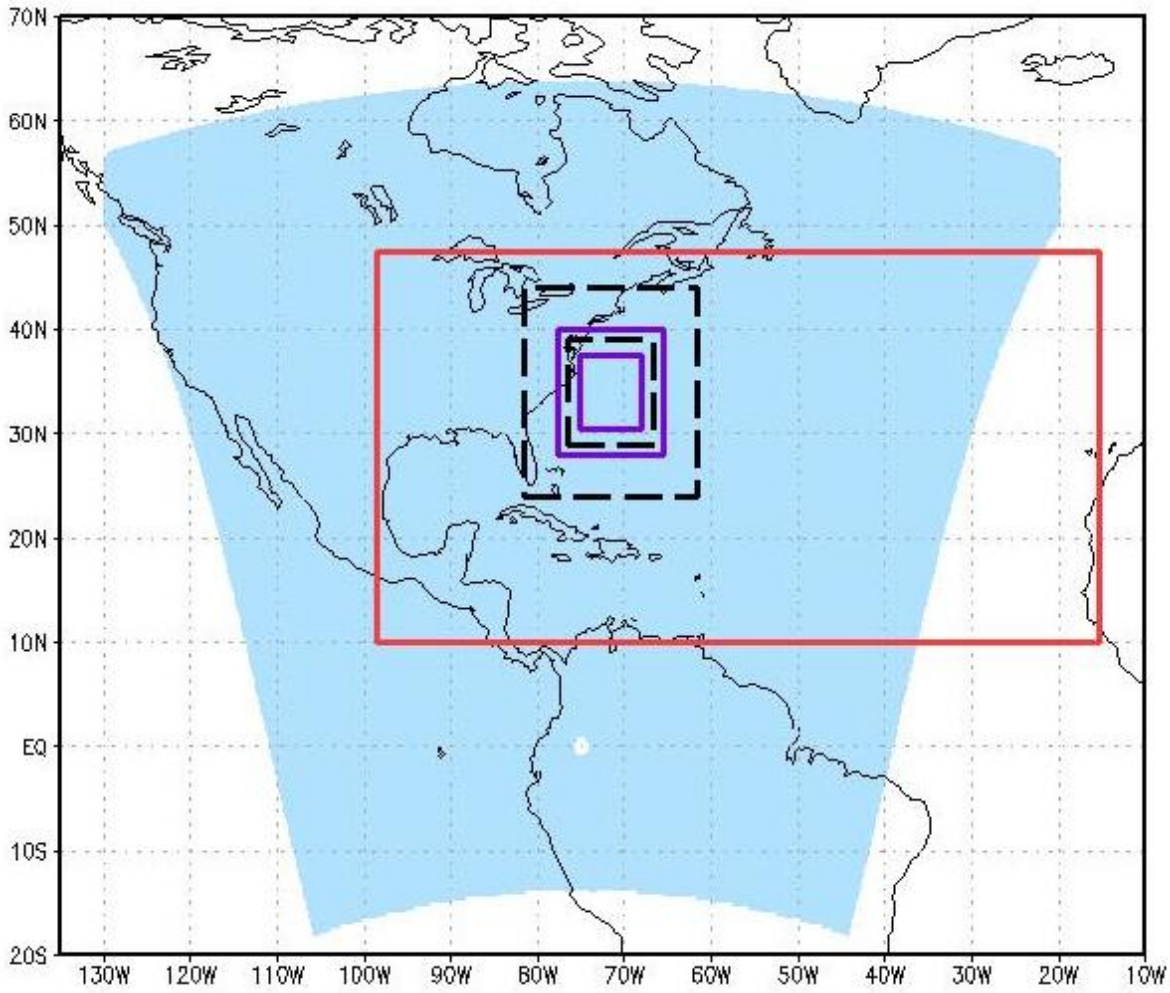

Figure 4.1: Example of the domains used by HWRF in the North Atlantic Basin. The blue region is the outer 27-km domain. The purple solid boxes show the sizes of the vortex-following 9-km and 3-km domains, while the black dashed lines are the ghost domains for d02 and d03. The red box is the unified Atlantic MPIPOM-TC domain.

only processes atmospheric data, so WPS is used to supply the initial conditions for soil temperature and moisture, as well as to supply the WRF model with the static information (topography, vegetation, etc.).

For general information about working with WPS, see the WRF-NMM documentation at

`http://www.dtcenter.org/wrf-nmm/users/docs/user_guide/V3/users_guide_nmm_chap1-7.pdf`

As part of the vortex initialization procedure described in Chapter 5, the vortex from the processed global model analysis is removed and substituted with an improved vortex. In order to locate the vortex in the global model, the GFDL Vortex Tracker is run at the analysis time on the post-processed 90-s output from the WRF Analysis run.

This chapter explains how to run the initialization procedure, including the launcher, WPS, prep_hybrid, real_nmm, WRF Analysis, and WRF Ghost to create the HWRF preliminary initial conditions.

4.2 Scripts

Three wrapper scripts are used to preprocess data for the atmospheric component of HWRF.

```
launcher_wrapper
init_gdas_wrapper
init_gfs_wrapper
```

The launcher wrapper calls `scripts/exhwrf_launch.py` to read the configuration files, set the output directory structures, and determine the location of the outer 27-km domain. The "init" wrapper scripts call the Python script `scripts/exhwrf_init.py`. For processing the GFS data, `exhwrf_init.py` is called once, and then three more times for processing GDAS data – once at each FGAT hour (3, 6, 9). Figures 4.2, 4.3, and 4.4 show the simplified outline of the processes that occur at each FGAT hour for both the GFS and GDAS initialization. The script `exhwrf_init.py` runs the three stages of WPS (geogrid, ungrib, and metgrid), prep_hybrid, real_nmm (to create initial and boundary conditions for the WRF Ghost and WRF Analysis runs), wrfghost, wrfanalysis, post, gribber, tracker, and realfcst (to create LBCs for the main forecast run). All these steps are needed for WRF initialization, and some of these are used again at later stages of the run (for example, post and tracker).

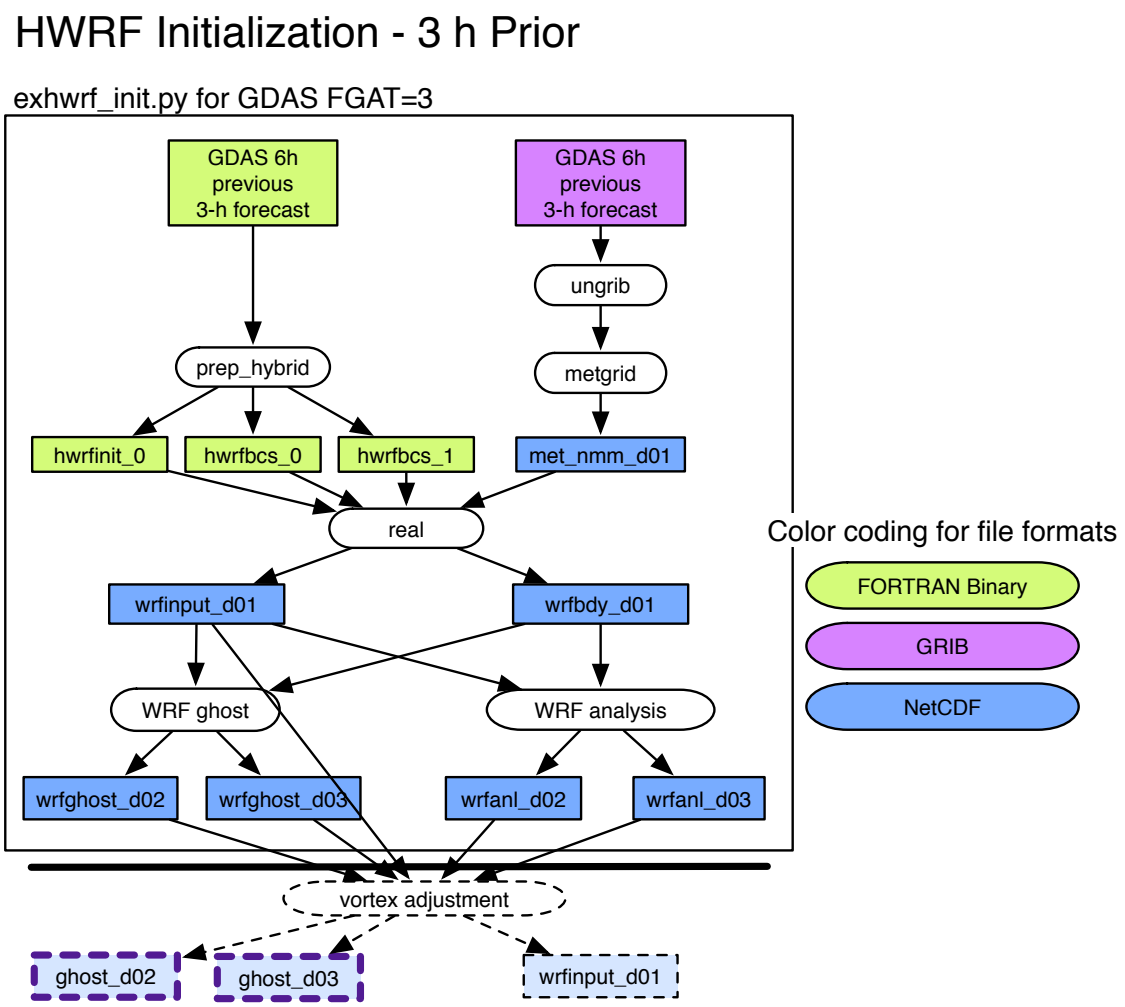

Figure 4.2: Simplified initialization procedures for the FGAT=3 valid 3 hours prior to HWRF initialization. All processes in the black box are run by calling `exhwrf_init.py` for GDAS at FGAT=3. Boxes with dashed outlines indicate modules and resulting files that are discussed in Chapter 5. Files that are outlined in heavy purple (dashed or solid) are used by subsequent processes described by Figure 6.1.

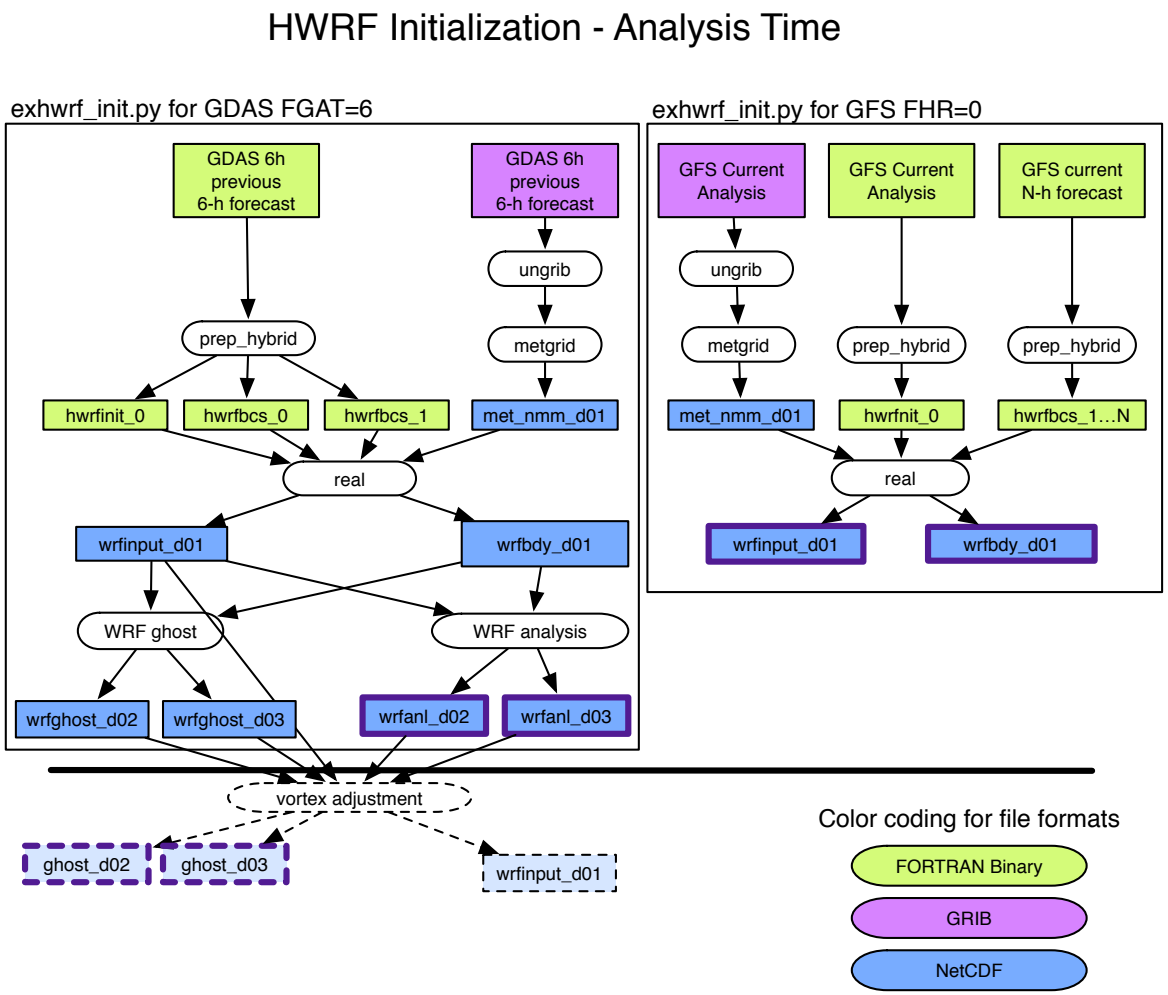

Figure 4.3: Simplified initialization procedures for the HWRF analysis time. All processes in the black box on the left are run by calling `exhwrf_init.py` for GDAS at FGAT=6, while the right black box are procedures from running `exhwrf_init.py` for GFS at analysis time. Boxes with dashed outlines indicate modules and resulting files that are discussed in Chapter 5. Files that are outlined in heavy purple (dashed or solid) are used by subsequent processes described by Figure 6.1.

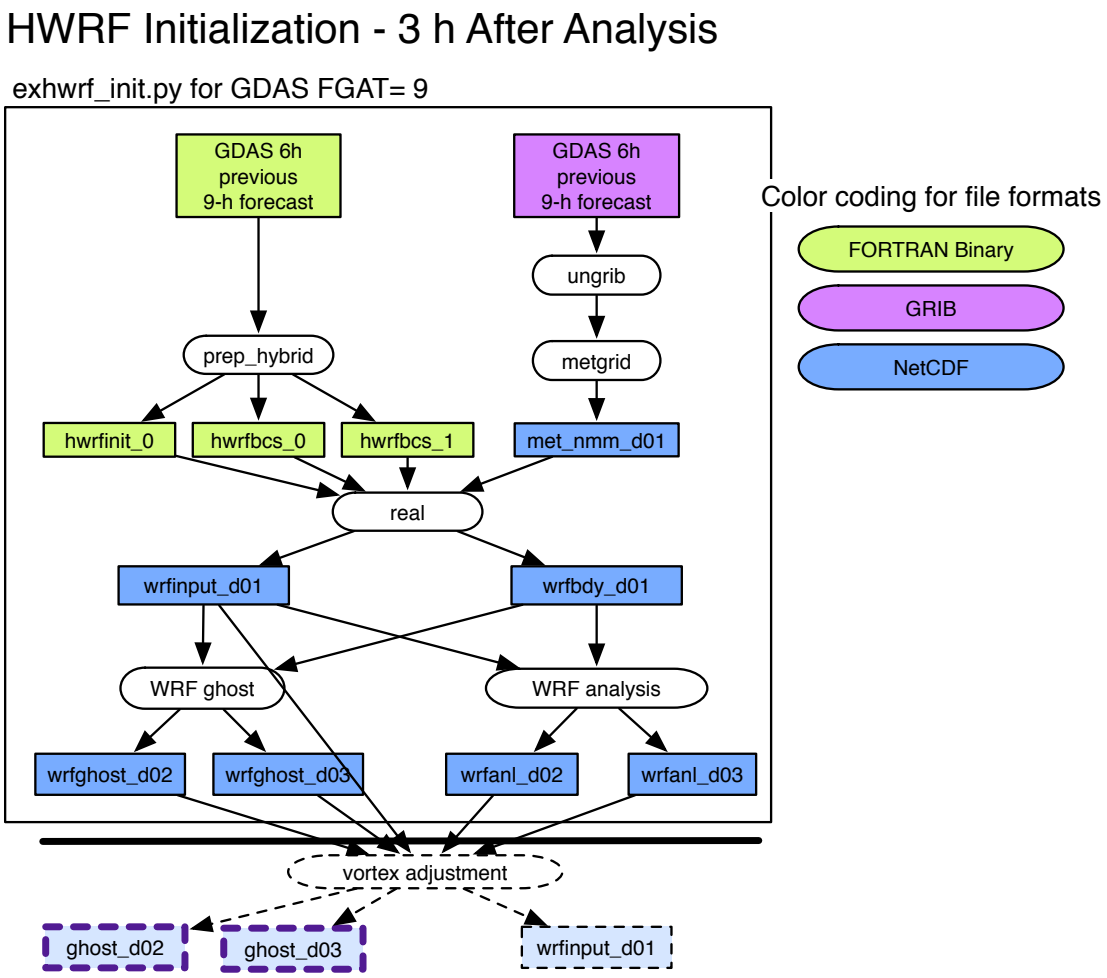

Figure 4.4: Simplified initialization procedures for the FGAT=9 valid 3 hours after HWRF initialization. All processes in the black box are run by calling `exhwrf_init.py` for GDAS at FGAT=9. Boxes with dashed outlines indicate modules and resulting files that are discussed in Chapter 5. Files that are outlined in heavy purple (dashed or solid) are used by subsequent processes described by Figure 6.1.

4.2.1 Overview of `exhwrf_launch.py`

1. Read in configure files from `parm/` directory
2. Set the paths to the directories containing HWRF source code and to the Python scripts (`$HOMEhwrf` and `$USHhwrf`, respectively)
3. Set the storm ID
4. Initialize the directory structure for the HWRF workflow
5. Locate and extract TC Vitals for the current storm and cycle, then write information to `$WORKhwrf`
6. Using TC Vitals, determine the domain center and write output to file `storminfo.hwrf_domain_center`

7. Parse the configuration files and write configure and holdvars files to com/
8. Write a startfile to the launch directory (i.e. wrappers/)

Output files:

$startfile	Name is defined in global_vars.ksh. Contains environment variables for the storm ID and paths to output data
storm1.conf	Located in com/ directory. Contains configuration information compiled from all configure files
storm1.holdvars.txt	Located in com/ directory. Only used in operational implementation of HWRF
SID.*YYYYMMDDHH*.domain.center	Coordinates of the domain center. *SID* is the storm ID, i.e. 18L for Sandy (2012), and *YYYYMMDDHH* is the analysis time

Status check:

The file $startfile defined in global_vars.ksh has been written in the wrappers/ directory.

Usage:

The following options are mandatory when running exhwrf_launch.py.

exhwrf_launch.py *YYYYMMDDHH STID case_root* **/path/to/parm**

In the previous command, *YYYYMMDDHH* should be replaced by the date to run, *STID* represents the storm ID, i.e. 18L for Sandy (2012), and *case_root* is replaced with HISTORY for retrospective runs and FORECAST for real-time runs. Many additional arguments are provided within the wrapper, but the options provided do not constitute an exhaustive list of arguments that could be passed to the Python script. Additional arguments may be included with the submission of exhwrf_launch.py by editing the launcher_wrapper. The launcher_wrapper included with HWRF v3.6a submits exhwrf_launch.py with the following arguments.

${HOMEhwrf}/scripts/exhwrf_launch.py *YYYYMMDDHH STID* **HISTORY **
**${HOMEhwrf}/parm config.startfile=***STARTFILE* **dir.HOMEhwrf=${HOMEhwrf} **
config.EXPT=*EXPT* **config.case_root=HISTORY**

In the previous command, ${HOMEhwrf} is set in the wrapper to the path where HWRF is installed, STARTFILE is the launcher output file, and EXPT is the chosen experiment name. The wrapper passes many of the values through environment variables, but values can also be directly passed.

There are three ways to pass configuration variables to the script exhwrf_launch.py.

- Customize the variables in the configuration files in the parm/ directory.
- Create your own configuration files and pass them to the script by adding the path to the additional files.

../parm/my_hwrf_config.conf

- Pass configuration variables in the command line. The following example would turn off GSI data assmilation.

  config.run_rungsi=no

4.2.2 Overview of the Init Scripts: `exhwrf_init.py` and Wrappers

In the following list, top-level numbers are calls from the wrapper, either `init_gdas_wrapper` or `init_gfs_wrapper`, the alphabetic level represents calls from the script `exhwrf_init.py`, and the lowest level are calls to modules from within `init.py`.

1. Initialize the objects used to run all components of HWRF by calling `hwrf_expt.init_module()`
2. Call `exhwrf_init.py` to process the global input files using INIT_PARTS=ALL four times. Once for GFS at analysis time from `init_gfs_wrapper`, then once for each FGAT hour for GDAS from `init_gdas_wrapper`
   a) Run initialization steps leading up to the WRF Analysis (run_through_anl())
      i. geogrid
      ii. ungrib
      iii. metgrid
      iv. prep_hybrid
      v. realinit
      vi. runwrfanl
   b) Run additional initialization steps needed for data assimilation (run_init_after_anl())
      i. ghost
      ii. post
      iii. gribber
      iv. tracker
   c) Run steps necessary to generate the LBCs for the main forecast (run_real_bdy())
      i. ungrib
      ii. metgrid
      iii. prep_hybrid
      iv. realfcst

4.2.3 Overview of Initialization Modules

Geogrid

The run module for a Geogrid object resides in `ush/hwrf/wps.py`. The module performs the following tasks.

1. Link the `GEOGRID.TBL` and `geog_data/` fixed files
2. Create the namelist

3. Run `geogrid.exe`

Output files:

`geo_nmm.d01.nc`	Static geographical data for the parent domain, with grid spacing of 0.18 degrees.
`geo_nmm_nest.l01.nc`	Static geographical data that covers the parent domain, with grid spacing of 0.06 degrees.
`geo_nmm_nest.l02.nc`	Static geographical data that covers the parent domain, with grid spacing of 0.02 degrees.

Status Check:

In the standard out file of an initialization task, you will find the line `"CRITICAL: WPS Geogrid completed."`

Executables:

`geogrid.exe`

FUNCTION	Interpolates static geographical data to the parent and nest grids
INPUT	Fix files from `${GEOG_DATA_PATH}` `GEOGRID.TBL` - defines parameters of input data sets `namelist.wps` - WPS namelist

OUTPUT	`geo_nmm.d01.nc`
	`geo_nmm_nest.l01.nc`
	`geo_nmm_nest.l02.nc`

USAGE	**`geogrid.exe`**

PrepHybrid

The run module for the PrepHybrid object is located in `ush/hwrf/prep.py`. The module preforms the following tasks.

1. Copy the input files
2. Run `hwrf_prep.exe`
3. Link the output files

Output files for ICs:

`hwrfinit_00`	Global model spectral data preprocessed by `hwrf_prep.exe` and ready to be used by real_nmm to generate preliminary HWRF ICs.

Output files in for LBCs:

`hwrfbcs00_${bc_index}`	Global model spectral data preprocessed by `hwrf_prep.exe` and ready to be used by real_nmm to generate preliminary HWRF LBCs. The variable bc_index=0,1,..,n corresponds to the forecast files that need to be preprocessed to create the LBCs for the HWRF forecast.

Status Check:

In the standard output file of the initialization task, you will find the line `"INFO: - exit status 0"` for the task `"prep_hybrid"`.

Executables:

`hwrf_prep.exe`

FUNCTION Preprocesses the GDAS or GFS spectral data on vertical sigma levels in binary format for use the by real_nmm

INPUT `geogrid.out` - link to geo_nmm.d01.nc
`prep_hybrid.nl` - prep_hybrid namelist
`fort.11` - link to `gfsbc${bc_index}`
`fort.44` - link to `itime` file contains `$bc_index`
`fort.45` - link to `domain.center` file
`gfsbc`*`${bc_index}`* - link to the global spectral file `[gdas1|gfs].${BKG_START_TIME}.sf${BKG_FCST_TIME}` where `$BKG_START_TIME` is the GDAS or GFS initialization time and `$BKG_FCST_TIME` is time the GDAS or GFS forecast lead time. For example to create the 12-h LBCs for the HWRF forecast initialized at 2012102806, the GFS initialized at the HWRF analysis time is used, and these variables would be set to the following.

```
BKG_START_TIME = 2012102806
BKG_FCST_TIME = 012
```

OUTPUT `hwrfinit_0`
`hwrfbcs00_${bc_index}`

USAGE **`$PREP_EXE $NX1 $NY1 $VERT_LEV $DXX $DYY`**
where `$NX1`, `$NY1`, and `$VERT_LEV` are the output file grid dimensions in the meridional, zonal and vertical directions, and `$DXX` and `$DYY` are the horizontal grid spacing.s

Ungrib

The run module for an Ungrib object resides in `ush/hwrf/wps.py`. The module performs the following tasks.

1. Create the namelist
2. Link the `Vtable` and input GRIB files
3. Run `ungrib.exe`

Output files:

The intermediate files written by `ungrib.exe` will have names of the form `FILE:`*`YYYY-MM-DD_HH`* (unless the prefix variable in `hwrf.conf` was set to a prefix other than `"FILE"`).

Status Check:

In the standard out file of an initialization task, you will find the line `"CRITICAL: WPS Ungrib completed."`

Executables:

`ungrib.exe`

FUNCTION	Extracts meteorological fields from GRIB formatted files and writes the fields to intermediate files
INPUT	GRIB files `Vtable` - codes to interpret GRIB files `namelist.wps` -WPS namelist
OUTPUT	`FILE:`*`YYYY-MM-DD_HH`*
USAGE	**`ungrib.exe`**

Metgrid

The run module for a Metgrid object resides in `ush/hwrf/wps.py`. The module performs the following tasks.

1. Create the namelist
2. Link the metgrid table
3. Copy in the output from geogrid and ungrib
4. Run `metgrid.exe`

Output files:

met_nmm.d01.*YYYY-MM-DD_HH:MM:SS.nc*	*YYYY- MM-DD_HH:MM:SS* refers to the valid date of the interpolated data in each file.

Status Check:

In the standard out file of an initialization task, you will find the line "CRITICAL: WPS Metgrid completed."

Executables:

metgrid.exe

FUNCTION	Horizontally interpolates the meteorological fields extracted by ungrib to the model parent grid
INPUT	METGRID.TBL - parameters for interpolating each field geo_nmm.d01.nc - output of geogrid namelist.wps - WPS namelist FILE:*YYYY-MM-DD_HH* - output of ungrib
OUTPUT	met_nmm.d01.*YYYY-MM-DD_HH:MM:SS.nc*
USAGE	**metgrid.exe**

Realinit

Realinit is a WRFTask object and its run module resides in ush/hwrf/wps.py. The module performs the following tasks.

1. Link the input and fixed files
2. Run hwrf_swcorner_dynamic.exe to calculate the istart and jstart values for a nest
3. Generate the namelist
4. Run real_nmm.exe to generate initial and boundary conditions. A high-resolution sea-mask data file (fort.65) for the entire outer domain is also generated. It is later used by the coupler.

Output files:

wrfinput_d01	ICs created from GDAS
wrfbdy_d01	LBCs created from GFS for the ghost and analysis runs
fort.65	High-resolution sea mask data

Status Check:

In the standard output file, you will find the line "/realinit: completed".

Executables:

`real_nmm.exe`

FUNCTION	Generates the initial and boundary conditions
INPUT	Fixed files† `hwrfbcs_[0-21]` - BC output of prep_hybrid `hwrfinit_0` - IC output of prep_hybrid `geo_nmm.d01.nc` - d01 output from geogrid `geo_nmm.l01.nc` - d02 output from geogrid `geo_nmm.l02.nc` - d03 output from geogrid `met_nmm.d01.YYYY-MM-DD_HH:00:00.nc` - d01 output from metgrid

OUTPUT	`fort.65`
	`wrfbdy_d01`
	`wrfinput_d01`

USAGE	**`real_nmm.exe`**

`hwrf_swcorner_dynamic.exe`

FUNCTION	Calculates the lower-left corner of a nest as (`i_parent_start`, `j_parent_start`)
INPUT	`storm.center` - storm center location `domain.center` - domain center location `fort.12 (namelist_main.input)`

OUTPUT	`set_nest`, which contains the `i_parent_start` and `j_parent_start`. For example the following set_nest file specifies that the middle nest domain lower-left corner location is at (99,225) on the parent domain grid.
	`istart=00099`
	`jstart=00225`

USAGE	**`hwrf_swcorner_dynamic.exe`**

Realfcst

The process for `realfcst` is the same as `realinit`, except that `realfcst` runs for the length of the HWRF forecast, instead of only at the analysis time.

Runwrfanl

Runwrfanl is a WRFAnl4Trak object whose run module resides in `ush/hwrf/fcsttask.py`. The module performs the following tasks.

1. Link the input and fixed files
2. Run `hwrf_swcorner_dynamic.exe` to calculate the istart and jstart values for a nest
3. Generate the namelist.input for grids identical to the HWRF forecast grids
4. Run `wrf.exe` to make a 90-s run of WRF and generate two analysis output files

Output files:

`wrfanl_d02_`*`YYYY-MM-DD_HH:HH:HH`* – "analysis" file for 9-km nest
`wrfanl_d03_`*`YYYY-MM-DD_HH:HH:HH`* – "analysis" file for 3-km nest

Status Check:

In the standard output file, you will find the line `"/wrfanl:  completed"`.

Executables:

`wrf.exe`

FUNCTION	Atmospheric model component of HWRF
INPUT	Fixed files[†] `geo_nmm.d01.nc` – d01 output from geogrid `geo_nmm.l01.nc` – d02 output from geogrid `geo_nmm.l02.nc` – d03 output from geogrid `wrfinput_d01` – ICs for parent domain from realinit `wrfbdy_d01` – LBCs for parent domain from realinit `namelist.input` – WRF namelist
OUTPUT	`wrfanl_d02_`*`YYYY-MM-DD_HH:HH:HH`* `wrfanl_d03_`*`YYYY-MM-DD_HH:HH:HH`*
USAGE	**`wrf.exe`**

Runghost

Runghost is a WRFGhost object whose run module resides in `ush/hwrf/fcsttask.py`. The module performs the following tasks.

1. Link the input and fixed files
2. Run `hwrf_swcorner_dynamic.exe` to calculate the istart and jstart values for a nest
3. Generate the namelist.input for the ghost domains

4. Run `wrf.exe` to make a 90-s run of WRF and generate two analysis output files

Output files:

`wrfanl_d02_`*`YYYY-MM-DD_HH:HH:HH`* – "ghost" analysis file for 9-km nest

`wrfanl_d03_`*`YYYY-MM-DD_HH:HH:HH`* – "ghost" analysis file for 3-km nest

Status Check:

In the standard output file, you will find the line `"/ghost: completed"` where YYYYMMDDHHHH is the real time of the FGAT hour.

Executables

`wrf.exe`

FUNCTION	Atmospheric model component of HWRF
INPUT	Fixed files† `geo_nmm.d01.nc` – d01 output from geogrid `geo_nmm.l01.nc` – d02 output from geogrid `geo_nmm.l02.nc` – d03 output from geogrid `wrfinput_d01` – ICs for parent domain from realinit `wrfbdy_d01` – LBCs for parent domain from realinit `namelist.input` – WRF namelist
OUTPUT	`wrfanl_d02_`*`YYYY-MM-DD_HH:HH:HH`* `wrfanl_d03_`*`YYYY-MM-DD_HH:HH:HH`*
USAGE	**`wrf.exe`**

†Fixed files:

```
eta_micro_lookup.dat
hwrf_track
aerosol.formatted
aerosol_lat.formatted
aerosol_lon.formatted
aerosol_plev.formatted
co2_trans
ETAMPNEW_DATA
ETAMPNEW_DATA_DBL
ETAMPNEW_DATA.expanded_rain
ETAMPNEW_DATA.expanded_rain_DBL
GENPARM.TBL
LANDUSE.TBL
MPTABLE.TBL
ozone.formatted
ozone_lat.formatted
ozone_plev.formatted
RRTM_DATA
RRTM_DATA_DBL
RRTMG_LW_DATA
RRTMG_LW_DATA_DBL
RRTMG_SW_DATA
RRTMG_SW_DATA_DBL
SOILPARM.TBL
tr49t67
tr49t85
tr67t85
URBPARM.TBL
URBPARM_UZE.TBL
VEGPARM.TBL
```

Post

Post is a PostOneWRF object whose run module resides in `ush/hwrf/post.py`. This instance of the module runs `unipost.exe` on the output of the 90-s WRF Analysis run. The purpose of this step is to de-stagger the HWRF native output, interpolate it vertically to pressure levels, compute derived variables, and output the result in GRIB format. Further details about Post objects, modules, and executables can be found in Chapter 10.

Output:

In the `intercom/fgat.t`*`YYYYMMDDHHHH`*`/post` directory, you will find the following file.

`fgat.t`*`YYYYMMDDHHHH`*`_post-moad.egrb`

Status check:

The line `"INFO: state=COMPLETED"` can be found in the GDAS Init standard output file for the `"post"` task. Performing a search for both of quoted strings should return one line for each FGAT hour for GDAS, and once for GFS.

Gribber

Gribber is a GRIBTask object and serves to regrib the output of post, interpolating the 27-km parent domain GRIB file to a 20°x 20°grid. This file is used as input to the GFDL Vortex Tracker. See Chapter 11 for more details about GRIBTask objects, modules, and executables.

Output:

In the `intercom/fgat.t`*`YYYYMMDDHHHH`*`/regribber` directory, you will find the files below, where `stormname` and `sid` are the name of the storm and the SID in lower case (e.g., `sandy 18l` for Hurricane Sandy):

```
quarter_degree.grb
stormnamesid.YYYYMMDDHH.hwrftrk.grbf00
stornnamesid.YYYYMMDDHH.hwrftrk.grbf00.grbindex
subset.grb
```

Status check:

In the standard output file for the GDAS Init wrapper, you will find the following string on the same line as "regribber" for each FGAT hour: `"WARNING: No subtasks incomplete. I think I am done running. Will exit regribber now."`

Tracker

The GFDL Vortex Tracker is run on the GRIB file resulting from the gribber step above. The Tracker object resides in `ush/hwrf/tracker.py`. More information about the tracker can be found in Section 11.

Output:

In the `intercom/fgat.t`*YYYYMMDDHHHH* directory, you will find the file.

> `gfs.track0.atcfunix` – contains the storm center at initial time in the WRF analysis run output

Status Check:

In the standard output file for GDAS Init, you should fine the line `"WARNING: Successful return status from gettrk."` for each FGAT hour.

5

Vortex Relocation

5.1 Introduction

The atmospheric component of HWRF, WRF-NMM, needs ICs and LBCs to produce forecasts. The GFS and GDAS fields are used to create the preliminary atmospheric fields, which are further improved through the vortex adjustment procedures and data assimilation to provide the final IC.

The vortex adjustment procedures are necessary because the initial vortex is often not realistically represented in the preliminary ICs since it originates from a low-resolution global data source, such as GDAS. Therefore, HWRF employs a sophisticated algorithm to adjust the vortex to match the observed storm intensity, location, and structure.

Initial conditions for HWRF d02 and d03 are created by ingesting GDAS fields onto the HWRF vortex initialization procedure. To prepare the fields for input in the vortex initialization, two 90-s atmosphere-only forecasts are conducted. These runs are referred to as the WRF Analysis and WRF Ghost runs, and their configuration is detailed in Chapter 4. Within the vortex relocation code, the fields are interpolated to the 3X domain, a temporary domain with 0.02° grid spacing. For historical reasons, some file names and executables use 4X when referring to the 3X domain.

The HWRF vortex relocation process has three possible stages, which are determined based on the intensity of the observed storm and on the availability of the 6-h forecast of the previous HWRF run. Figure 5.1 describes Stages 1 and 2, and Figure 5.2 describes Stage 3. If the previous cycle HWRF forecast exists, and if the observed storm intensity is at least 14 ms^{-1}, HWRF is run in cycled mode. In cycled mode, the the 6-h forecast vortex from the previous HWRF cycle, adjusted according to the TC Vitals, is used for initializing the

current cycle. If those conditions are not met, the HWRF initialization is a "cold start".

For a cold start of storms with observed intensity less than 20 ms^{-1}, the GDAS vortex is adjusted and then used. Conversely, for storms with observed intensity greater than or equal to 20 ms^{-1}, a bogus vortex is used. A cycled run will go through all the three stages, while a "cold start" run will go through Stages 2 and 3 only.

Stage 1: The previous cycle 6-h HWRF forecast is separated into environment fields and a storm vortex. This step is run only for cycled cases.
Stage 2: The preliminary IC generated by real_nmm and the WRF ghost and analysis runs is separated into environment fields and a storm vortex.
Stage 3: The storm vortex from the 6-h forecast from the previous cycle (for cycled runs), from the GDAS, or from the bogus vortex is adjusted to match the observed location, intensity, and structure provided by the NHC for the current time. Then the vortex and environment fields are combined.

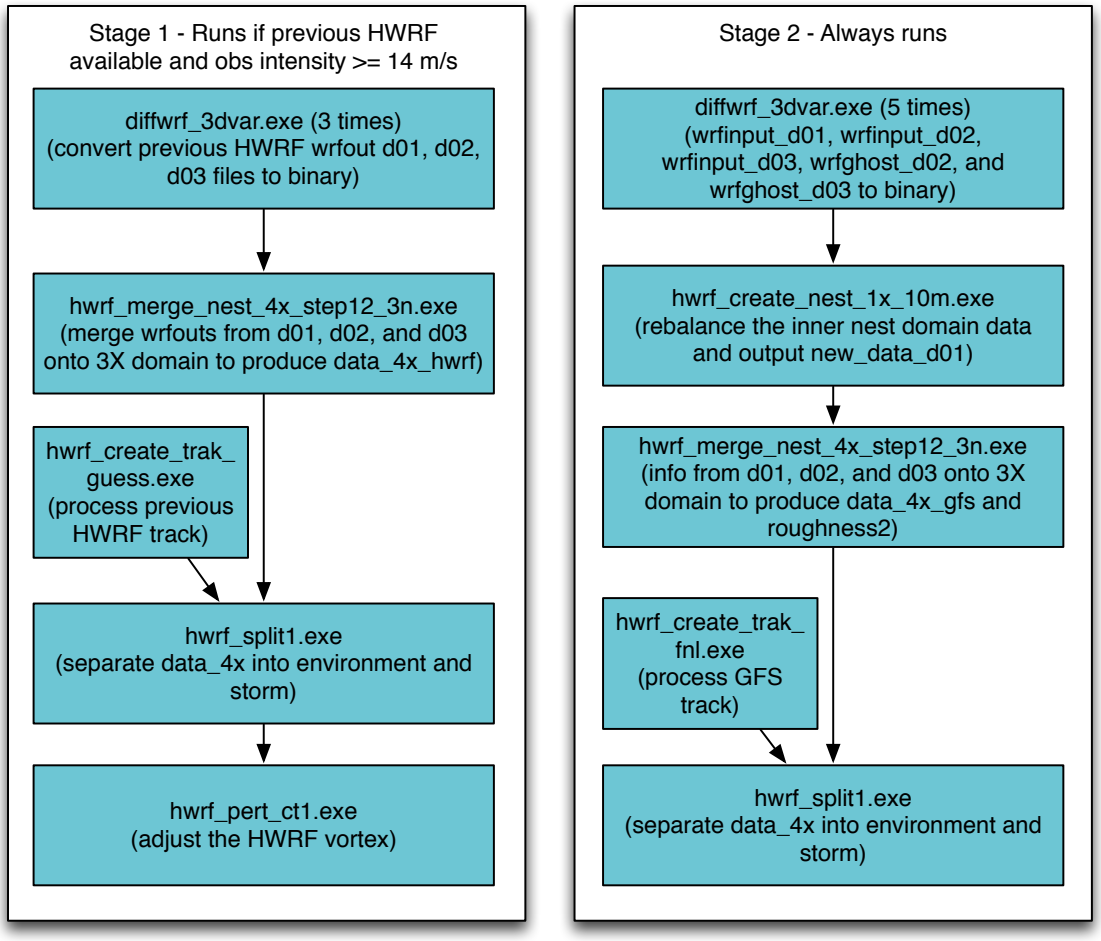

Figure 5.1: Simplified flow of Stages 1 and 2 of the vortex relocation process.

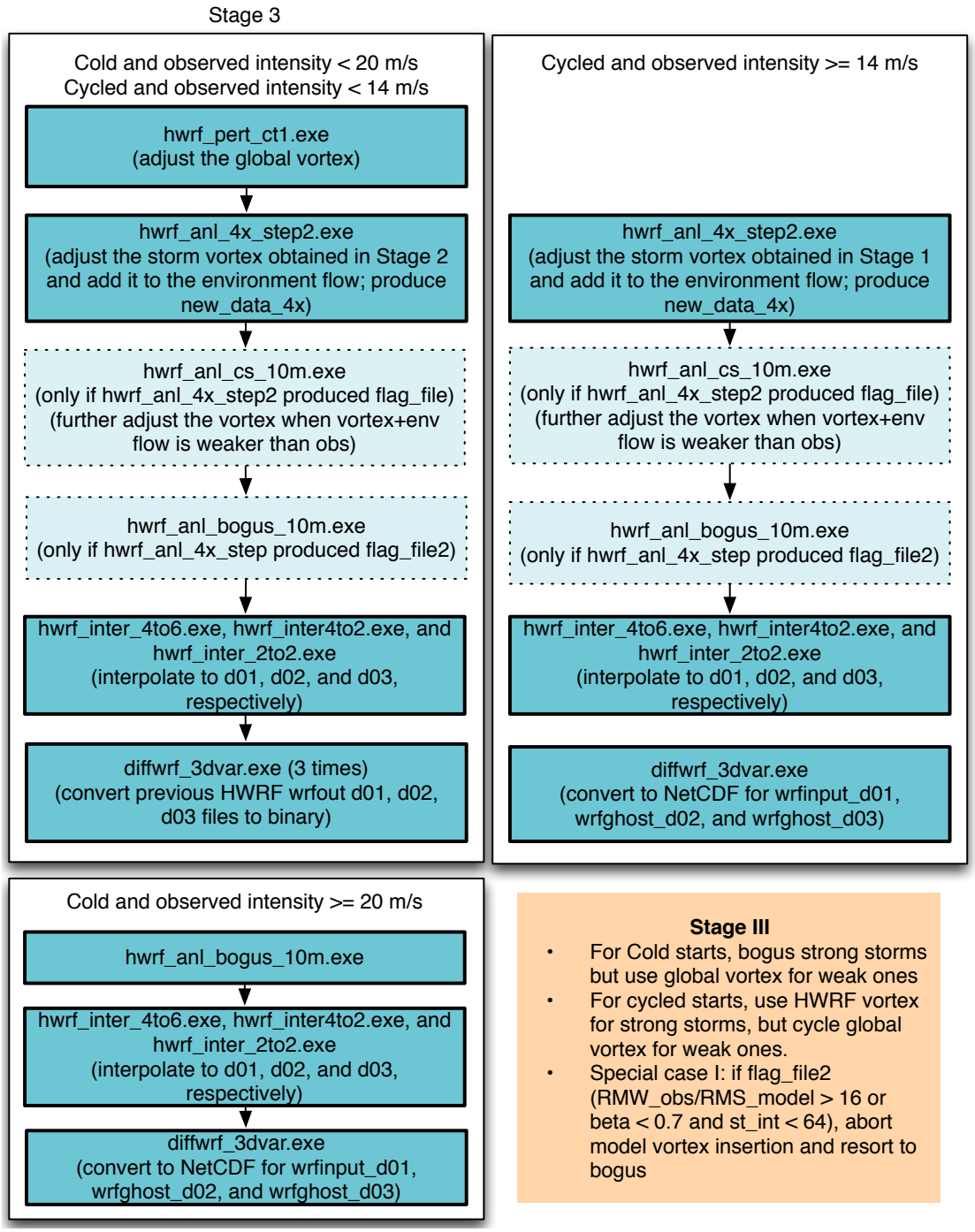

Figure 5.2: Simplified flow of Stage 3 of the vortex relocation process.

5.2 Scripts

The vortex improvement procedure is entirely driven by the wrapper script `relocate_wrapper`, which calls 4 instances of `scripts/exhwrf_relocate.py`. The first instance runs relocate on the 90-s WRF Analysis run initialized from the GFS analysis. The other three instances run relocate on the 90-s WRF Analysis runs created at each FGAT time for the domains initialized by the GDAS forecasts. If the relocation procedure using the GDAS-derived input files is successful, the relocate results from GFS-derived fields are discarded.

5.2.1 Overview of `exhwrf_relocate.py`

The numbered items in the following list indicate calls made from the `exhwrf_relocate.py` script, while the lower level list items are calls made within the Python modules.

1. Initialize the objects used to run all components of HWRF by calling `hwrf_expt.init_module()`.
2. If GFS: run relocate at analysis time
   a) Stage 1
   b) Stage 2
   c) Stage 3
3. If GDAS: run relocate for each FGAT time
   a) Stage 1
   b) Stage 2
   c) Stage 3

5.2.2 Overview of the Relocate Modules

Stage1

1. Copy the fixed files and input files to the working directory.
2. Check if the HWRF forecast from the previous cycle exists, and if the storm intensity is greater than 14 ms^{-1}; if not, continue to Stage 2.
3. Run `diffwrf_3dvar.exe` to convert the previous cycle forecast output `wrfout_d0[1-3]` into unformatted data files `old_hwrf_d0[1-3]` respectively.
4. Run `merge_nest_4x_step12_3n.exe` to merge `wrfout_d0[1-3]` onto 3X domain and produce a file containing the merged data: `data_4x_hwrf`.
5. Run `hwrf_create_trak_guess.exe` to produce a guess track (0,3,6,9 hour) for the current forecast using previous cycle forecast track.
6. Run `wrf_split1.exe` to separate `data_4x_hwrf` into two parts, an environment field (`wrf_env`) and a storm vortex (`storm_pert`). A storm radius data file

(storm_radius) is also generated.

7. Run hwrf_pert_ct1.exe to do adjustments to storm_pert. The new storm vortex data (storm_pert_new) as well as two files containing the storm size information (storm_size_p) and the symmetric part of the vortex (storm_sym) are generated.

Output files:

storm_size_p	Storm size information
storm_pert_new	New storm vortex after adjustments by hwrf_pert_ct1.exe
storm_sym	Symmetric part of the vortex
storm_radius	Storm radius information
wrf_env	Environment field

Status Check:

If the line "CRITICAL: Stage 1 completed" is found in the standard output, Stage 1 was successful.

Executables

diffwrf_3dvar.exe

This executable serves two functions, denoted by 1 or 2 below.

FUNCTION 1. Converts netCDF input to unformatted file (when first argument is "storm_relocate")
2. Updates existing netCDF file with new unformatted file (when first argument is "3dvar_update")

INPUT 1. netCDF format input files or previous cycle 6-h forecast
2. Unformatted file containing new vortex fields

OUTPUT 1. Unformatted data file
2. Updated netCDF file

USAGE 1. **diffwrf_3dvar.exe storm_relocate** *input_file* **flnm3** \
output_file
The command above writes the WRF file *input_file* into an unformatted file, *output_file*, which will be used in the vortex relocation procedures.
2. **diffwrf_3dvar.exe 3dvar_update** *input_file output_file*
The command above updates *input_file* with unformatted file *output_file*, which contains new vortex fields.

hwrf_merge_nest_4x_step12_3n.exe

FUNCTION Merges inner and outer domains onto a 3X domain

INPUT `$gesfhr (=6)` – last digit of the input/output fort file, i.e. fort.2**6**
`$st_int` – the 68-69 characters in the tcvital.as)
`$ibgs(=1)` – argument indicating if a cold start (`ibgs=1`) or a cycled run (`ibgs=0`)
`tcvitals.as (fort.11)` – observed storm center
`old_hwrf_d01` or `new_gfs_d01 (fort.26)`
`old_hwrf_d02` or `new_gfs_d02 (fort.36)`
`old_hwrf_d03` or `new_gfs_d03 (fort.46)`

OUTPUT `data_4x_hwrf (fort.56)` – merged data from inner and outer domains
`roughness1` or `roughness2 (fort.66)` – sea-mask (1=sea, 0=land) and ZNT (roughness length) merged onto the 3X domain.
`30_degree_data (fort.61)`: partially merged data from inner and outer domains (not used later)

USAGE **`echo $gesfhr $st_int $ibgs $BASIN | \`**
`hwrf_merge_nest_4x_10m2.exe`

`hwrf_create_trak_guess.exe`

FUNCTION Guesses storm center from previous 6-h forecast position

INPUT `$storm_id` – storm ID
`$ih` – model initial hour
`tcvitals.as (fort.11)` – observed storm center
`hdas_atcfunix (fort.12)` – track file from previous cycle 6-h forecast.

OUTPUT `trak.fnl.all (fort.30)` – storm center guess (at 0, 3, 6, 9 h)

USAGE **`echo $storm_id $ih $BASIN | hwrf_create_trak_guess.exe`**

`hwrf_split1.exe`

FUNCTION Splits the vortex from the background (environmental) field

INPUT `$gesfhr=6` – last digit of the input/output fort file, i.e. fort.2**6**
`$ibgs (=1)`
`$st_int` – the 68-69 characters in the tcvital.as
`tcvitals.as (fort.11)` – storm center obs
`data_4x_hwrf (fort.26)` – merged data, on 3X domain, from inner and outer domains
`trak.fnl.all (fort.30)` – storm center guess
`old_hwrf_d01 (fort.46)` – outer domain data

OUTPUT `wrf_env (fort.56)` - environmental flow
`storm_pert (fort.71)` - separated 3D vortex field
`storm_radius (fort.85)` - average of model and observed storm radius
`rel_inform.$cdate (fort.52)` - diagnostics file (obs-previous 6-h forecast)
`vital_syn.$cdate (fort.55)` - information for generating bogus if storm not found in previous 6-h forecast

USAGE **`echo $gesfhr $ibgs $st_int $BASIN | hwrf_split.exe`**

`hwrf_pert_ct1.exe`

FUNCTION Adjusts storm vortex (`storm_pert`)

INPUT `$gesfhr (=6)` - last digit of the input/output fort file, i.e. fort.2**6**
`hdas_atcfunix (fort.12)` - storm track
`tcvitals.as (fort.11)` - storm center obs
`wrf_env (fort.26)` - environmental flow (from `hwrf_split1.exe`)
`storm_pert (fort.71)` - separated 3D vortex field (from `hwrf_split1.exe`)

OUTPUT `storm_pert_new (fort.58)` - adjusted storm perturbation
`storm_size_p (fort.14)` - storm size information
`storm_sym (fort.23)` - storm symmetry information

USAGE **`echo $gesfhr $BASIN | hwrf_pert_ct1.exe`**

Stage2

1. Copy the fix files and namelist.
2. Run `diffwrf_3dvar.exe` to convert `wrfinput_d0[1-3]` and `wrfghost_d0[2-3]` to binary files `new_gfs_d0[1-3]` and `new_ght_d0[2-3]`, respectively.
3. Run `hwrf_create_nest_1x_10m.exe` to rebalance the inner nest domain data. This will generate the data file `new_data_d01` that contains the rebalanced outer and inner domain data.
4. Run `hwrf_create_trak_fnl.exe` to create `trak.fnl.all_gfs`, a guess track file from atcfunix.
5. Run `hwrf_merge_nest_4x_step12_3n.exe` to merge all three HWRF domains (`new_gfs_d0[1-3]`) onto the 3X domain. This will generate the file containing the merged data on the 3X domain (`data_4x_gfs`) and a file containing sea mask and roughness length data (`roughness2`).
6. Run `hwrf_split1.exe` to separate the `data_4x_gfs` into environment data (`gfs_env`) and storm vortex (`storm_pert_gfs`). A file containing the storm radius information will be generated, too (`storm_radius_gfs`).

Status Check:

In the standard output file, the line `"CRITICAL: Stage 2 completed"` should exist.

Output files:

`gfs_env`	environment fields from GFS data
`roughness2`	sea mask and roughness length from GFS data
`storm_pert_gfs`	storm vortex from GFS data
`storm_radius_gfs`	storm radius information from GFS data

Executables

`diffwrf_3dvar.exe`
Refer to Stage 1 in Section 5.2.2.

`hwrf_create_nest_1x_10m.exe`

FUNCTION Rebalances inner nest data

INPUT `$gesfhr (=6)` - last digit of the input/output fort file, i.e. fort.2**6**
`new_gfs_d02 (fort.46)`
`new_gfs_d01 (fort.26)`

OUTPUT `new_data_d01 (fort.57)` - outer domain data interpolated to inner domain

USAGE **`echo $gesfhr $BASIN | hwrf_create_nest_1x_10m.exe`**

`hwrf_create_trak_fnl.exe`
Refer to Stage 1 in Section 5.2.2.

`hwrf_merge_nest_4x_step12_3n.exe`
Refer to Stage 1 in Section 5.2.2.

`hwrf_split1.exe`
Refer to Stage 1 in Section 5.2.2.

Stage3

For a cold start or cycled start of a weak storm: The vortex and environment are obtained from the global data.

1. Link the input and fixed files.
2. Run `hwrf_pert_ct1.exe` to adjust the global vortex (`storm_pert_gfs` from Stage 2).

3. Run `hwrf_anl_4x_step2.exe` to adjust the storm vortex (`storm_pert_gfs1`) and add the new storm vortex to the environment flow (`gfs_env`) on the 3X domain grid. This will produce a new file (`new_data_4x`) containing the combined environment flow and the adjusted storm vortex.
4. When the combined vortex and environment flow is weaker than observations, discard the new file (`new_data_4x`), and run `hwrf_anl_cs_10m.exe` to further adjust the analysis. This produces a new version of `new_data_4x` containing the combined environment flow and adjusted vortex.
5. Run `hwrf_inter_4to6.exe` to interpolate the `new_data_4x` from the 3X domain onto the 27-km HWRF grid. This will produce the new `data_merge_d01`. Input file for storm radius is `storm_radius_gfs`.
6. Run `hwrf_inter_4to2.exe` to interpolate the `new_data_4x` from the 3X domain onto the ghost_d03 grid. This will produce the new `data_merge_g03`.
7. Run `hwrf_inter_2to2.exe` to interpolate the `new_data_4x` from the 3X domain onto the ghost_d02 grid. This will produce the new `data_merge_g02`.
8. Run `diffwrf_3dvar.exe` to convert the merged data files (`data_merge_d01` and `data_merge_g0[2-3]`) to NetCDF files (`wrfinput_d01` and `wrfghost_d0[2-3]`).

For a cycled start of a strong storm: Performs all steps from cold/cycled weak storm except for Step 2. For a strong cycled storm, `hwrf_pert_ct1.exe` runs in Stage 1 and the vortex is taken from the previous HWRF forecast.

For a cold start of a strong storm: The vortex is an adjusted bogus vortex.

1. Link the input and fixed files.
2. Run `hwrf_anl_bogus_10m.exe` to create a bogus vortex and add it to the environment.
3. Perform Steps 5-8 of the weak storm procedure.

Status Check:

The line `"CRITICAL: Stage 3 completed"` exists in the standard output.

Executables

`hwrf_pert_ct1.exe`
Refer to Stage 1 in Section 5.2.2.

`hwrf_anl_4x_step2.exe`

FUNCTION Adjusts the storm vortex and adds the new storm vortex to the environment flow on the 3X domain grid

INPUT `$gesfhr (=6)` – last digit of the input/output fort file, i.e. fort.2**6**
`storm_size_p (fort.14)` – from hwrf_pert_ctl.exe
`tcvitals.as (fort.11)` – storm center obs
`hdas_atcfunix (fort.12)` – input track file from previous 6-h forecast
`storm_sym (fort.23)` – symmetric part of storm
`gfs_env (fort.26)` – GFS environmental flow
`roughness1 (fort.46)` – roughness from `merge_nest_4x_step2.exe`
`storm_pert_new (fort.71)` – adjusted storm perturbation from `hwrf_pert_ct1.exe`

OUTPUT `wrf_env_new (fort.36)` – new environmental flow
`new_data_4x (fort.56)` – adjusted vortex + environment on 3X domain

USAGE **`echo $gesfhr $BASIN 0 1 | hwrf_anl_4x_step2.exe`**

`hwrf_anl_cs_10m.exe`

FUNCTION Further adjusts the storm vortex when combined vortex + environmental flow is less than the observed maximum wind speed

INPUT `$gesfhr (=6)` – last digit of the input/output fort file, i.e. fort.2**6**
`tcvitals.as (fort.11)` – observed storm center
`wrf_env_new (fort.26)` – new environmental flow (from `hwrf_anl_4x_step2.exe`)
`storm_sym (fort.23)` – symmetric part of storm (from `hwrf_pert_ct1.exe`)
`roughness (fort.46)` – roughness info for boundary layer calculation (from `hwrf_merge_nest_4x_step2.exe`)
`storm_radius (fort.85)` – from `wrf_split.exe`
`hwrf_storm_cyn_axisy_47 (fort.71,72,73,74,75,78)` input static vortex data
`hwrf_storm_20 (fort.76, 77)` – input static vortex data

OUTPUT `new_data_4x (fort.56)` – adjusted field on 3X domain when combined vortex + environmental flow is less than the observed maximum wind speed (replaces previous file)

USAGE **`echo $gesfhr $BASIN | hwrf_anl_cs_10m.exe`**

`hwrf_inter_4to6.exe`

FUNCTION Interpolates from 3X domain onto outer domain

INPUT
- `$gesfhr (=6)` – last digit of the input/output fort file, i.e. fort.2**6**
- `tcvitals.as (fort.11)` – observed storm center
- `new_gfs_d01 (fort.26)` – outer domain adjusted GFS data
- `new_data_4x (fort.36)` – adjusted storm
- `new_gfs_d01 (fort.46)` – outer domain adjusted GFS data
- `storm_radius (fort.85)`

OUTPUT `data_merge_d01 (fort.56)` – merged data on outer domain

USAGE **`echo $gesfhr $BASIN | hwrf_inter_4to6.exe`**

`hwrf_inter_4to2.exe`

FUNCTION Interpolates from 3X domain to ghost_d03

INPUT
- `$gesfhr (=6)` – last digit of the input/output fort file, i.e. fort.2**6**
- `tcvitals.as (fort.11)` – observed storm center
- `new_data_4x (fort.26)` – adjusted vortex + environment
- `new_ght_d03 (fort.36)` – input ghost file in binary format

OUTPUT `data_merge_g03 (fort.56)` – merged data on outer domain

USAGE **`echo $gesfhr $BASIN | hwrf_inter_4to2.exe`**

`hwrf_inter_2to2.exe`

FUNCTION Interpolates from 3X domain to ghost_d02

INPUT
- `$gesfhr (=6)` – last digit of the input/output fort file, i.e. fort.2**6**
- `tcvitals.as (fort.11)` – observed storm center
- `new_data_4x (fort.26)` – adjusted vortex + environment
- `new_ght_d02 (fort.36)` – input ghost file in binary format
- `new_gfs_d01 (fort.46)` – outer domain adjusted GFS data

OUTPUT `data_merge_g02 (fort.56)` – merged data on outer domain

USAGE **`echo $gesfhr $BASIN | hwrf_inter_2to2.exe`**

`diffwrf_3dvar.exe`

Refer to Stage 1 in Section 5.2.2.

`hwrf_anl_bogus_10m.exe`

FUNCTION Creates a bogus storm and adds it to the environmental flow

INPUT
- `$gesfhr (=6)` – last digit of the input/output fort file, i.e. fort.2**6**
- `tcvitals.as (fort.11)` – observed storm center
- `gfs_env (fort.26)` – GFS environmental flow
- `data_4x_gfs (fort.36)` – merged GFS inner/outer domain data
- `roughness2 (fort.46)` – roughness info for boundary layer calculation
- `storm_pert_gfs (fort.61)` – separated GFS 3D vortex field
- `storm_radius_gfs (fort.85)`
- `hwrf_storm_cyn_axisy_47 (fort.71,72,73,74,75,78)` input static vortex data
- `hwrf_storm_20 (fort.76, 77)` input static vortex data

OUTPUT `new_data_4x`: combined environment flow and bogus field on the 3X domain

USAGE **`echo $gesfhr $BASIN | hwrf_anl_bogus_10m.exe`**

6

Data Assimilation

6.1 Introduction

The preliminary initial conditions created by downscaling the global model data and performing the vortex relocation procedures are further modified with data assimilation using GSI on the 9- and 3-km WRF Ghost domains. No data assimilation is done in the 27-km parent domain. The term HDAS refers to the process of running GSI for data assimilation in HWRF.

The data assimilation in HWRF is performed using the hybrid ensemble-variational method. This indicates that the background error covariance information is a combination of two sources, a static, pre-generated matrix for the global model, and a flow-dependent matrix derived from the GFS ensemble 6-h forecasts. Because HWRF uses the GFS ensemble, but does not feedback into it, this procedure is termed "one-way hybrid". For more information on the ensemble-variational method, refer to the HWRF v3.6a Scientific Documentation available from the DTC website (`www.dtcenter.org/HurrWRF/users`).

The datasets assimilated in operations in the 9-km (d02) and 3-km (d03) domains are described in the HWRF Scientific Documentation. HWRF has the capability of assimilating tropical cyclone inner-core data such as the NOAA's P3 Tail Doppler Radar (TDR) observation. To collect inner-core observations, an aircraft has to penetrate the target TC multiple times to finish one mission, which may take several hours; therefore the observations in one TDR data set are collected at different times. In order for GSI to calculate the innovation, defined as the difference between the first guess and the analysis, it needs to have the first guess and the observations valid at the same time. To accomplish this for observations that span a range of times, the First Guess at Appropriate Time (FGAT) procedure is used. In FGAT, first-guess fields valid at various times are supplied to GSI,

which then interpolates the data to the time in which each observation was taken. For HWRF, first-guess fields are created at three time levels: 3 h before the HWRF initial time (Figure 4.2); at the HWRF initial time (Figure 4.3); and 3 h after the HWRF initial time (Figure 4.4). In order to create the three first-guesses, the real_nmm, short WRF forecasts, and vortex adjustment procedures are performed three times. This produces the three ghost d03 and three ghost d02 output files that are used by GSI in its FGAT operation (Figure 6.1) After the data is assimilated in the ghost d02 and ghost d03 domains, the preliminary analysis for the parent domain, the middle and inner domain output from the WRF Analysis, and the ghost d02 and ghost d03 GSI analysis (which used FGAT) are merged to produce the final atmospheric IC for the 5-day forecast. In order to perform the data assimilation in the ghost domain, users should run GSI and then merge. For more details about GSI, please consult the GSI Users' Guide available from the DTC at `http://www.dtcenter.org/com-GSI/users/docs/users_guide/GSIUserGuide_v3.3.pdf`.

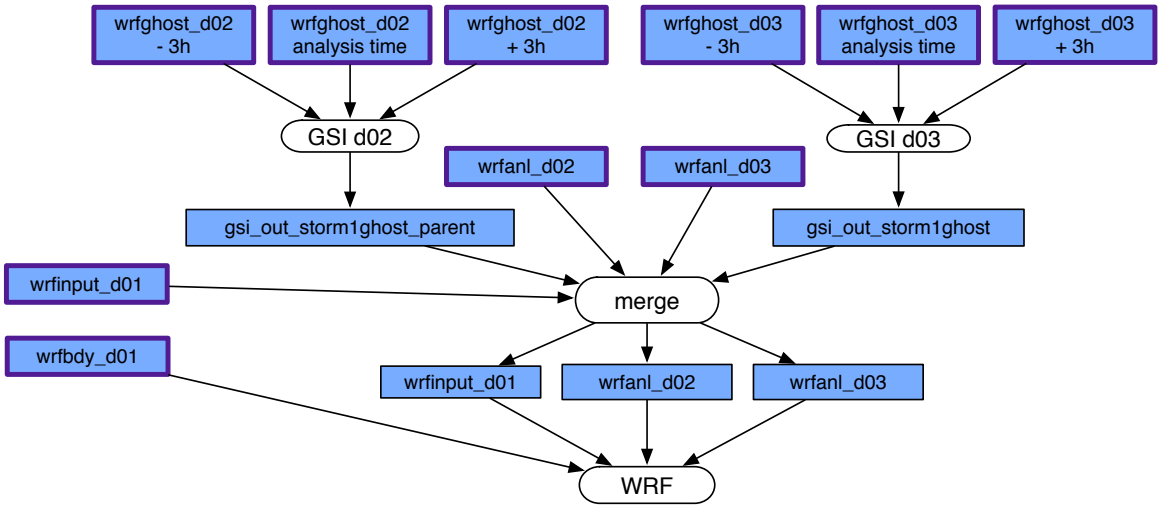

Figure 6.1: Simplified GSI and Merge procedures. Purple outlined boxes correspond to the purple outlined boxes of the figures in Section 4.2. Blue boxes are netCDF files.

6.2 Scripts

The HWRF data assimilation component is run using two wrapper scripts, `gsi_d02_wrapper` and `gsi_d03_wrapper`. These wrappers are responsible for calling their respective instances of `scripts/exhwrf_gsi.py`.

6.2.1 Overview of `exhwrf_gsi.py`

1. Initialize the objects used to run all components of HWRF by calling `hwrf_expt.init_module()`.
2. Run GSI for the appropriate domain.

6.2.2 Overview of the GSI Module

The run module for GSI (an FGATGSI class object) is located in `ush/hwrf/gsi.py`, and is responsible for the following tasks.

1. Link the fixed files
2. Link the GFS ensemble files
3. Link the observation files
4. Link the bias correction files
5. Create a namelist for the GSI analysis
6. Copy the background file (`wrfinout`) from the corresponding WRF Ghost run
7. Run `gsi.exe`

Output files:

`stdout`	The standard text output file. It is the file most often used to check the GSI analysis processes as it contains basic and important information about the analyses.
`wrf_inout`	Analysis results; format is same as the input background file

Status Check:

In the standard output file, you will find the line `"CRITICAL: GSI succeeded"` followed by the domain for which the assimilation was run.

Executables:

`gsi.exe`

FUNCTION Performs the GSI 3D-VAR data assimilation analysis

INPUT `gsiparm.anl` – GSI namelist, created by the script by modifying template `/parm/hwrf_gsi.nml`
`filelist` – ASCII file with 80 lines, each one containing a file name for a GFS ensemble member (used for ensemble-based background covariance)
`satbias_angle` – file containing information on satellite angle, from dataset directory
`satbias_in` – file containing information on satellite bias, from dataset directory
`wrf_inout` – background file, copied from WRF Ghost output
Various observations in BUFR and prepBUFR format

OUTPUT `wrf_inout` – analysis results if GSI completes successfully. The format is the same as the background file.

USAGE **`gsi.exe < gsiparm.anl`**

7

Merge

7.1 Introduction

Once the HWRF atmospheric initialization has been completed with the use of the vortex relocation and data assimilation, the adjusted ICs on all grids must be merged to provide the final ICs for the HWRF 5-day forecast. The origin of the files going into the merge procedure is shown in Figure 6.1, and is run by the wrapper script `merge_wrapper`. A description of the domains used in HWRF is included in Section 4.1.

7.2 Scripts

Merge is run by submitting the `merge_wrapper`, which sets necessary environment variables before running Python script `exhwrf_merge.py`.

7.2.1 Overview of `exhwrf_merge.py`

1. Initialize the objects used to run all components of HWRF by calling `hwrf_expt.init_module()`.
2. Run the `gdas_merge` Python module.

7.2.2 Overview of Merge Module

Merge is a RelocationTask object whose run module lives in `ush/hwrf/relocate.py` and is responsible for the following tasks.

1. Copy the input files.
2. Check to see whether `storm_radius` file exists from relocate process and contains information.
3. Run `diffwrf_3dvar.exe` to convert the netCDF format `wrfinput_d0[1-3]` and `wrfghost_d0[2-3]` to unformatted data files `new_hdas_d01`, `new_gfs_d0[2-3]`, and `new_ght_d0[2-3]`.
4. Run `hwrf_inter_2to1.exe` to interpolate the data in file `new_ght_d03` and `new_gfs_d03` to the inner nest domain grid. This will produce the merged data on the inner nest grid (`data_merge_d03`).
5. Run `hwrf_inter_2to1.exe` to interpolate the data in file `new_ght_d02` and `new_gfs_d02` to the inner nest domain grid. This will produce the merged data on the inner nest grid (`data_merge_d02`).
6. Run `hwrf_inter_2to6.exe` to interpolate the files `new_hdas_d01`, `new_gfs_d02`, and `new_ght_d02` to the outer domain grid. This will produce the merged data on the outer domain grid (`data_merge_d01`).
7. Run `diffwrf_3dvar.exe` to convert the unformated files `data_merge_d0[1-3]` to the netCDF format files `wrfinput_d0[1-3]`.
8. Deliver the products.

Output files:

`wrfinput_d01`	IC for the outer domain containing the new vortex
`wrfinput_d02`	IC for the middle nest domain containing the new vortex
`wrfinput_d03`	IC for the inner nest domain containing the new vortex

Status Check:

Check that output files exist in the `com/` directory for the current cycle. The line `"CRITICAL: Merge running in directory:"` will also appear near the end of the standard output file.

Executables:

`diffwrf_3dvar.exe`
Refer to Stage 1 in Section 5.2.2.

`hwrf_inter_2to1.exe`

FUNCTION Interpolates from ghost domains to nest domains (*DOMAIN* is "02" or "03")

INPUT `$gesfhr (=6)` - last digit of the input/output fort file, i.e. fort.2**6**
`new_ght_d{DOMAIN} (fort.26)` - data on ghost domain
`new_gfs_d{DOMAIN} (fort.36)` - data on inner nest domain

OUTPUT `data_merge_d{DOMAIN} (fort.56)` - interpolated data on inner domain

USAGE **`echo $gesfhr $BASIN | hwrf_inter_2to1.exe`**

`hwrf_inter_2to6.exe`

FUNCTION Interpolates data from ghost domain to outer domain.

INPUT `$gesfhr (=6)` - last digit of the input/output fort file, i.e. fort.2**6**
`new_gfs_d02 (fort.26)` - data on HWRF middle nest grid
`new_ght_d02 (fort.36)` - data on ghost d03 grid
`new_hdas_d01 (fort.46)` - data on outer domain grid
`storm_radius (fort.85)` - storm radius obtained from wrf_split1.exe in either Stage 1 (cycled run) or Stage 2 (cold start)

OUTPUT `data_merge_d01 (fort.56)` - interpolated data on outer domain

USAGE **`echo $gesfhr $BASIN | hwrf_inter_2to6.exe`**

8

Ocean Initialization for MPIPOM-TC

8.1 Introduction

This chapter explains how to run the initialization of the MPIPOM-TC component of the HWRF model, available from the DTC. Users are also encouraged to read the HWRF v3.6a Scientific Documentation.

8.2 Scripts

The initialization of the HWRF ocean model, MPIPOM-TC, is accomplished by running the `init_ocean_wrapper`, which is responsible for linking the ocean executables to the `exec/` directory and running `exhwrf_ocean_init.py` to generate updated initial conditions for the ocean forecast component of HWRF.

8.2.1 Overview of `exhwrf_ocean_init.py`

1. Initialize the objects used to run all components of HWRF by calling hwrf_expt.init_module().

2. Run pominit. If successful, write a status file to indicate that the forecast will be coupled, otherwise indicate that the forecast will be uncoupled.

8.2.2 Overview of Ocean Init Modules

1. Determine the region for which the ocean model will be run. Currently supported options are the transatlantic domain for storms in the North Atlantic Ocean (L) and the East Pacific domain for storms in the Eastern North Pacific Ocean (E). Ocean coupling is not currently supported in other ocean basins, although options exist for MPIPOM-TC coupling worldwide.
2. Determine the ocean initial condition module to be used. Currently supported options, which are consistent with the 2014 operational configuration of HWRF, include the Generalized Digital Environmental Model (GDEM) temperature and salinity climatology with feature-based modifications for the transatlantic domain and unmodified GDEMv3 climatology for the East Pacific domain.
3. Link the input and fix files.
4. Run `gfdl_getsst.exe` to obtain the sea surface temperature and land/sea mask from the GFS analysis.
5. Run `gfdl_sharp_mcs_rf_l2m_rmy5.exe` (in transatlantic domain only) to assimilate ocean features, including major fronts and eddies, and sharpen the frontal gradients.
6. Run `transatl06prep.xc` (in transatlantic domain only) to blend the sharpened GDEM and the unsharpened GDEM along 50 W longitude.
7. Prepare ocean initial conditions for MPIPOM-TC Phase 1. By default, `pomprep_fbtr.xc` is used for the transatlantic domain, and `pomprep_gdm3.xc` is used for the East Pacific domain. Also, by default, both executables are set to assimilate the GFS SST analysis into the upper ocean mixed layer, creating an ocean initial condition at the sea surface that is identical to the atmospheric initial condition at the sea surface.
8. Run `hwrf_ocean_init.exe` for Phase 1 to spin up the ocean currents. The SST is held constant during Phase 1. Historically, Phase 1 has also been known as Phase 3, so the terms *Phase 1* and *Phase 3* are sometimes used interchangeably.
9. Run `hwrf_ocean_init.exe` for Phase 2 to generate the cold wake at the sea surface prior to the start of the coupled model forecast. Historically, Phase 2 has also been known as Phase 4, so the terms *Phase 2* and *Phase 4* are sometimes used interchangeably.

Executables

`gfdl_getsst.exe`

FUNCTION Extract SST, land/sea mask, and lon/lat data from the GFS spectral files.

INPUT `for11 (gfs.`*YYYYMMDDHH*`.t`*HH*`z.sfcanl)`
`fort.11 (gfs.`*YYYYMMDDHH*`.t`*HH*`z.sfcanl)`
`fort.12 (gfs.`*YYYYMMDDHH*`.t`*HH*`z.sanl)`

OUTPUT `fort.23 (lonlat.gfs)`
`fort.74 (sst.gfs.dat)`
`fort.77 (mask.gfs.dat)`
`getsst.out`

USAGE **`gfdl_getsst.exe > getsst.out`**

`gfdl_sharp_mcs_rf_l2m_rmy5.exe`

FUNCTION Run the feature-based sharpening program, which takes the GDEM T/S climatology, horizontally interpolates it onto the old POM-TC grid for the United domain, and employs the diagnostic, feature-based modeling procedure, as described in the HWRF Scientific Documentation. This executable is called for the transatlantic domain only.

INPUT `input_sharp`
`fort.66 (gfdl_ocean_topo_and_mask.`*`REGION`*`)`
`fort.8 (gfdl_gdem.`*`MM`*`.ascii)`
`fort.90 (gfdl_gdem.`*`MM±1`*`.ascii)`
`fort.24 (gfdl_ocean_readu.dat.`*`MM`*`)`
`fort.82 (gfdl_ocean_spinup_gdem3.dat.`*`MM`*`)`
`fort.50 (gfdl_ocean_spinup_gspath.`*`MM`*`)`
`fort.55 (gfdl_ocean_spinup.BAYuf)`
`fort.65 (gfdl_ocean_spinup.FSgsuf)`
`fort.75 (gfdl_ocean_spinup.SGYREuf)`
`fort.91 (mmdd.dat)`
`fort.31 (hwrf_gfdl_loop_current_rmy5.dat.`*`YYYYMMDD`*`)`
`fort.32 (hwrf_gfdl_loop_current_wc_ring_rmy5.dat.`*`YYYYMMDD`*`)`

OUTPUT `fort.13 (gfdl_initdata.united.`*`MM`*`)`
`sharp_mcs_r_l2b.out`

USAGE **`gfdl_sharp_mcs_rf_l2m_rmy5.exe < input_sharp > \`**
`sharp_mcs_r_l2b.out`

`transatl06prep.xc`

FUNCTION Blend T/S between sharpened GDEM and unsharpened GDEM along 50W. This executable is called for the transatlantic domain only.

INPUT `fort.8 (gfdl_gdem.`*`MM`*`.ascii)`
`fort.90 (gfdl_gdem.`*`MM±1`*`.ascii)`
`fort.91 (`*`mmdd`*`.dat)`
`fort.13 (gfdl_initdata.united.`*`MM`*`)`

OUTPUT `fort.113 (gfdl_initdata.`*`REGION`*`.`*`MM`*`)`
`transatl06prep.out`

USAGE **`transatl06prep.xc > transatl06prep.out`**

`pomprep_fbtr.xc`

FUNCTION Read sharpened and blended GDEM climatology, horizontally interpolate it onto the high resolution MPIPOM-TC grid, incorporate the bathymetry and a land/sea mask, assimilate the GFS SST, and prepare the ICs for MPIPOM-TC. This executable is called for the transatlantic domain only.

INPUT `input`
`fort.13 (gfdl_initdata.transatl.`*`MM`*`)`
`fort.66 (gfdl_ocean_topo_and_mask.`*`REGION`*`.lores)`
`fort.21 (sst.gfs.dat)`
`fort.22 (mask.gfs.dat)`
`fort.23 (lonlat.gfs)`

OUTPUT *`STORM`*`.el_initial.nc`
`STORM``.grid.nc`
`STORM``.ts_clim.nc`
`STORM``.ts_initial.nc`
`STORM``.uv_initial.nc`
`ocean_pomprep.out`

USAGE **`pomprep_fbtr.xc < input > ocean_pomprep.out`**

`pomprep_gdm3.xc`

FUNCTION Read GDEMv3 climatology, horizontally interpolate it onto the high resolution MPIPOM-TC grid, incorporate the bathymetry and a land/sea mask, assimilate the GFS SST, and prepare the ICs for MPIPOM-TC. This executable is currently supported for the East Pacific domain only, but it can be used worldwide.

INPUT `input`
`tin.nc (tgdemv3s`*`MM`*`.nc)`
`sin.nc (sgdemv3s`*`MM`*`.nc)`
`fort.66 (gfdl_ocean_topo_and_mask.`*`REGION`*`.lores)`
`fort.21 (sst.gfs.dat)`
`fort.22 (mask.gfs.dat)`
`fort.23 (lonlat.gfs)`

OUTPUT *STORM*.el_initial.nc
STORM.grid.nc
STORM.ts_clim.nc
STORM.ts_initial.nc
STORM.uv_initial.nc
ocean_pomprep.out

USAGE **pomprep_gdm3.xc < input > ocean_pomprep.out**

hwrf_ocean_init.exe

FUNCTION Run MPIPOM-TC ocean Phase 1 or Phase 2 (also known historically as ocean Phase 3 and Phase 4, respectively, as in the model code).

INPUT For Phase 1
pom.nml
STORM.el_initial.nc
STORM.grid.nc
STORM.ts_clim.nc
STORM.ts_initial.nc
STORM.uv_initial.nc
For Phase 2
pom.nml
STORM.el_initial.nc
STORM.grid.nc
STORM.ts_clim.nc
STORM.ts_initial.nc
STORM.uv_initial.nc
restart.phase1.nc

OUTPUT For Phase 1
restart.phase1.nc
STORM.0000.nc
STORM.0001.nc
STORM.0002.nc
For Phase 2
restart.phase2.nc
STORM.0000.nc
STORM.0001.nc
STORM.0002.nc
STORM.0003.nc

USAGE **hwrf_ocean_init.exe > ocean_init.out**

9

Forecast Model

9.1 Introduction

The operational HWRF, which runs in the North Atlantic and Eastern North Pacific basins, is an atmosphere-ocean coupled forecast system, which includes an atmospheric component (WRF-NMM), an ocean component (MPIPOM-TC), and the NCEP Coupler. Therefore, HWRF is a Multiple Program Multiple Data (MPMD) system which consists of three executables, WRF, MPIPOM-TC, and Coupler. After the ocean and atmosphere initializations are successfully completed, the coupled HWRF system run can be submitted. In the non-operational basins, Central Pacific, West Pacific, and Indian Ocean, HWRF can only be run in atmosphere standalone mode, that is, uncoupled.

9.2 Scripts

The wrapper script `forecast_wrapper` is responsible for calling the Python script `exhwrf_forecast.py` in the `scripts/` directory. The wrapper script sets the number of tasks for the parallel forecast job. In operations, 210 tasks are used: 200 for the WRF forecast, 9 for the MPIPOM-TC, and 1 for the NCEP Coupler. While this configuration is recommended, you may change the total number of tasks to reflect the following relationship,

$$TOTAL\_TASKS = np + 9 + 1, \tag{9.1}$$

where np is an integer multiple of 4. The number of processors used should match `TO-`

TAL_TASKS.

For uncoupled runs, you should change the variable TOTAL_TASKS in forecast_wrapper to reflect the reduction of tasks (i.e., subtract 10 for the MPIPOM-TC and Coupler). In this case, you should also change the MODEL variable to "UNCOUPLED" so that the Python script will run only wrf.exe.

9.2.1 Overview of exhwrf_forecast.py

1. Initialize all of the objects used to run HWRF
2. Run the HWRF main forecast, coupled or uncoupled (runwrf.run)

9.2.2 Overview of the Forecast Module

For coupled forecasts, runwrf is an object of the WRFCoupledPOM subclass of fcst-task.WRFAtmos. The run module is responsible for the following tasks.

1. Link the input files required by WRF (fix files, initial and boundary condition files, and geographical data files).
2. Make the Coupler namelist.
3. Make the POM namelist.
4. Copy POM inputs.
5. Run hwrf_swcorner_dynamic.exe to calculate the location of the middle nest.
6. Make the WRF namelist.
7. Submit the MPI forecast run (three executables: wrf.exe, hwrf_ocean_fcst.exe, hwrf_wm3c.exe).

Output files:

Several types of primary output files containing most variables, output every hour for the first nine hours, then every three hours.

wrfout_d01_*yyyy-mm-dd_hh:mm:ss*
wrfout_d02_*yyyy-mm-dd_hh:mm:ss*
wrfout_d03_*yyyy-mm-dd_hh:mm:ss*

Auxiliary output files containing accumulated precipitation and 10-m winds, with hourly output in a single file for each domain.

wrfdiag_d01
wrfdiag_d02
wrfdiag_d03

Text file with time series of storm properties.

`hifreq_d03.htcf`

File `hifreq_d03.htcf` has nine columns containing the following items.

1. Forecast lead time (s)
2. Minimum Sea Level Pressure (MSLP) in the inner nest (hPa)
3. Latitude of grid point with minimum sea level pressure
4. Longitude of grid point with minimum sea level pressure
5. Maximum wind in the inner nest at the lowest model level (kt)
6. Latitude of grid point with the maximum wind
7. Longitude of grid point with the maximum wind
8. Latitude of the location of the center of the inner nest
9. Longitude of the location of the center of the inner nest

The ocean model will produce diagnostic output files with the following naming convention.

STORMNAME.00*NN*.nc	Ocean forecast output numbered consecutively with integers (frequency depends on namelist variables) in netCDF format
flux.00*DD*	Forecast ocean flux for each day
sst.00*DD*	Forecast sea surface temperature for each day

Status Check:

To check whether the run was successful, look for `"SUCCESS COMPLETE" at the end of the log file (e.g., rsl.out.0000)`. This check is also done in the code, and can be found in the standard output file.

Executables

`hwrf_swcorner_dynamic.exe`
Refer to Section 4.2.3.

`wrf.exe`

FUNCTION Atmospheric component of HWRF

INPUT
`geo_nmm.d01.nc` – Geogrid static files for d01
`geo_nmm_nest.l01.nc` – Geogrid static files for d02
`geo_nmm_nest.l02.nc` – Geogrid static files for d03
`wrfbdy_d01` – LBCs for d01
`wrfinput_d01` – ICs for d01
`wrfanl_d02_`*YYYY-MM-DD_HH*`:00:00` – ICs for d02
`wrfanl_d03_`*YYYY-MM-DD_HH*`:00:00` – ICs for d03
`gwd_surface` – Gravity wave drag file
`namelist.input` – Example in Appendix B
`fort.65`
WRF Fix files (Refer to Section 4.2.3)

OUTPUT
`wrfout_d01_`*YYYY-MM-DD_HH*`:00:00`
`wrfout_d02_`*YYYY-MM-DD_HH*`:00:00`
`wrfout_d03_`*YYYY-MM-DD_HH*`:00:00`
`wrfdiag_d01`
`wrfdiag_d02`
`wrfdiag_d03`
`hifreq_d03.htcf`

USAGE
For a coupled HWRF forecast, `wrf.exe` must be submitted with the coupler and the ocean model. Refer to MPI Explanation below.
For an uncoupled run, you only need to issue the executable.
`wrf.exe`

`hwrf_ocean_fcst.exe`

FUNCTION MPIPOM-TC ocean model for HWRF

INPUT
STORM`.el_initial.nc`
STORM`.grid.nc`
STORM`.ts_clim.nc`
STORM`.ts_initial.nc`
STORM`.uv_initial.nc`
`restart.phase2.nc`

OUTPUT
STORM`.0000.nc`
STORM`.0001.nc`
STORM`.0002.nc`
STORM`.0003.nc`
STORM`.0004.nc`
STORM`.0005.nc`

USAGE
For a coupled HWRF forecast, the ocean model `hwrf_ocean_fcst.exe` must be submitted to the computers with the atmosphere model `wrf.exe` and the coupler `hwrf_wm3c.exe`. Refer to the MPI Explanation below.

`hwrf_wm3c.exe`

FUNCTION Coupler that links the atmospheric component and oceanic component

INPUT `cpl.nml` - coupler namelist

OUTPUT None

USAGE Refer to the MPI Explanation below

Explanation of the MPI command for the forecast model

As mentioned in Section 6.1, HWRF can be run as either a coupled or uncoupled model of the atmosphere and ocean. The operational HWRF runs coupled in the North Atlantic and Eastern North Pacific basins. The scripting system submits coupled runs in the North Atlantic and Eastern North Pacific basins when `run_ocean=yes` in `hwrf_basic.conf`, and uncoupled runs in other basins (`run_ocean=no`). If an uncoupled run is desired, file `parm/hwrf_basic.conf` needs to be manually altered as described in Sections 3.2.2 and 3.6.2.

- Coupled

  With LSF, using the command `mpirun.lsf`

  `mpirun.lsf -cmdfile` *`cmdfile`*

  where *`cmdfile`* is a file containing the list of executables. For example, the cmdfile file below indicates that the coupled run will be submitted to 210 processors, one for the coupler (`hwrf_wm3c.exe`), nine for the ocean domain (`hwrf_ocean_fcst.exe`) and 200 for `wrf.exe`.

  ```
  hwrf_wm3c.exe
  hwrf_ocean_fcst.exe
  wrf.exe
  wrf.exe
  wrf.exe
  wrf.exe...
  ```

  With MOAB/Torque, using the command `mpiexec`

  ```
  mpiexec -np 1 ./hwrf_wm3c.exe :  -np 9 \
  ./hwrf_ocean_fcst.exe :  -np 200 ./wrf.exe
  ```

  For example, the previous command will run the coupled model using 210 processors, one for the coupler (`hwrf_wm3c.exe`), nine for the ocean domain (`hwrf_ocean_fcst.exe`), and 200 for `wrf.exe`

- Uncoupled
  - With LSF, using the command `mpirun.lsf`

    `mpirun.lsf -procs 200 ${WRF_ROOT}/main/wrf.exe`
  - With MOAB/Torque, using the command `mpiexec`

    `mpiexec -np 200 ${WRF_ROOT}/main/wrf.exe`

10

HWRF Post Processor

10.1 Introduction

The NCEP UPP is used to de-stagger the HWRF parent and nest domain output, compute diagnostic variables, and interpolate the output from the native WRF grids to NWS standard levels (pressure, height etc.) and standard output grids (latitude/longitude, Lambert Conformal, polar-stereographic, Advanced Weather Interactive Processing System grids, etc.). The UPP outputs files in GRIB1 format. This package also merges the parent and nest domains forecasts onto one combined domain grid. Information on how to acquire and build the UPP code is available in Section 2.

There are two main executables in UPP, `unipost.exe` and `copygb.exe`. This chapter covers only the module that calls `unipost.exe`. The use of `copygb.exe` is covered in Chapter 11.

10.2 Scripts

The postprocessing using UPP is run using two wrappers, `unpost_wrapper` and `post_wrapper`. These wrappers call the `exhwrf_unpost.py` and `exhwrf_post.py`, respectively.

10.2.1 Overview of `exhwrf_unpost.py`

The purpose of this script is to delete output from any previous run of the same cycle.

1. Initialize the objects used to run all components of HWRF by calling `hwrf_expt.init_module()`.
2. Run the unrun modules for the following tasks.
   - runwrf
   - wrfcopier
   - satpost
   - nonsatpost
   - gribber

10.2.2 Overview of `exhwrf_post.py`

The Python script contains a loop that continually checks the status of the forecast and post-processes any output files that are available. As long as there are tasks remaining, it runs copies of `wrfcopier`, `nonsatpost`, and `satpost`. Note that `satpost` refers to the postprocessing to produce synthetic satellite brightness temperatures, while `nonsatpost` refer to postprocessing to produce all other variables (temperature, winds etc.)

10.2.3 Overview of UPP Python Modules

Wrfcopier

Wrfcopier is a WRFCopyTask class object that lives in `ush/hwrf/copywrf.py`. It serves the primary purpose of delivering files from the WRF run directory to the `com/` directory.

Nonsatpost and Satpost

Nonsatpost and satpost are PostManyWRF class objects that run the UPP on the WRF output files. They are used to produce general forecast products and synthetic satellite images, respectively. The run module for these tasks performs the following duties.

1. Link the input file (wrfout forecast or analysis file)
2. Make a control file that corresponds to the input file
3. Write `itag` file which contains the following four lines to be read by `unipost.exe`.
   - Name of the WRF output file to be post processed
   - Format of the WRF output (NetCDF or binary; choose NetCDF for HWRF)
   - Forecast valid time (not model start time) in WRF format
   - Model name (NMM or NCAR; choose NMM for HWRF)

4. Run `unipost.exe`

Output files:

In the `intercom/` directory, there are directories for each forecast hour containing the satpost and nonsatpost output. The following list describes the naming convention for these directories and files. The forecast hour, *hh*, by default is hourly for the first 9 hours, and 3 hourly after that.

```
satpost-fhh00m/
    satpost-fhhh00m-moad.egrb
    satpost-fhhh00m-storm1inner.egrb
    satpost-fhhh00m-storm1outer.egrb
nonsatpost-fhh00m/
    nonsatpost-fhhh00m-moad.egrb
    nonsatpost-fhhh00m-storm1inner.egrb
    nonsatpost-fhhh00m-storm1outer.egrb
```

Status check:

The string `"INFO: completed post"` will appear in the standard output file.

Executables:

`unipost.exe`

FUNCTION	De-staggers the HWRF native output, interpolates it vertically to pressure levels, computes derived variables, and outputs in GRIB format.
INPUT	`hwrf_eta_micro_lookup.dat` `wrfout_d01, wrfout_d02 or wrfout_d03` - HWRF native output `itag` - namelist unipost control file: for satpost: `hwrf_cntrl.sat`; and, for nonsatpost: `hwrf_cntrl.nonsat${BASIN}`, where *BASIN* can be L, E, C, or W for N. Atlantic, E. N. Pacific, Central N. Pacific, or W. N. Pacific, respectively.
OUTPUT	HWRF postprocessed output in GRIB format
USAGE	**`unipost.exe < itag`**

11

Forecast Products

11.1 Introduction

HWRF v3.6a will produce two types of forecast products, processed GRIB files (projected to lat-lon grids) and track files containing information about the tropical cyclone. The processed GRIB files are produced on several different grids outlined in Figure 11.1 for the general atmospheric fields and Figure 11.2 for satellite-derived products. Those GRIB files are used as input to the GFDL Vortex Tracker. They can also be used to create images with visualization packages such as GrADS, NCL, etc. Image generation is not covered in this Users' Guide.

The GFDL Vortex Tracker is a program that ingests model forecasts in GRIB format, objectively analyzes the data to provide an estimate of the vortex center position (latitude and longitude), and tracks the storm for the duration of the forecast. Additionally, it reports metrics of the forecast storm, such as intensity (maximum 10-m winds and MSLP) and structure (wind radii for 34-, 50-, and 64-knot thresholds in each quadrant of the storm) at each output time. The GFDL Vortex Tracker requires the forecast grids to be on a cylindrical equidistant, latitude-longitude (lat/lon) grid. For HWRF, UPP is used to process the raw model output and create the GRIB files for the tracker.

The vortex tracker creates two output files containing the vortex position, intensity, and structure information: one in Automated Tropical Cyclone Forecast (ATCF) format; and another in a modified ATCF format.

The GFDL Vortex Tracker locates the hurricane vortex center positions by searching for the average of the maximum or minimum of several parameters in the vicinity of an input first-guess position of the targeted vortex. The primary tracking parameters are relative vorticity

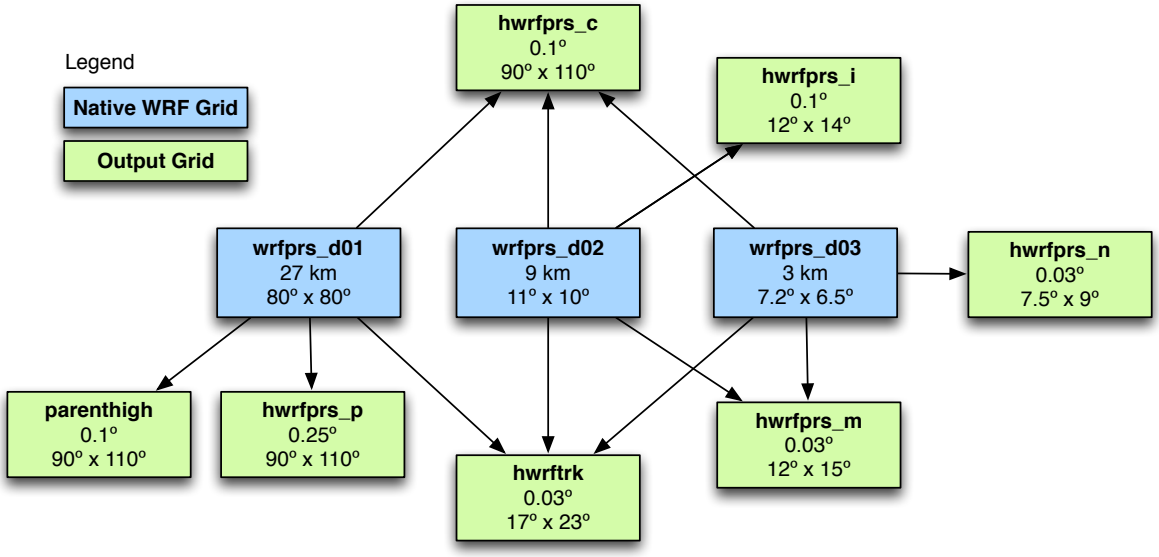

Figure 11.1: Naming convention, resolution, and size for the output grids that contain conventional atmospheric data. Blue boxes indicate the grids from the `wrfout` files. Green boxes are the grids in the final GRIB files.

at 850 hPa and 700 hPa, MSLP, and geopotential height at 850 and 700 hPa. Secondarily, wind speed at 10 m, 850 hPa, and 700 hPa are used. Winds at 500 hPa are used, together with other parameters, for advecting the storm and creating a first guess position for all times beyond initialization. Many parameters are used in order to provide more accurate position estimates for weaker storms, which often have poorly defined structures/centers.

Besides the forecast file in GRIB format, the vortex tracker also ingests a GRIB index file, which is generated by running the program grbindex. The utility wgrib is also used for preparing data for the tracker. Both grbindex and wgrib were developed by NCEP and are distributed by the DTC as part of the HWRF Utilities.

This version of the tracker contains added capabilities of tracking cyclogenesis and identifying cyclone thermodynamic phases. The identification of cyclone thermodynamic phases requires that the input data contain temperature every 50 hPa from 300 to 500 hPa (for the "vtt" scheme) or the geopotential height every 50 hPa from 300 to 900 hPa (for the "cps" scheme).

11.2 Scripts

The forecasts products are obtained by running the `products_wrapper`, which calls `scripts/exhwrf_products.py` after setting a few environment variables to re-direct the standard output and standard error files. These files can be placed anywhere by changing the environment variables `REGRIBBER_LOGS` and `TRACKER_LOGS` to the desired path.

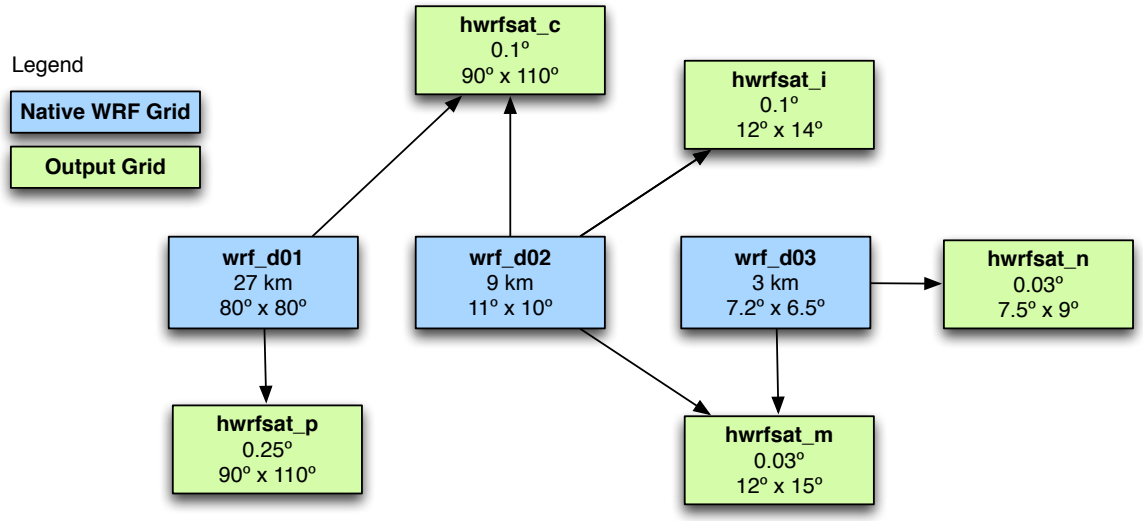

Figure 11.2: Naming convention, resolution, and size for the output grids that contain satellite-derived products. Blue boxes indicate the grids from the `wrfout` files. Green boxes are the grids in the final GRIB files.

The GFDL Vortex Tracker is driven by the wrapper script `tracker_wrapper`, which calls `scripts/exhwrf_track.py`. The Python script runs the tracker on the processed GRIB forecast files.

11.2.1 Overview of `exhwrf_products.py`

1. Initialize the objects used to run all components of HWRF by calling `hwrf_expt.init_module()`
2. Launch the four parallel tasks. The parent process launches `products`, while the subprocesses run copies of `gribber` and `tracker`. Each of these continually check the availability of files before running.
3. Deliver products to `com/` directory as the become available from the `products`, `gribber`, and `tracker` processes.

Files delivered to `com/` directory

The following examples are for Hurricane Sandy (2012), which was Atlantic Storm number 18. The string `"sandy18"` would be replaced by the name and number of the storm in the given experiment. *`SID`* is the storm ID (i.e., 18L for Sandy). The "l" or "L" following `"sandy18"` is a product of an operational naming convention requiring some files to have identical counterparts, but different capitalization. In this case, the letter "L" denotes that the files are being delivered for a storm in the Atlantic Basin. Capital *`YYYYMMDDHH`* denotes analysis time, while lower case *`yyyymmddhh`* denotes forecast time.

18l.*YYYYMMDDHH*.domain.center	ASCII file containing the coordinates of the domain center
18l.trak.hwrf.atcfunix.*YYYYMMDDHH*.combine	Forecast track
18l.wrfout_d[01-03]_*yyyy-mm-dd_hh*:00:00	WRF fcst output for the first 9 hrs in 3-hr increments
aal182012_HWRF_hPYHW_*YYYYMMDDHH*.dat	a-deck file for this forecast period
sandy18l.*YYYYMMDDHH*.afos	Storm position and heading/speed hourly for 9 hours, 3 hourly afterward for delivery to NHC
sandy18L.*YYYYMMDDHH*.afos	Same as above
sandy18l.*YYYYMMDDHH*.binary_d[01-03]	Binary forecasts on each domain
sandy18l.*YYYYMMDDHH*.fort.65	Land-sea mask for the coupler
sandy18l.*YYYYMMDDHH*.gsi_cvs[2-3].biascr	Satellite bias correction file
sandy18l.*YYYYMMDDHH*.hwrf_d03.htcf	Storm info from d03 every 5 mins
sandy18L.*YYYYMMDDHH*.hwrf_d03.htcf	Same as above
sandy18l.*YYYYMMDDHH*.hwrf_d03.htcfstats	Storm info from d03 every 3 hours
sandy18L.*YYYYMMDDHH*.hwrf_d03.htcfstats	Same as above
sandy18l.*YYYYMMDDHH*.hwrfprs_c. grb2f[00-126]	Non-satellite vars from all domains merged onto 0.1° d01-sized grid
sandy18l.*YYYYMMDDHH*.hwrfprs_i. grb2f[00-126]	Non-satellite vars from d02 on native d02 grid
sandy18l.*YYYYMMDDHH*.hwrfprs_m. grb2f[00-126]	Non-satellite vars from d02 & d03 merged onto 0.03° d02-sized grid
sandy18l.*YYYYMMDDHH*.hwrfprs_n. grb2f[00-126]	Non-satellite vars from d03 on native d03 grid
sandy18l.*YYYYMMDDHH*.hwrfprs_p. grb2f[00-126]	Non-satellite vars from all domains merged onto 0.25° d01-sized grid
sandy18l.*YYYYMMDDHH*.hwrfsat_c. grb2f[00-126]	Satellite vars from all domains merged onto 0.1° d01-sized grid
sandy18l.*YYYYMMDDHH*.hwrfsat_i. grb2f[00-126]	Satellite vars from d02 on native d02 grid
sandy18l.*YYYYMMDDHH*.hwrfsat_m. grb2f[00-126]	Satellite vars from d02 & d03 merged onto 0.03° d02-sized grid
sandy18l.*YYYYMMDDHH*.hwrfsat_n. grb2f[00-126]	Satellite vars from d03 on native d03 grid
sandy18l.*YYYYMMDDHH*.hwrfsat_p. grb2f[00-126]	Satellite vars from all domains merged onto 0.25° d01 grid
sandy18l.*YYYYMMDDHH*.hwrftrk. grbf[00-126]	Tracker vars merged from all domains on a 0.03° grid, slightly larger than d02
sandy18l.*YYYYMMDDHH*.hwrftrk. grbf[00-126].grbindex	Index file for hwrftrk
sandy18l.*YYYYMMDDHH*.namelist.input	WRF namelist
sandy18l.*YYYYMMDDHH*.pom.[0000-0005].nc	Daily ocean forecast files
sandy18l.*YYYYMMDDHH*.pom.el_initial.nc	Init salinity data used for MPIPOM-TC
sandy18l.*YYYYMMDDHH*.pom.grid.nc	Gridded ocean initialization data
sandy18l.*YYYYMMDDHH*.pom.ts_clim.nc	Init climatological temperature data used for MPIPOM-TC
sandy18l.*YYYYMMDDHH*.pom.ts_initial.nc	Init temperature data used for MPIPOM-TC
sandy18l.*YYYYMMDDHH*.pom.uv_initial.nc	Init wind data used for MPIPOM-TC
sandy18l.*YYYYMMDDHH*.rainfall.ascii	
sandy18l.*YYYYMMDDHH*.resolution	Text file containing info about nest motion
sandy18L.*YYYYMMDDHH*.resolution	Same as above
sandy18l.*YYYYMMDDHH*.stats.short	Storm info at each forecast hour
sandy18l.*YYYYMMDDHH*.stats.tpc	Storm info at each forecast hour for delivery to NHC
sandy18L.*YYYYMMDDHH*.stats.tpc	Same as above
sandy18l.*YYYYMMDDHH*.swath.ctl	GrADS control file for swath
sandy18l.*YYYYMMDDHH*.swath.dat	Along-track wind and rain information

sandy18l.*YYYYMMDDHH*.track_d03.patcf	Tracker info from d03
sandy18l.*YYYYMMDDHH*.trak.hwrf.3hourly	Tracker output in ATCF format
sandy18l.*YYYYMMDDHH*.trak.hwrf.atcfunix	Same as above
sandy18l.*YYYYMMDDHH*.trak.hwrf.raw	Same as above
sandy18l.*YYYYMMDDHH*.wind10hrly.ascii	Hourly maximum 10m wind
sandy18l.*YYYYMMDDHH*.wind10m.ascii	
sandy18l.*YYYYMMDDHH*.wrfanl_d02	Input analysis for d02
sandy18l.*YYYYMMDDHH*.wrfanl_d02_org	Same as above
sandy18l.*YYYYMMDDHH*.wrfanl_d03	Input analysis for d03
sandy18l.*YYYYMMDDHH*.wrfanl_d03_org	Same as above
sandy18l.*YYYYMMDDHH*.wrfbdy_d01	Boundary conditions for all forecast times
sandy18l.*YYYYMMDDHH*.wrfdiag_d01	WRF auxiliary output stream for d01
sandy18l.*YYYYMMDDHH*.wrfdiag_d02	WRF auxiliary output stream for d02
sandy18l.*YYYYMMDDHH*.wrfdiag_d03	WRF auxiliary output stream for d03
sandy18l.*YYYYMMDDHH*.wrfinput_d01	Input analysis for d01
sandy18l.*YYYYMMDDHH*.wrfinput_d01_org	Same as above

Products

Products is a module that calls an NHCProducts object, whose run module lives in `ush/hwrf/nhc_products.py`, and is responsible for the following tasks.

1. Make namelist `products.nml`
2. Link the input files
3. Run `nhc_products.exe`
4. Deliver output

Output files:

An example for Hurricane Sandy (2012):

```
sandy18l.2012102812.wind10hrly.ascii
sandy18l.2012102812.rainfall.ascii
sandy18l.2012102812.wind10m.ascii
SANDY18L.2012102812.afos
SANDY18L.2012102812.stats.tpc
SANDY18L.2012102812.hwrf_d03.htcf
```

Status Check:

The string `"WARNING: No subtasks incomplete. I think I am done running. Will exit regribber now."` will appear in each of the products standard out files.

Gribber

The Gribber is a GRIBTask object whose run module resides in `ush/hwrf/gribtask.py`. Its primary function is to run `copygb.exe` to horizontally interpolate the native UPP output

files to a variety of regular lat/lon grids.

Output files:

The following sets of files get delivered to the `intercom/`.

`hwrftrk.`*`YYYYMMDD.HH`*`0000`
`hwrftrk.`*`YYYYMMDD.HH`*`0000.grbindex`
`parenthigh.`*`YYYYMMDD.HH`*`0000`
`trkbasic.`*`YYYYMMDD.HH`*`0000`

Status Check:

The string `"INFO: storm1: completed regribbing job for"` will appear in each of the POST standard output files, which will be set by the environment variable `REGRIBBER_LOGS` in the `products_wrapper`.

Executables:

`copygb.exe`

This executable performs two functions. The functions are separated in the explanation by their respective numeric items.

FUNCTION

1. Interpolates a GRIB file to a user-specified grid
2. Combines two GRIB files

INPUT

1. `${hr_grid}` - User-specified grid
   One grib file, for example
2. `${hr_grid}` - User-specified grid
   Two GRIB files, for example

OUTPUT GRIB file on grid `${hr_grid}`

USAGE

1. **`copygb.exe -xg "${hr_grid}"`** *`input_GRIB_file \ out_GRIB_file`*
2. When a "-M" option is used and the argument following it is a GRIB file, the GRIB file will be interpreted as a merge file. This option can be used to combine two GRIB files.
   The following command will combine two GRIB files on different grids, onto a grid specified by `${hr_grid}`, placing the result in *`out_GRIB_file`*.
   `copygb.exe -g "${hr_grid}" -xM` *`input_GRIB_file_1 \ input_GRIB_file_2 out_GRIB_file`*

Tracker

The TrackerTask is responsible for running the GFDL Vortex Tracker. The tracker reads the HWRF postprocessed files in the combined domain. It produces a 3-hourly track and a 6-hourly track for the entire forecast length, as well as another 3-hourly track for the 12-hr forecast, using a merged grid from all three domains with 0.03° resolution. The track for the 12-hr forecast is used in the vortex relocation procedure for the following cycle. The tracker module resides in `ush/hwrf/tracker.py` and performs the following actions.

1. Link the input GRIB files
2. Make the tracker namelist
3. Run `hwrf_gettrk.exe`
4. Deliver files

Output files:

The following output files are an example for Hurricane Sandy (2012).

```
sandy18l.trak.hwrf.raw
sandy18l.trak.hwrf.atcfunix
sandy18l.trak.hwrf.3hourly
sandy18l.trak.hwrf.combine
```

Status Check:

The standard output file will contain the string `"CRITICAL: Successful return status from gettrk."`

Executables:

`hwrf_gettrk.exe`

FUNCTION Runs the GFDL Vortex Tracker

INPUT
- `fort.11` – GRIB file containing the postprocessed HWRF forecast
- `fort.12` – TCVitals file containing the first guess location of the forecast vortex
- `fort.14` – TCVitals file used for tropical cyclogenesis tracking. This file is not used in HWRF's operational configuration. File fort.14, which can be blank, should exist in the directory where the tracker is run, otherwise the tracker will stop.
- `fort.15` – Forecast lead times (in minutes) the tracker will process
- `fort.31` – a GRIB index file generated by the program grbindex
- `input.namelist` – namelist

OUTPUT
- `fort.69` – Modified ATCF file
- `fort.64` – Modified ATCF file
- `fort.66` – Modified ATCF file produced only in "cyclgenesis mode"
- `fort.74` – Modified ATCF file produced when IKEFLAY=Y

USAGE `hwrf_gettrk.exe <namelist`

Refer to Appendix C for a sample namelist and an explanation of contents of output files.

11.2.2 Additional Tracking Utilities

In addition to the utilities of the GFDL Vortex Tracker implemented by HWRF, there are capabilities to generate phase space diagnostics and to run in cyclogenesis mode. The wrapper and Python scripts automatically include these diagnostics. Just for reference, the section below explains the steps necessary to compute phase space diagnostics.

Phase Space Diagnostics

1. In the GFDL Vortex Tracker namelist set the items listed below.
   `phaseflag=y`
   `phasescheme=both` or `cps` or `vtt`
   `wcore_depth`=1.0
2. If `phasescheme` is set to `cps`, run `hwrf_vint.exe` to vertically interpolate the geopotential from 300 to 900 hPa at a 50 hPa interval. Then append these geopotential variables to the tracker's GRIB format input file.
3. If `phasescheme` is set to `vtt` , run `hwrf_vint.exe` to vertically interpolate the temperature from 300 to 500 hPa at a 50 hPa interval. Then run `hwrf_tave.exe` to obtain the average temperature between 300 and 500 hPa. This average temperature field is appended to the tracker's GRIB format input file.
4. If `phasescheme` is set to `both`, then both steps 2) and 3) are needed.
5. When the phase space diagnostics is performed, the output will be generated in `fort.64` as fields 37-41.

Executables:

`hwrf_vint.exe`

FUNCTION Interpolates from various pressure levels onto a regularly spaced grid, with 50-hPa vertical level intervals. Each run only processes one lead time. Therefore it is necessary to use this executable separately for all lead times.

INPUT `fort.11` - GRIB file containing the postprocessed HWRF output with temperature at least at 300 and 500 hPa.
`fort.16` - text file containing the number of input pressure levels.
`fort.31` - index file of `fort.11`
`namelist` - generated by **`echo "&timein \ ifcsthour=$fhour iparm=$gparm/" > namelist`** where `$fhour` is the forecast lead time and `$gparm` is the variable to be processed. For phase space diagnostics, geopotential height (when `phasescheme=cps`, `$gparm=7`) or temperature (when `phasescheme=vtt`, `$gparm=11`) or both (when `phasescheme=both`) need to be processed.

OUTPUT `fort.51` - GRIB file that contains the temperature data on vertical levels 300, 350, 400, 450, and 500 hPa

USAGE **`hwrf_vint.exe < namelist`**

`hwrf_tave.exe`

FUNCTION Vertically averages temperature in the 500-300 hPa layer

INPUT `fort.11` - GRIB file containing the temperature at least at levels 300, 350, 400, 450, and 500 hPa. This file can be generated by `hwrf_vint.exe`
`fort.16` - text file containing the number of input pressure levels.
`fort.31` - index file of `fort.11`
`namelist` - generated by the command **`echo "&timein \ ifcsthour=$fhour, iparm=11/" > namelist`**

OUTPUT `fort.51` - GRIB file containing the mean temperature in the 300-500 hPa layer.

USAGE **`hwrf_tave.exe < namelist`**

Running in Tracker Mode

1. In the GFDL Vortex Tracker namelist set the items listed below.
   `phaseflag=y`
   `phasescheme=both` or `cps` or `vtt`
   `wcore_depth=1.0`
2. If `phasescheme` is set to `cps`, run `hwrf_vint.exe` to vertically interpolate the geopotential from 300 to 900 hPa at a 50 hPa interval. Then append these geopotential variables to the tracker's GRIB format input file.
3. If `phasescheme` is set to `vtt` , run `hwrf_vint.exe` to vertically interpolate the temperature from 300 to 500 hPa at a 50 hPa interval. Then run `hwrf_tave.exe` to obtain the average temperature between 300 and 500 hPa. This average temperature

field is appended to the tracker's GRIB format input file.
4. If phasescheme is set to both, then both steps 2) and 3) are needed.
5. When the phase space diagnostics is performed, the output will be generated in fort.64 as fields 37-41 (see Appendix C).

11.3 How to Plot the Tracker Output Using ATCF_PLOT

The GFDL Vortex Tracker comes with atcf_plot, a set of GrADS scripts that can be used to plot hurricane track files in ATCF format. These scripts can be found in the directory: gfdl-vortextracker/trk_plot/plottrak.

To use atcf_plot to plot the storm's track, perform the following steps.

- Enter the directory gfdl-vortextracker/trk_plot.
- Run gribmap on the GrADS ctl file plottrak.ctl by typing: **gribmap -v -i plottrak.ctl**. gribmap is a GrADS utility that maps the contents of the binary data using the ctl file. It creates a map (plottrak.ix) that points to the locations where the requested binary data start for the different variables and levels.

You should see one line in the output that has "MATCH" in the string. Both the plottrak.ctl and the newly created plottrak.ix map file need to be in the directory where the script below is run.

- Edit the atcfplot.sh to set the following paths.
  1. gradsv2 – path to the GrADS executable (for example, /contrib/grads/bin/gradsc).
  2. GADDIR – path to the directory containing the supplemental font and map files in for GrADS (for example, /contrib/grads/lib).
  3. scrdir – path to the working directory (for example, ${SCRATCH}/sorc/gfdl-vortextracker/trk_plot/plottrak).
  4. plotdir – path to the directory where the plot files will be created (for example, ${SCRATCH}/sorc/gfdl-vortextracker/trk_plot/plottrak/tracks).
- Edit atcfplot.gs to define the following paths.
  1. rundir – same as scrdir in atcfplot.sh (note rundir must end with a "/")
  2. netdir – same as plotdir in atcfplot.sh (note netdir must end with a "/")
  3. Change all instances of HTUT to HCOM
- Edit get_mods.sh to define the following paths.
  1. rundir – same as scrdir in atcfplot.sh
  2. netdir – same as plotdir in atcfplot.sh
  3. ndate – path to the script ndate.ksh
  4. nhour – path to the script nhour.ksh
- Edit get_verif.sh to define the following paths.
  1. rundir – same as scrdir in atcfplot.sh
  2. netdir – same as plotdir in atcfplot.sh
  3. ndate – path to the script ndate.ksh
  4. nhour – path to the script nhour.ksh

- The users need to insert or append their vortex tracker output, `fort.64`, into the file a*BASIN|SID|YYYY*`.dat`. The following two commands are an example of how to do this for Hurricane Sandy a-deck files.

  `sed -i 's/HWRF/HCOM/g' $COMhwrf/`*`YYYYMMDDHH`***`/18L/aal182012_HWRF_hPYHW_YYYYMMDDHH.dat`**

  `cat $COMhwrf/`*`YYYYMMDDHH`***`/18L/aal182012_HWRF_hPYHW_YYYYMMDDHH.dat aal182012.dat`**
- After setting up the paths to the correct locations in your system, run the script using the following command.
  `atcfplot.sh` *YYYY BASIN*

This will start a GUI window and read in ATCF format track files `a${BASIN}${SID}${YYYY}.dat` in `$rundir`.

For example, the user can use the command **`atcfplot.sh 2011 al`** to plot the track files `aal${SID}2011.dat` in `$rundir`.

When the GUI window appears, from the drop down menu, select a storm, start date, and a model name ("atcfname" in the GFDL Vortex Tracker namelist), then click the "Plot" button to plot the track. The plots can be exported to image files by using the "Main" and then "Print" menu options. The default tracker namelist is set to use the ATCF model name "HCOM". If the user changes this name in the tracker namelist, the ATCF_PLOT GUI will not recognize the new name. In this case, the user needs to replace an unused atcfname with the new atcfname. The atcfnames in the GUI can be found by searching in function "modnames" in `atcfplot.gs`. Note all three instances of the unused atcfname need to be replaced in `atcfplot.gs`.

For example, if "USER" was employed as the ATCF model name in the users' GFDL Vortex Tracker output `fort.64`, `atcfplot.gs` needs to be modified to have the ATCF_PLOT program GUI interface show a button for the atcfname "USER". To do that, open `atcfplot.gs`, go to function "modnames", find an atcfname that will not be used, for example "HCOM", and manually replace the string "HCOM" with "USER".

12

HWRF Idealized Tropical Cyclone Simulation

12.1 Introduction

Initial conditions for the HWRF Idealized Tropical Cyclone case are specified using an idealized vortex superposed on a base state quiescent sounding. The default initial vortex has an intensity of 20 ms^{-1} and a radius of maximum winds of 90 km. To initialize the idealized vortex, a nonlinear balance equation in pressure-based sigma coordinates is solved within the rotated latitude-longitude E-grid framework.

The default initial ambient base state assumes an f-plane at the latitude of $12.5°$. The sea surface temperature is time-invariant and horizontally homogeneous, with the default set to 302 K. No land is used in the simulation domain.

The lateral boundary conditions used in the HWRF idealized simulation are the same as used in real data cases. This inevitably leads to some reflection when gravity waves emanating from the vortex reach the outer domain lateral boundaries.

The idealized simulation uses the operational HWRF triple-nested domain configuration with grid spacing at 27-, 9-, and 3-km. All the operational atmospheric physics, as well as the supported experimental physics options in HWRF, can be used in the idealized HWRF framework. The UPP (see Chapter 10) can be used to postprocess the idealized HWRF simulation output.

The setup of the idealized simulation requires the use of WPS to localize the domain (`geogrid.exe`) and to process GFS data for initial and boundary conditions (`ungrib.exe` and `metgrid.exe`). The initialization using WPS just provides a framework for the initial conditions, which are actually specified in `ideal.exe` to be composed of a quiescent environment with a prescribed vortex. The boundary conditions generated with WPS are also overwritten by `ideal.exe` to be consistent with the quiescent environment.

The initial base state temperature and humidity profile is prescribed in file `sound.d`, while the vortex properties are specified in `input.d`. The latter file is also used to specify options for f-plane and β-plane.

12.2 How to Use HWRF for Idealized Tropical Cyclone Simulations

12.2.1 Source Code

This section describes the process to implement HWRF v3.6a in the idealized setting. Only the WPS and WRFV3 components are required for the idealized tropical cyclone simulations. The UPP can be used for postprocessing. The other HWRF components do not need to be compiled. Please see Chapter 2 for instructions to compile the WPS, WRF, and, if desired, UPP. Note that the executable file `wrf.exe` needed for the idealized simulation is not the same as the one needed for the simulation for real data. Therefore, users should follow the instructions specific for building the idealized `wrf.exe`. In this Users' Guide, we assume that the user will install HWRF in directory `${SCRATCH}/hwrfrun`.

12.2.2 Input Files and Datasets

Two GFS GRIB files are needed to provide a template for creating the initial and lateral boundary conditions. One of the GFS GRIB files should be the analysis valid at the same time of the desired HWRF initialization. The other GRIB file should be a forecast, with lead time equal to or greater than the desired HWRF simulation. The meteorological data in these files will not be used to initialize the simulation – these files are for template purposes only.

As an example, files `0825012000000` and `0825512000000`, are included in the tar file `http://www.dtcenter.org/HurrWRF/users/downloads/datasets/Idealized_2014/hwrfv3.6a_idealized.tar.gz`.

Next the user must ensure that all the input files below exist in `${SCRATCH}/hwrfrun/sorc/WRFV3/test/nmm_tropical_cyclone`.

`namelist.wps`	Namelist file for WPS; Note that geog_data_path should be modified to point to the actual path of the geog data files.
`namelist.input`	Namelist file for WRF
`input.d`	Vortex description file
`sound.d`	Sounding data; four sounding files (`sound.d`, `sound_gfdl.d`, `sound_jordan.d`, and `sound_wet.d`) are provided in `${SCRATCH}/hwrfrun/sorc/WRFV3/test/\ nmm_tropical_cyclone`, however, only the one named `sound.d` will be used. In order to use a different sounding, rename it to `sound.d`.
`storm.center`	Vortex center file
`sigma.d`	Sigma file

12.2.3 General Instructions for Running the Executables

To perform the idealized simulation the following executables need to be run: `geogrid.exe`, `ungrib.exe`, `mod_levels.exe`, `metgrid.exe`, `ideal.exe`, and `wrf.exe`. Since the executables are compiled with distributed memory capability, many computing platforms require they be run on compute nodes. Instructions for running jobs on compute nodes can be found in Section 3.5.1.

The wrappers and Python scripts described in previous chapters for running HWRF using real data are not used for the idealized simulation. Since the workflow of the idealized simulation is fairly simple, the commands can be run manually.

12.2.4 Running WPS to Create the ICs and LBCs

The steps below outline the procedure to preprocess the data for the creation of initial and boundary conditions for the idealized simulation. It assumes that the run will be conducted in a working directory named `$WORKDIR/wpsprd`.

1. Create and change into directory for running WPS.
   ```
   mkdir $WORKDIR/wpsprd
   cd $WORKDIR/wpsprd
   ```
2. Run geogrid
   a) Make a directory for the geogrid table and change into it.
      ```
      mkdir geogrid
      cd geogrid
      ```
   b) Link the geogrid table.
      ```
      ln -fs ${SCRATCH}/hwrfrun/sorc/WPSV3/geogrid/\
      GEOGRID.TBL.NMM ./GEOGRID.TBL
      ```
   c) Copy the WPS namelist.
      ```
      cd $WORKDIR/wpsprd
      ```

```
cp ${SCRATCH}/hwrfrun/sorc/WRFV3/test/\
nmm_tropical_cyclone/namelist.wps .
```

d) Edit `namelist.wps` to make sure geog_data_path points to the location of the WPS geographical data files.

e) Run executable `geogrid.exe` on the command line or submit it to a compute node or batch system.

```
${SCRATCH}/hwrfrun/sorc/WPSV3/geogrid.exe
```

f) Verify that the output files were created.

```
ls -l geo_nmm_nest.l01.nc geo_nmm.d01.nc
```

3. Run ungrib

a) Link the ungrib table.

```
ln -fs ${SCRATCH}/hwrfrun/sorc/WPSV3/ungrib/\
Variable_Tables/Vtable.GFS ./Vtable
```

b) Extract the two input GFS files.

Download tarfile with GFS input data `http://www.dtcenter.org/HurrWRF/users/downloads/datasets/Idealized_2014/hwrfv3.6a_idealized.tar.gz`.

```
tar -xzvf hwrfv3.6a_idealized.tar.gz
gunzip 0825012000000.gz
gunzip 0825512000000.gz
ls -l 0825012000000 0825512000000
```

c) Link the GFS files to the names expected by ungrib.

```
${SCRATCH}/hwrfrun/sorc/WPSV3/link_grib.csh \
0825012000000 0825512000000
```

d) Run executable `ungrib.exe` on the command line or submitting it to a compute node or batch system.

```
${SCRATCH}/hwrfrun/sorc/WPSV3/ungrib.exe
```

e) Verify that the output files were created.

```
ls -l GFS:2008-09-06_12 GFS:2008-09-11_12
```

4. Run metgrid

a) Make a directory for the metgrid table and change into it.

```
mkdir metgrid
cd metgrid
```

b) Link the metgrid table.

```
ln -fs ${SCRATCH}/hwrfrun/sorc/WPSV3/metgrid/\
METGRID.TBL.NMM ./METGRID.TBL
```

c) Run executable mod_levels.exe twice on the command line or submitting it to a compute node or batch system. This program is used to reduce the number of vertical levels in the GFS file. Only the levels listed in variable press_pa in namelist.wps will be retained.

```
cd $WORKDIR/wpsprd
${SCRATCH}/hwrfrun/sorc/WPSV3/util/mod_levs.exe \
GFS:2008-09-06_12 new_GFS:2008-09-06_12
${SCRATCH}/hwrfrun/sorc/WPSV3/util/mod_levs.exe \
GFS:2008-09-11_12 new_GFS:2008-09-11_12
```

d) Verify that the output files were created.

```
ls -l new_GFS:2008-09-06_12 new_GFS:2008-09-11_12
```

e) Run executable `metgrid.exe` on the command line or submitting it to a com-

pute node or batch system.

```
${SCRATCH}/hwrfrun/sorc/WPSV3/metgrid.exe
```

f) Verify that the output files were created.

```
ls -l met_nmm.d01.2008-09-06_12:00:00.nc \
met_nmm.d01.2008-09-11_12:00:00.nc
```

12.2.5 Running `ideal.exe` and `wrf.exe`

The steps below outline the procedure to create initial and boundary conditions for the idealized simulation. It assumes that the run will be conducted in a working directory named `$WORKDIR/wrfprd`.

1. Create and change into directory for running ideal and real.

```
mkdir $WORKDIR/wrfprd
cd $WORKDIR/wrfprd
```

2. Run ideal

a) Link WRF input files.

```
ln -fs ${SCRATCH}/hwrfrun/sorc/WRFV3/run/ETAMPNEW_DATA \
ETAMPNEW_DATA
ln -fs ${SCRATCH}/hwrfrun/sorc/WRFV3/run/GENPARM.TBL \
GENPARM.TBL
ln -fs ${SCRATCH}/hwrfrun/sorc/WRFV3/run/LANDUSE.TBL \
LANDUSE.TBL
ln -fs ${SCRATCH}/hwrfrun/sorc/WRFV3/run/SOILPARM.TBL \
SOILPARM.TBL
ln -fs ${SCRATCH}/hwrfrun/sorc/WRFV3/run/VEGPARM.TBL \
VEGPARM.TBL
ln -fs ${SCRATCH}/hwrfrun/sorc/WRFV3/run/tr49t67 tr49t67
ln -fs ${SCRATCH}/hwrfrun/sorc/WRFV3/run/tr49t85 tr49t85
ln -fs ${SCRATCH}/hwrfrun/sorc/WRFV3/run/tr67t85 tr67t85
```

b) Link the WPS files.

```
ln -fs $WORKDIR/wpsprd/met_nmm* .
ln -fs $WORKDIR/wpsprd/geo_nmm* .
```

c) Copy the idealized simulation input files.

```
cp ${SCRATCH}/hwrfrun/sorc/WRFV3/test/\
nmm_tropical_cyclone/input.d .
cp ${SCRATCH}/hwrfrun/sorc/WRFV3/test/\
nmm_tropical_cyclone/sigma.d .
cp ${SCRATCH}/hwrfrun/sorc/WRFV3/test/\
nmm_tropical_cyclone/sound.d .
cp ${SCRATCH}/hwrfrun/sorc/WRFV3/test/\
nmm_tropical_cyclone/storm.center .
```

d) Copy namelist input.

```
cp ${SCRATCH}/hwrfrun/sorc/WRFV3/test/\
nmm_tropical_cyclone/namelist.input .
```

e) Edit and modify files `input.d`, `sound.d`, if desired.

- The sounding files provided have 30 vertical levels. In order to use a sounding with different number of levels, it is necessary to modify the source code in `${SCRATCH}/hwrfrun/sorc/WRFV3/dyn_nmm/\ module_initialize_tropical_cyclone.F`. In subroutine tem, parameter nv should be modified from 30 to the number of levels in the sounding.
- File `storm.center` should not be altered to make sure the storm is located in the center of the inner nest.
- File `sigma.d` should not be modified as it does not pertain to the vertical levels of the sounding or of the simulation. Rather, it defines the vertical levels used to create the initial vortex.

f) Run executable `ideal.exe` on the command line or submitting it to a compute node or batch system.

`${SCRATCH}/hwrfrun/sorc/WRFV3/main/ideal.exe`

g) Verify that the output files were created.

`ls -l wrfinput_d01 wrfbdy_d01 fort.65`

3. Run WRF

a) Run executable `wrf.exe` on the command line or submitting it to a compute node or batch system.

`${SCRATCH}/hwrfrun/sorc/WRFV3/main/wrf.exe`

Note that executable `wrf.exe` must have been created using the instructions for idealized simulations described in Chapter 2. The executable created for regular HWRF simulations that ingest real data should not be used to conduct idealized simulations.

b) Verify that the output files were created.

`ls -l wrfout_d01* wrfout_d02* wrfout_d03*`

A

Example of Computational Resources

Table A.1 gives an example of the resources required to run HWRF compiled with Intel on the NOAA Research Supercomputer Jet and should be used as a guideline for scaling to the resources at the individual user's disposal. In the example below, the available resources include 5440 cores with 16 2.6 GHz per node. Each node has 32 GB (2 GB per core). With peak performance capable of 113.2 TF, and end-to-end run of HWRF would take about 4 hours.

	Wall clock time	Total Cores	Core Layout	Virtual Memory
launcher_wrapper	10:00	1	n/a	n/a
init_gdas_wrapper	2:39:00	32	n/a	n/a
init_gfs_wrapper	2:39:00	32	n/a	30 GB
init_ocean_wrapper	1:39:00	9	n/a	30 GB
relocate_wrapper	3:00:00	n/a	nodes=1	20 GB
gsi_d02_wrapper	0:39:00	n/a	nodes=4:ppn=9+1:ppn=4	25 GB
gsi_d03_wrapper	0:39:00	n/a	nodes=15:ppn=10	25 GB
merge_wrapper	0:39:00	1	n/a	n/a
unpost_wrapper	0:05:00	1	n/a	n/a
forecast_wrapper	3:40:00	210	n/a	n/a
post_wrapper	4:00:00	12	n/a	25 GB
products_wrapper	3:00:00	1	n/a	30 GB

Table A.1: Example of resources required to run HWRF at near operational capability.

B

Example WRF Namelist

The WRF namelist used for the release case, Hurricane Sandy (2012), is listed below.

```
&time_control
start_year                          = 2012, 2012, 2012,
start_month                         = 10, 10, 10,
start_day                           = 28, 28, 28,
start_hour                          = 6, 6, 6,
start_minute                        = 0, 0, 0,
start_second                        = 0, 0, 0,
end_year                            = 2012, 2012, 2012,
end_month                           = 11, 11, 11,
end_day                             = 2, 2, 2,
end_hour                            = 12, 12, 12,
end_minute                          = 0, 0, 0,
end_second                          = 0, 0, 0,
interval_seconds                    = 21600,
history_interval                    = 180, 180, 180,
auxhist1_interval                   = 60, 60, 60,
auxhist2_interval                   = 60, 60, 60,
auxhist3_interval                   = 180, 180, 180,
history_end                         = 540, 540, 540,
auxhist2_end                        = 540, 540, 540,
auxhist1_outname                    = "wrfdiag_d<domain>",
auxhist2_outname                    = "wrfout_d<domain>_<date>",
auxhist3_outname                    = "wrfout_d<domain>_<date>",
frames_per_outfile                  = 1, 1, 1,
frames_per_auxhist1                 = 999, 999, 999,
frames_per_auxhist2                 = 1, 1, 1,
frames_per_auxhist3                 = 1, 1, 1,
analysis                            = F, T, T,
restart                             = F,
restart_interval                    = 36000,
reset_simulation_start              = F,
```

```
io_form_input             = 2,
io_form_history           = 2,
io_form_restart           = 2,
io_form_boundary          = 2,
io_form_auxinput1         = 2,
io_form_auxhist1          = 202,
io_form_auxhist2          = 2,
io_form_auxhist3          = 2,
auxinput1_inname          = "met_nmm.d<domain>.<date>",
debug_level               = 1,
tg_reset_stream           = 1,
override_restart_timers   = T,
io_form_auxhist4          = 2,
io_form_auxhist5          = 2,
io_form_auxhist6          = 2,
io_form_auxinput2         = 2,
nocolons                  = F,
/
&fdda
/
&domains
time_step                 = 45,
time_step_fract_num       = 0,
time_step_fract_den       = 1,
max_dom                   = 3,
s_we                      = 1, 1, 1,
e_we                      = 216, 106, 198,
s_sn                      = 1, 1, 1,
e_sn                      = 432, 204, 354,
s_vert                    = 1, 1, 1,
e_vert                    = 61, 61, 61,
dx                        = 0.18, 0.06, 0.02,
dy                        = 0.18, 0.06, 0.02,
grid_id                   = 1, 2, 3,
tile_sz_x                 = 0,
tile_sz_y                 = 0,
numtiles                  = 1,
nproc_x                   = -1,
nproc_y                   = -1,
parent_id                 = 0, 1, 2,
parent_grid_ratio         = 1, 3, 3,
parent_time_step_ratio    = 1, 3, 3,
i_parent_start            = 0, 96, 20,
j_parent_start            = 0, 219, 45,
feedback                  = 1,
num_moves                 = -99,
num_metgrid_levels        = 27,
p_top_requested           = 200.0,
ptsgm                     = 15000.0,
```

```
eta_levels                = 1.0, 0.995253, 0.990479, 0.985679, 0.980781,
                            0.975782, 0.970684, 0.965486, 0.960187,
                            0.954689, 0.948991, 0.943093, 0.936895,
                            0.930397, 0.923599, 0.916402, 0.908404,
                            0.899507, 0.888811, 0.876814, 0.862914,
                            0.847114, 0.829314, 0.809114, 0.786714,
                            0.762114, 0.735314, 0.706714, 0.676614,
                            0.645814, 0.614214, 0.582114, 0.549714,
                            0.517114, 0.484394, 0.451894, 0.419694,
                            0.388094, 0.356994, 0.326694, 0.297694,
                            0.270694, 0.245894, 0.223694, 0.203594,
                            0.185494, 0.169294, 0.154394, 0.140494,
                            0.127094, 0.114294, 0.101894, 0.089794,
                            0.078094, 0.066594, 0.055294, 0.044144,
                            0.033054, 0.022004, 0.010994, 0.0,
use_prep_hybrid           = T,
num_metgrid_soil_levels   = 2,
/
&physics
num_soil_layers           = 4,
mp_physics                = 85, 85, 85,
ra_lw_physics             = 98, 98, 98,
ra_sw_physics             = 98, 98, 98,
sf_sfclay_physics         = 88, 88, 88,
sf_surface_physics        = 88, 88, 88,
bl_pbl_physics            = 3, 3, 3,
cu_physics                = 84, 84, 0,
mommix                    = 1.0, 1.0, 1.0,
var_ric                   = 1.0,
coef_ric_l                = 0.16,
coef_ric_s                = 0.25,
h_diff                    = 1.0, 1.0, 1.0,
gwd_opt                   = 2, 0, 0,
sfenth                    = 0.0, 0.0, 0.0,
nrads                     = 80, 240, 720,
nradl                     = 80, 240, 720,
nphs                      = 2, 6, 6,
ncnvc                     = 2, 6, 6,
movemin                   = 3, 6, 12,
gfs_alpha                 = 0.7, 0.7, 0.7,
sas_pgcon                 = 0.55, 0.2, 0.2,
sas_mass_flux             = 0.5, 0.5, 0.5,
co2tf                     = 1,
vortex_tracker            = 2, 2, 7,
nomove_freq               = 0, 6, 6,
tg_option                 = 1,
ntornado                  = 1, 3, 12,
/
&dynamics
non_hydrostatic           = T, T, T,
euler_adv                 = F,
wp                        = 0, 0, 0,
coac                      = 0.75, 3.0, 4.0,
codamp                    = 6.4, 6.4, 6.4,
terrain_smoothing         = 2,
/
&bdy_control
spec_bdy_width            = 1,
```

```
specified                 = T,
/
&namelist_quilt
poll_servers              = T,
nio_tasks_per_group       = 4,
nio_groups                = 4,
/
&logging
compute_slaves_silent     = T,
io_servers_silent         = T,
stderr_logging            = 0,
/
```

C

Sample GFDL Vortex Tracker Namelist

Sample namelist

inp%bcc	First 2 digits of the year for the initial time of the forecast (e.g., the "20" in "2012")
inp%byy	Last 2 digits of the year for the initial time of the forecast (e.g., the "12" in "2012")
inp%bmm	2-digit month (01, 02, etc) for the initial time of the forecast
inp%bdd	2-digit day for the initial time of the forecast
inp%bhh	2-digit hour for the initial time of the forecast
inp%model	Model ID number as defined by the user in the script. This is used in subroutine getdata to define what the GRIB IDs are for surface wind levels. Create a unique number in the script for your model and make sure you have the corresponding IDs set up for it in subroutine getdata. For HWRF use 17. The Model ID numbers for other models are listed below: (1) GFS, (2) MRF, (3) UKMET, (4) ECMWF, (5) NGM, (6) NAM, (7) NOGAPS, (8) GDAS, (10) NCEP Ensemble, (11) ECMWF Ensemble, (13) SREF Ensemble, (14) NCEP Ensemble, (15) CMC, (16) CMC Ensemble, (18) HWRF Ensemble, (19) HDAS, (20) Ensemble RELOCATION (21) UKMET hi-res (NHC)

inp%lt_units	'hours' or 'minutes', this defines the lead time units used by the PDS in your GRIB header
inp%file_seq	'onebig' or 'multi', this specifies if the tracker will process one big input file or multiple files for each individual lead times. 'onebig' is used as the default method in the community HWRF scripts
inp%modtyp	Type of the model. Either 'global' or 'regional'. For HWRF, choose 'regional'
inp%nesttyp	Type of the nest grid. Either 'moveable' or 'fixed'. For HWRF, choose 'moveable'
fnameinfo%gmodname	Defines the model name in the input files, e.g., 'hwrf'. Only when inp%file_seq='multi'
fnameinfo%rundescr	Describes the model runs in the input files, e.g., 'combined'. Only when inp%file_seq= 'multi'
fnameinfo%atcfdescr	Describe the storm information in the input files, e.g., 'irene09l'. Only when inp%file_seq='multi'
atcfnum	Obsolete; can be set to any integer
atcfname	Character model ID that will appear in the ATCF output (e.g., GFSO, HWRF, AHW, HCOM etc)
atcfymdh	10-digit yyyymmddhh date that will be used in output text track files
atcffreq	Frequency (in centahours) of output for atcfunix.Default value is 600 (six hourly).
trkrinfo%westbd	For genesis runs, the western boundary for searching for new storms. Does not need to match the boundaries of your grid, it can be smaller
trkrinfo%eastbd	For genesis runs, the eastern boundary for searching for new storms. Does not need to match the boundaries of your grid, it can be smaller than your grid.
trkrinfo%northbd	For genesis runs, the northern boundary for searching for new storms. Does not need to match the boundaries of your grid, it can be smaller than your grid.
trkrinfo%southbd	For genesis runs, the southern boundary for searching for new storms. Does not need to match the boundaries of your grid, it can be smaller than your grid.
trkrinfo%type	trkrinfo%type defines the type of tracking to do. A 'tracker' run functions as the standard TC tracker and tracks only storms from the TC Vitals. 'tcgen' and 'midlat' run in genesis mode and will look for new storms in addition to tracking from TC Vitals. 'tcgen' will look for all parameters at the various vertical levels, while 'midlat' will only look for MSLP and no checks are performed to differentiate tropical from non-tropical cyclones.For HWRF, choose 'tracker'.
trkrinfo%mslpthresh	Threshold for the minimum MSLP gradient (units hPa/km) that must be met in order to continue tracking.
trkrinfo%v850thresh	Threshold for the minimum azimuthally-average 850 hPa cyclonic tangential wind speed (m/s) that must be exceeded in order to keep tracking.
trkrinfo%gridtype	'global' or 'regional', this defines the type of domain grid. For HWRF or other limited area models, choose 'regional'.

trkrinfo%contint	This specifies the interval (in Pa) used by subroutine check_closed_contour to check for a closed contour in the mslp field when running in genesis mode. Note that check_closed_contour is also called from the routine that checks for a warm core, but the contour interval is hard-wired in the executable as 1.0 degree K for that usage.
trkrinfo%out_vit	This is only set to 'y' if the tracker is running in genesis mode, and it tells the tracker to write out a "TC Vitals" record for any storms that it finds at the model initialization time. For HWRF, choose 'n'.
phaseflag	'y' or 'n', tells the program whether or not to determine the cyclone thermodynamic phase
phasescheme	'cps', 'vtt', 'both', tells the program which scheme to use for checking the cyclone phase. 'cps' is Hart's cyclone phase space, 'vtt' is a simple 300-500 hPa warm core check based on Vitart, and 'both' tells the program to use both schemes. Not used if phaseflag='n'
wcore_depth	The contour interval (in deg K) used in determining if a closed contour exists in the 300-500 hPa temperature data, for use with the vtt scheme
structflag	'y' or 'n', tells the program whether or not to determine the cyclone thermodynamic structure.
Ikeflag	'y' or 'n', tells the program whether or not to calculate the Integrated Kinetic Energy (IKE) and Storm Surge Damage Potential (SDP).
use_waitfor	'y' or 'n', for waiting for input files. Use 'n' unless for real-time operational runs.
verb	Level of detail printed to terminal. Choose from 0 (no output),1 (error messages only), 2 (more messages), 3 (all messages).

Contents of the output files

A sample of the vortex tracker output `fort.69` is listed below:

AL, 18, 2012102806, 03, HCOM, 00000, 315N, 737W, 65, 949, XX, 34, NEQ, 0351, 0106, 0141, 0329, 0, 0, 36
AL, 18, 2012102806, 03, HCOM, 00000, 315N, 737W, 65, 949, XX, 50, NEQ, 0058, 0058, 0070, 0070, 0, 0, 36
AL, 18, 2012102806, 03, HCOM, 00000, 315N, 737W, 65, 949, XX, 64, NEQ, 0000, 0000, 0039, 0037, 0, 0, 36
AL, 18, 2012102806, 03, HCOM, 00300, 319N, 731W, 65, 951, XX, 34, NEQ, 0333, 0133, 0205, 0323, 0, 0, 30
AL, 18, 2012102806, 03, HCOM, 00300, 319N, 731W, 65, 951, XX, 50, NEQ, 0064, 0065, 0065, 0058, 0, 0, 30
AL, 18, 2012102806, 03, HCOM, 00300, 319N, 731W, 65, 951, XX, 64, NEQ, 0035, 0043, 0045, 0000, 0, 0, 30
AL, 18, 2012102806, 03, HCOM, 00600, 323N, 724W, 60, 956, XX, 34, NEQ, 0313, 0173, 0205, 0308, 0, 0, 39
AL, 18, 2012102806, 03, HCOM, 00600, 323N, 724W, 60, 956, XX, 50, NEQ, 0068, 0080, 0076, 0043, 0, 0, 39
AL, 18, 2012102806, 03, HCOM, 00900, 329N, 718W, 57, 959, XX, 34, NEQ, 0297, 0192, 0217, 0306, 0, 0, 43
AL, 18, 2012102806, 03, HCOM, 00900, 329N, 718W, 57, 959, XX, 50, NEQ, 0064, 0069, 0054, 0000, 0, 0, 43
AL, 18, 2012102806, 03, HCOM, 01200, 337N, 715W, 57, 959, XX, 34, NEQ, 0274, 0280, 0257, 0285, 0, 0, 42
AL, 18, 2012102806, 03, HCOM, 01200, 337N, 715W, 57, 959, XX, 50, NEQ, 0076, 0076, 0000, 0041, 0, 0, 42
AL, 18, 2012102806, 03, HCOM, 01500, 344N, 713W, 58, 958, XX, 34, NEQ, 0319, 0269, 0274, 0323, 0, 0, 41
AL, 18, 2012102806, 03, HCOM, 01500, 344N, 713W, 58, 958, XX, 50, NEQ, 0125, 0060, 0146, 0094, 0, 0, 41
AL, 18, 2012102806, 03, HCOM, 01800, 349N, 716W, 64, 957, XX, 34, NEQ, 0343, 0340, 0269, 0319, 0, 0, 106
AL, 18, 2012102806, 03, HCOM, 01800, 349N, 716W, 64, 957, XX, 50, NEQ, 0163, 0127, 0153, 0159, 0, 0, 106
AL, 18, 2012102806, 03, HCOM, 02100, 353N, 717W, 75, 955, XX, 34, NEQ, 0351, 0264, 0252, 0290, 0, 0, 39

Column 1:	basin name. "AL" represents Atlantic and "EP" northeast Pacific.
Column 2:	ATCF storm ID number. Sandy was the 18th storm in the Atlantic Basin in 2012.
Column 3:	model starting time.
Column 4:	constant and 03 simply indicates that this record contains model forecast data.
Column 5:	model ATCF name.

Column 6:	forecast lead time in hours multiplied by 100 (e.g, 00900 represents 9.00 h).
Column 7-8:	vortex center position (latitude and longitude multiplied by 10).
Column 9:	vortex maximum 10-m wind (in kt).
Column 10:	vortex minimum MSLP (in hPa).
Column 11:	placeholder for character strings that indicate whether the storm is a depression, tropical storm, hurricane, subtropical storm etc. Currently, that storm type character string is only used for the observed storm data in the NHC Best Track data set.
Column 12:	thresholds wind speed in knots, an identifier that indicates whether this record contains radii for the 34-, 50- or 64-knot wind thresholds.
Column 13:	"NEQ" indicates that the four radii values that follow will begin in the northeast quadrant and progress clockwise.
Column 14-17:	wind radii (in nm) for the threshold winds in each quadrant.
Column 18-19:	not used.
Column 20:	radius of maximum winds, in nautical miles.

A sample of the vortex tracker output `fort.64` is listed below:

AL, 18, 2012102806, 03, HCOM, 000, 315N, 737W, 65, 949, XX, 34, NEQ, 0351, 0106, 0141, 0329, 0, 0, 36, 0, 0, , 0, , 0, 0, , , , , 0, 0, 0, 0, THERMO PARAMS, -9999, -9999, -9999, U, 10, DT, -999
AL, 18, 2012102806, 03, HCOM, 000, 315N, 737W, 65, 949, XX, 50, NEQ, 0058, 0058, 0070, 0070, 0, 0, 36, 0, 0, , 0, , 0, 0, , , , , 0, 0, 0, 0, THERMO PARAMS, -9999, -9999, -9999, U, 10, DT, -999
AL, 18, 2012102806, 03, HCOM, 000, 315N, 737W, 65, 949, XX, 64, NEQ, 0000, 0000, 0039, 0037, 0, 0, 36, 0, 0, , 0, , 0, 0, , , , , 0, 0, 0, 0, THERMO PARAMS, -9999, -9999, -9999, U, 10, DT, -999
AL, 18, 2012102806, 03, HCOM, 003, 319N, 731W, 65, 951, XX, 34, NEQ, 0333, 0133, 0205, 0323, 0, 0, 30, 0, 0, , 0, , 0, 0, , , , , 0, 0, 0, 0, THERMO PARAMS, -9999, -9999, -9999, U, 10, DT, -999
AL, 18, 2012102806, 03, HCOM, 003, 319N, 731W, 65, 951, XX, 50, NEQ, 0064, 0065, 0065, 0058, 0, 0, 30, 0, 0, , 0, , 0, 0, , , , , 0, 0, 0, 0, THERMO PARAMS, -9999, -9999, -9999, U, 10, DT, -999
AL, 18, 2012102806, 03, HCOM, 003, 319N, 731W, 65, 951, XX, 64, NEQ, 0035, 0043, 0045, 0000, 0, 0, 30, 0, 0, , 0, , 0, 0, , , , , 0, 0, 0, 0, THERMO PARAMS, -9999, -9999, -9999, U, 10, DT, -999
AL, 18, 2012102806, 03, HCOM, 006, 323N, 724W, 60, 956, XX, 34, NEQ, 0313, 0173, 0205, 0308, 0, 0, 39, 0, 0, , 0, , 0, 0, , , , , 0, 0, 0, 0, THERMO PARAMS, -9999, -9999, -9999, U, 10, DT, -999
AL, 18, 2012102806, 03, HCOM, 006, 323N, 724W, 60, 956, XX, 50, NEQ, 0068, 0080, 0076, 0043, 0, 0, 39, 0, 0, , 0, , 0, 0, , , , , 0, 0, 0, 0, THERMO PARAMS, -9999, -9999, -9999, U, 10, DT, -999
AL, 18, 2012102806, 03, HCOM, 009, 329N, 718W, 57, 959, XX, 34, NEQ, 0297, 0192, 0217, 0306, 0, 0, 43, 0, 0, , 0, , 0, 0, , , , , 0, 0, 0, 0, THERMO PARAMS, -9999, -9999, -9999, U, 10, DT, -999
AL, 18, 2012102806, 03, HCOM, 009, 329N, 718W, 57, 959, XX, 50, NEQ, 0064, 0069, 0054, 0000, 0, 0, 43, 0, 0, , 0, , 0, 0, , , , , 0, 0, 0, 0, THERMO PARAMS, -9999, -9999, -9999, U, 10, DT, -999
AL, 18, 2012102806, 03, HCOM, 012, 337N, 715W, 57, 959, XX, 34, NEQ, 0274, 0280, 0257, 0285, 0, 0, 42, 0, 0, , 0, , 0, 0, , , , , 0, 0, 0, 0, THERMO PARAMS, -9999, -9999, -9999, U, 10, DT, -999
AL, 18, 2012102806, 03, HCOM, 012, 337N, 715W, 57, 959, XX, 50, NEQ, 0076, 0076, 0000, 0041, 0, 0, 42, 0, 0, , 0, , 0, 0, , , , , 0, 0, 0, 0, THERMO PARAMS, -9999, -9999, -9999, U, 10, DT, -999AL, 18, 2012102806, 03, HCOM, 015, 344N, 713W, 58, 958, XX, 34, NEQ, 0319, 0269, 0274, 0323, 0, 0, 41, 0, 0, , 0, , 0, 0, , , , , 0, 0, 0, 0, THERMO PARAMS, -9999, -9999, -9999, U, 10, DT, -999

Column 1-20:	same as `fort.69` except that column 6, the forecast lead time, instead of being a 5-digit integer as in fort.69, is a 3-digit integer.
Column 21-35:	space fillers.
Column 36:	"THERMO PARAMS" indicating that thermodynamics parameters will follow.
Column 37-39:	The three cyclone phase space parameters, and all values shown have been multiplied by a factor of 10. The values are listed below.

1. Parameter B (left-right thickness asymmetry)
2. Thermal wind (warm/cold core) value for lower troposphere (900-600 hPa)
3. Thermal wind value for upper troposphere (600-300 hPa)

Column 40:	Presence of a warm core. In this sample it is "U", which stands for "undetermined", meaning the warm core check was not performed. When the warm core check is performed, this field will be either 'Y' or 'N', indicating whether the warm core is identified or not.

Column 41: Warm core strength x 10 (in degrees). It indicates the value of the contour interval that was used in performing the check for the warm core in the 300-500 hPa layer.
Column 42-43: Constant strings.

A sample of the vortex tracker output `fort.66` is listed below:

TG, 0001, 2011082312_F000_204N_0706W_FOF, 2011082312, 03, HCOM, 000, 204N, 706W, 87, 978, XX, 34, NEQ, 0103, 0077, 0058, 0095, 1005, 56, 24, -999, -9999, -9999, U, 288, 39, 1191, 6649, 1186, 5714
TG, 0001, 2011082312_F000_204N_0706W_FOF, 2011082312, 03, HCOM, 000, 204N, 706W, 87, 978, XX, 50, NEQ, 0058, 0042, 0032, 0054, 1005, 56, 24, -999, -9999, -9999, U, 288, 39, 1191, 6649, 1186, 5714
TG, 0001, 2011082312_F000_204N_0706W_FOF, 2011082312, 03, HCOM, 000, 204N, 706W, 87, 978, XX, 64, NEQ, 0043, 0027, 0019, 0041, 1005, 56, 24, -999, -9999, -9999, U, 288, 39, 1191, 6649, 1186, 5714
TG, 0001, 2011082312_F000_204N_0706W_FOF, 2011082312, 03, HCOM, 006, 208N, 714W, 94, 965, XX, 34, NEQ, 0156, 0096, 0059, 0145, 976, 17, 21, -999, -9999, -9999, U, 292, 45, 1164, 3306, 1116, 3302
TG, 0001, 2011082312_F000_204N_0706W_FOF, 2011082312, 03, HCOM, 006, 208N, 714W, 94, 965, XX, 50, NEQ, 0065, 0056, 0037, 0058, 976, 17, 21, -999, -9999, -9999, U, 292, 45, 1164, 3306, 1116, 3302
TG, 0001, 2011082312_F000_204N_0706W_FOF, 2011082312, 03, HCOM, 006, 208N, 714W, 94, 965, XX, 64, NEQ, 0047, 0031, 0030, 0042, 976, 17, 21, -999, -9999, -9999, U, 292, 45, 1164, 3306, 1116, 3302
TG, 0001, 2011082312_F000_204N_0706W_FOF, 2011082312, 03, HCOM, 012, 209N, 722W, 93, 964, XX, 34, NEQ, 0123, 0098, 0059, 0104, 979, 21, 21, -999, -9999, -9999, U, 282, 42, 1187, 3668, 1122, 2694
TG, 0001, 2011082312_F000_204N_0706W_FOF, 2011082312, 03, HCOM, 012, 209N, 722W, 93, 964, XX, 50, NEQ, 0069, 0053, 0047, 0058, 979, 21, 21, -999, -9999, -9999, U, 282, 42, 1187, 3668, 1122, 2694
TG, 0001, 2011082312_F000_204N_0706W_FOF, 2011082312, 03, HCOM, 012, 209N, 722W, 93, 964, XX, 64, NEQ, 0044, 0033, 0033, 0044, 979, 21, 21, -999, -9999, -9999, U, 282, 42, 1187, 3668, 1122, 2694

Column 1: "TG", the basin id for cyclogenesis (when trkrinfo%type is set to midlat, this id is named "ML")
Column 2: the number of cyclogenesis the tracker identified
Column 3: the ID for the cyclogenesis, ${YYYYMMDDHH}_F${FFF}_$Lat_$Lon_FOF where YYYYMMDDHH, FFF, Lat and Lon are the model initialization time, the forecast lead time, the latitude and the longitude, respectively, in which the cyclogenesis was first identified
Column 4-18: same as Columns 3-17 in `fort.64`
Column 19: pressure of last closed isobar (in hPa)
Column 20: radius of last closed isobar (nm)
Column 21: radius of maximum wind (nm)
Column 22-24: The cyclone phase space parameters, and all values shown have been multiplied by a factor of 10. The values are listed below

1. Parameter B (left-right thickness asymmetry)
2. Thermal wind (warm/cold core) value for lower troposphere (900-600 hPa)
3. Thermal wind value for upper troposphere (600-300 hPa)

Column 25: Presence of a warm core. In this sample it is "U", which stands for "undetermined", meaning the warm core check is not performed. When the warm core check is performed, this field will be either 'Y' or 'N', indicating whether the warm core is identified or not
Column 26: storm moving direction (in degrees)
Column 27: storm moving speed (in tenths of ms^{-1})
Column 28: mean 850 hPa vorticity (s^{-1}x10e5)
Column 29: max (gridpoint) 850 hPa vorticity (s^{-1}x10e5)
Column 30: mean 700 hPa vorticity (s^{-1}x10e5)
Column 31: max (gridpoint) 700 hPa vorticity (s^{-1}x10e5)

A sample of the vortex tracker output `fort.74` is listed below:

```
AL, 09, 2011082312, 03, HCOM, 000, 204N, 706W, 87, 978, XX, 91, IKE, 0, 23, 34, 16, 5, 0, 0, 0, 2039N, 7062W
AL, 09, 2011082312, 03, HCOM, 006, 208N, 714W, 94, 965, XX, 91, IKE, 0, 28, 42, 25, 8, 0, 0, 0, 2081N, 7142W
AL, 09, 2011082312, 03, HCOM, 012, 209N, 722W, 93, 964, XX, 91, IKE, 0, 28, 44, 25, 8, 0, 0, 0, 2088N, 7220W
AL, 09, 2011082312, 03, HCOM, 018, 213N, 728W, 99, 962, XX, 91, IKE, 0, 25, 46, 19, 9, 0, 0, 0, 2131N, 7276W
AL, 09, 2011082312, 03, HCOM, 024, 218N, 733W, 92, 962, XX, 91, IKE, 0, 27, 50, 23, 8, 0, 0, 0, 2179N, 7333W
AL, 09, 2011082312, 03, HCOM, 030, 225N, 741W, 97, 959, XX, 91, IKE, 0, 28, 51, 26, 9, 0, 0, 0, 2245N, 7415W
AL, 09, 2011082312, 03, HCOM, 036, 231N, 749W, 95, 961, XX, 91, IKE, 0, 29, 51, 27, 11, 0, 0, 0, 2314N, 7488W
AL, 09, 2011082312, 03, HCOM, 042, 239N, 756W, 100, 956, XX, 91, IKE, 0, 29, 54, 28, 11, 0, 0, 0, 2387N, 7562W
AL, 09, 2011082312, 03, HCOM, 048, 248N, 762W, 107, 953, XX, 91, IKE, 0, 30, 58, 30, 14, 0, 0, 0, 2479N, 7621W
AL, 09, 2011082312, 03, HCOM, 054, 258N, 767W, 111, 949, XX, 91, IKE, 0, 32, 62, 34, 16, 0, 0, 0, 2575N, 7668W
AL, 09, 2011082312, 03, HCOM, 060, 267N, 770W, 113, 946, XX, 91, IKE, 0, 33, 65, 38, 18, 0, 0, 0, 2668N, 7696W
AL, 09, 2011082312, 03, HCOM, 066, 277N, 773W, 111, 944, XX, 91, IKE, 0, 34, 67, 40, 21, 0, 0, 0, 2769N, 7731W
AL, 09, 2011082312, 03, HCOM, 072, 286N, 774W, 114, 944, XX, 91, IKE, 0, 35, 68, 42, 23, 0, 0, 0, 2864N, 7742W
AL, 09, 2011082312, 03, HCOM, 078, 296N, 775W, 113, 941, XX, 91, IKE, 0, 35, 73, 43, 22, 0, 0, 0, 2959N, 7753W
AL, 09, 2011082312, 03, HCOM, 084, 304N, 774W, 107, 944, XX, 91, IKE, 0, 35, 74, 43, 22, 0, 0, 0, 3037N, 7742W
AL, 09, 2011082312, 03, HCOM, 090, 312N, 774W, 108, 941, XX, 91, IKE, 0, 36, 77, 46, 23, 0, 0, 0, 3119N, 7745W
AL, 09, 2011082312, 03, HCOM, 096, 320N, 773W, 107, 942, XX, 91, IKE, 0, 37, 79, 51, 26, 0, 0, 0, 3198N, 7728W
AL, 09, 2011082312, 03, HCOM, 102, 328N, 772W, 111, 938, XX, 91, IKE, 0, 38, 78, 53, 28, 0, 0, 0, 3278N, 7719W
AL, 09, 2011082312, 03, HCOM, 108, 336N, 769W, 111, 937, XX, 91, IKE, 0, 37, 76, 51, 30, 0, 0, 0, 3360N, 7690W
AL, 09, 2011082312, 03, HCOM, 114, 347N, 766W, 106, 939, XX, 91, IKE, 0, 35, 68, 43, 21, 0, 0, 0, 3473N, 7664W
AL, 09, 2011082312, 03, HCOM, 120, 361N, 764W, 90, 950, XX, 91, IKE, 0, 32, 57, 35, 10, 0, 0, 0, 3611N, 7642W
AL, 09, 2011082312, 03, HCOM, 126, 375N, 764W, 69, 957, XX, 91, IKE, 0, 27, 42, 24, 2, 0, 0, 0, 3745N, 7637W
```

Column 1-11:	Same as `fort.64`
Column 12-13:	fixed fields
Column 14:	wind damage potential (wdp) (not computed in this version, therefore is always zero)
Column 15:	storm surge damage potential (SDP) (multiplied by 10)
Column 16-18:	IKE, in terajoule, for 10 ms^{-1}, 18 ms^{-1} and 33 ms^{-1} winds, respectively
Column 19-21:	IKE for 25-40 ms^{-1}, 41-54 ms^{-1} and 55 ms^{-1} winds, currently not computed, therefore are always zero
Column 22-23:	vortex center position (latitude and longitude multiplied by 100)