# Aether Documentation

***Release 0.0.1***

**Aether Developers**

**Oct 11, 2022**

# GETTING STARTED

Welcome to the documentation of the Aether Model.

Aether is an extremely flexible community-based multi-scale ionosphere-thermosphere model. Ensemble capability, data assimilation, and uncertainty quantification are fundamentally integrated into the core of the model.

Aether is an open-source model, allowing contributors from across the globe to create, incorporate, and commit new solvers, schemes, and physics modules.

Users have straightforward defaults in the model configuration, but are able to select any contributor's modules in order to allow for maximum flexibility. For example, users can employ faster, less accurate solvers (for prediction) and slower, more accurate solvers (for science).

CHAPTER

ONE

# GITHUB

Since Aether is an open-source project, all of the model source code is available via GitHub.com.

**Note:** If you want to contribute your own code to Aether, please read the *Contributors' Guide*.

If you just want to download the source code, use the following steps.

## 1.1 Download the source code from GitHub

This guide assumes you are using an operating system such as MacOS, Linux or UNIX.

1. Open a terminal window, navigate to the directory you want the source code downloaded into and clone the repository.

```
$ git clone https://github.com/AetherModel/Aether.git
```

2. Navigate into `src` subdirectory of the repository.

```
$ cd Aether/src
$ ls
Makefile                   init_geo_grid.cpp
add_sources.cpp                    inputs.cpp
advance.cpp                        ions.cpp
bfield.cpp                         main.cpp
calc_chemical_sources.cpp neutrals.cpp
calc_chemistry.cpp                 output.cpp
[...]
file_input.cpp                     test.cpp
fill_grid.cpp                      time.cpp
grid.cpp                   time_conversion.cpp
indices.cpp                        transform.cpp
```

CHAPTER

TWO

# SYSTEM REQUIREMENTS

Aether is written using C, C++ and python. Aether also uses NetCDF.

## 2.1 C and C++ compilers

The GNU compiler collection provides compilers for C and C++.

## 2.2 Python

Python is an interpreted language, meaning it doesn't need to be compiled.

## 2.3 NetCDF

The Network Common Data Form (NetCDF) is a set of software libraries created by Unidata.

CHAPTER

THREE

# PLEIADES

## 3.1 Environment overview

Pleiades' environment is protected by secure front ends (SFEs) that users must log in to before ssh'ing into either the Pleiades front ends (PFEs) or the Lou front ends (LFEs).

## 3.2 Logging on and getting onto a PFE

```
ssh $USER@sfe7.nas.nasa.gov
[Password]
[PASSCODE from SecurID]
ssh pfe
[Password]
```

## 3.3 Home and scratch

The home directory can be accessed via:

```
/home3/$USER
```

The scratch directory can be accessed via:

```
/nobackup/$USER
```

## 3.4 Accounting

Jobs are charged different rates depending on which machine they are run on.

A user's default group id is listed in the `/etc/passwd` file.

CHAPTER

FOUR

# INSTALLING CESM

Community Earth System Model (CESM) installation instructions are available via the README on the GitHub repository. The cube sphere grid is available as of CESM2.2.0.

**Note:** svn isn't installed on the pfe nodes, thus the `checkout_externals` script fails because it is needed to download `CESM/components/cam/chem_proc`. To get around this issue, download CESM on a laptop or Cheyenne, tar the resulting file and transfer it to Pleiades.

## 4.1 Download CESM on a laptop or Cheyenne

```
cd <installation_directory>
git clone https://github.com/ESCOMP/CESM.git cesm2_2_0
cd cesm2_2_0
git checkout release-cesm2.2.0
./manage_externals/checkout_externals
```

## 4.2 Log on to a pfe node

```
cd /nobackup/$USER
sftp <user>@data-access.ucar.edu
get /glade/work/$USER/CESM.tar.gz
exit
tar -xvf CESM.tar.gz
```

CHAPTER

FIVE

# CREATING A CUBE SPHERE CASE

**Important:** CESM has already been ported and should work "out of the box" on most of the supercomputers that are widely used in the geosciences community, including Pleiades. When compiling the model, ensure to set the machine command line option, `--mach` to match the supercomputer you are working on. However, the modules that are installed on any machine change over time. CESM requires knowledge of which MPI and netCDF libraries are available. These modules are set in the `config_machines.xml` file.

## 5.1 config_machines.xml

As of February 22, 2022, the `config_machines.xml` file is configured properly to compile CESM on Pleiades Broadwell cluster which is denoted as `pleaides-bro`.

The `config_machines.xml` can be copied from:

```
/nobackup/bjohns28/CESM/cime/config/cesm/machines/config_machines.xml
```

to the analagous path in your own CESM installation or you can use the `create_newcase` build script in this CESM installation.

## 5.2 Available grids

Lauritzen et al. (2017)[1] list the available cube sphere grids in their Table 1. A subset of their table is reproduced here.

| Grid name | Average node spacing | Model timestep |
|---|---|---|
| ne16np4 | 208 km | 1,800 s |
| ne30np4 | 111 km | 1,800 s |
| ne60np4 | 56 km | 900 s |
| ne120np4 | 28 km | 450 s |
| ne240np4 | 14 km | 225 s |

[1] Lauritzen, P. H., and Coauthors, 2018: NCAR Release of CAM-SE in CESM2.0: A Reformulation of the Spectral Element Dynamical Core in Dry-Mass Vertical Coordinates With Comprehensive Treatment of Condensates and Energy. Journal of Advances in Modeling Earth Systems, **10**, 1537–1570, doi:10.1029/2017MS001257.

## 5.3 Building a case

The scripts for building cases within CESM are part of a software collection known as the Common Infrastructure for Modeling the Earth (CIME). This software supports both NCAR models and those developed within the Department of Energy's Energy Exascale Earth System Model (E3SM) collection. Thus the build scripts to create a new case are contained within the `cime` subdirectory.

```
cd /nobackup/bjohns28/CESM/cime/scripts
ls
climate_reproducibility  create_newcase  data_assimilation     lib           query_
↪testlists  Tools
create_clone             create_test     fortran_unit_testing  query_config  tests
```

The `create_newcase` script is invoked and passed command line arguments to build a new case.

| Command line option | Meaning |
|---|---|
| `--case` | The directory the case will be built in. It is common practice to include the experiment's grid resolution and component set (described below) in the name of the case so that these aspects can be easily identified when browsing the file system later. |
| `--compset` | The component set of the experiment, including which models will be actively integrating (atmosphere, land, ocean, sea ice) and what boundary forcing will be used. CESM has an extensive list of component set definitions and these instructions using the `FHIST` compset, which has an active atmospheric component, the Community Atmosphere Model version 6, and historical sea surface forcing, staring in 1979. |
| `--res` | The grid resolution the model will run on. Each grid includes at least two parts, the atmospheric/land grid and the ocean/sea ice grid to which it is coupled. These instructions use a low-resolution cube-sphere grid for the atmosphere, `ne30np4` and couple it to a ~1° ocean/sea ice grid, `gx1v7`. These grid names are truncated into `ne30_g17`. Again, CESM has an extensive list of available grids. |
| `--mach` | The upercomputer the case will be built on. These instructions build a case on NCAR's Cheyenne computer, however, if you are building on Pleiades, consult the table in the note below. |
| `--project` | The account code the project will be run on. When jobs from the experiment are run, the specified account will automatically be debited. Replace `PXXXXXXXX` with your project code. |
| `--run-unsupported` | Since the cube-sphere grid is a newly released aspect of CESM that is not used in Coupled Model Intercomparison Project runs, it is not considered a scientifically supported grid yet. In order to use it, you need to append this option. |

**Note:** If you are building on `pleiades`, the core layout per node differs based on which nodes you are using. These differences are alreay accounted for within CESM. When specifying `--mach` there are four valid options:

| Compute node processor | Corresponding `--mach` option |
|---|---|
| Broadwell | `pleiades-bro` |
| Haswell | `pleiades-has` |
| Ivy Bridge | `pleiades-ivy` |
| Sandy Bridge | `pleiades-san` |

### 5.3.1 Identifying your GroupID

You will need to find your GroupID on NASA systems using the `groups` command:

```
groups $USER
<user> : sXXXX
```

Insert the returned group after the `--project` option when invoking `create_newcase` below.

To build a case using the ~1° `ne30` cube sphere grid:

```
mkdir /nobackup/bjohns28/cases
cd /nobackup/bjohns28/CESM/cime/scripts
./create_newcase --case /nobackup/bjohns28/cases/FHIST.cesm2_2_0.ne30_g17.001 --compset␣
↪FHIST --res ne30_g17 --mach pleiades-bro --project sXXXX --run-unsupported
[...]
Creating Case directory /nobackup/bjohns28/cases/FHIST.cesm2_2_0.ne30_g17.001
```

The case directory has successfully been created. Change to the case directory and set up the case.

```
cd /nobackup/bjohns28/cases/FHIST.cesm2_2_0.ne30_g17.001
./case.setup
```

The `case.setup` script scaffolds out the case directory, creating the `Buildconf` and `CaseDocs` directories that you can customize. These instructions use the default configurations and continue on to compiling the model. On machines such as pleaides that don't throttle CPU usage on the pfe nodes, the `case.build` command can be invoked directly.

```
./case.build
```

---

**Note:** On Cheyenne, however, CPU intensive activities are killed on the login nodes, you will need to use a build wrapper to build the model on a shared compute node and specify a project code. Again, replace `PXXXXXXXX` with your project code.

```
qcmd -q share -l select=1 -A PXXXXXXXX -- ./case.build
```

---

The model build should progress for several minutes. If it compiles properly, a success message should be printed.

```
[...]
Time spent not building: 20.459729 sec
Time spent building: 719.937638 sec
MODEL BUILD HAS FINISHED SUCCESSFULLY
```

The model is actually built and run in a user's scratch space.

```
/nobackup/bjohns28/FHIST.cesm2_2_0.ne30_g17.001/bld/cesm.exe
```

## 5.4 Submitting a job

To submit a job, change to the case directory and use the `case.submit` script. The `-M begin,end` option sends the user an email when the job starts and stops running.

When the case is built, its default configuration is to run for five model days. This setting can be changed to run for a single model day using `./xmlchange STOP_N=1`.

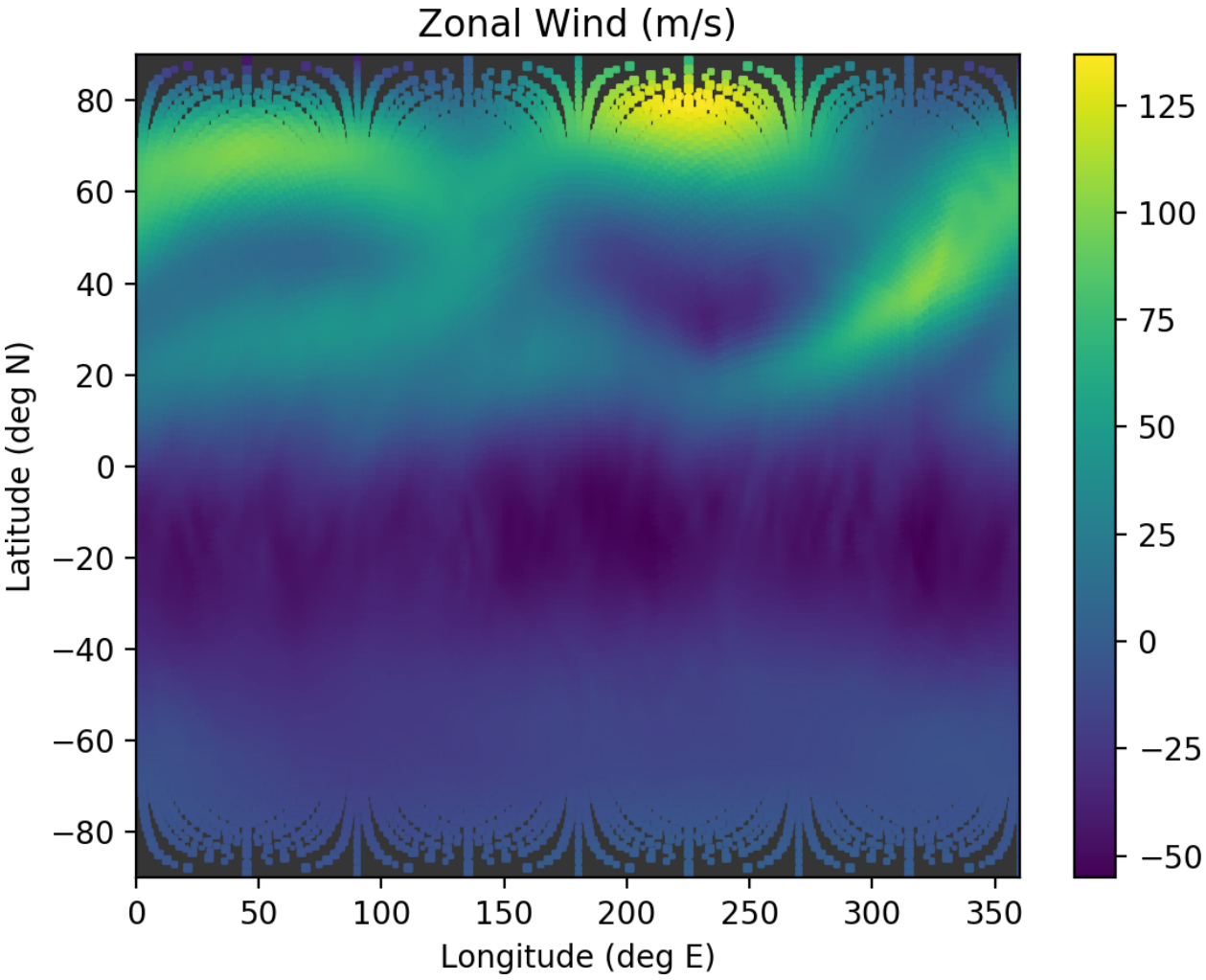
```
cd /nobackup/bjohns28/cases/FHIST.cesm2_2_0.ne30_g17.001
./xmlchange STOP_N=1
./case.submit -M begin,end
```

## 5.5 Restart file

After the job completes, restart files are written to the run directory which is also in scratch space. These restart files are written for both active and data components. The CAM restart file contains a `cam.r` substring. By default, the `FHIST` case begins on January 1st, 1979. Thus, the restart file will be for January 2nd, 1979.

```
/nobackup/bjohns28/FHIST.cesm2_2_0.ne30_g17.001/run/FHIST.cesm2_2_0.ne30_g17.001.cam.r.
↪1979-01-02-00000.nc
```

The fields in the restart file can be plotted using various langauges such as MATLAB or Python's matplotlib, as seen here.




## 5.6 References

CHAPTER

# SIX

# GRID CODE

CAM's dyn_grid.F90 module is responsible for creating a grid object for CAM. There are four different versions of this module:

```
cd /nobackup/bjohns28/CESM/components/cam/src/dynamics
find . -name "dyn_grid.F90"
./se/dyn_grid.F90
./fv/dyn_grid.F90
./eul/dyn_grid.F90
./fv3/dyn_grid.F90
```

Each of these modules depends on other CAM modules, obviously, but the intersection set of modules that all four of the dyn_grid.F90 is small and manageable.

According to the comments at the top of the source code, the module has two primary responsibilites:

- Provide the physics/dynamics coupler (in module phys_grid) with data for the physics grid on the dynamics decomposition.
- Create CAM grid objects that are used by the I/O functionality to read data from an unstructured grid format to the dynamics data structures, and to write from the dynamics data structures to unstructured grid format. The global column ordering for the unstructured grid is determined by the SE dycore.

The goal is to extract the se/dyn_grid.F90 source code so that it can be compiled without having to build the rest of the CESM code and provide a grid object for Aether.

```
cd /nobackup/bjohns28/CESM/components/cam/src/
grep -Rl dyn_grid ./
[...]
./dynamics/se/dyn_comp.F90
./dynamics/se/interp_mod.F90
./dynamics/se/restart_dynamics.F90
./dynamics/se/gravity_waves_sources.F90
./dynamics/se/dp_coupling.F90
./dynamics/se/stepon.F90
[...]
```

CHAPTER

SEVEN

# CAM-SE

## 7.1 Overview

This page documents the attempt to understand how CAM is compiled within the CIME framework.

## 7.2 First attempt

Compile a CAM-SE executable and examine the build log.

From examining the build log here, it appears that CAM is compiled as a static library file. Library files contain a set of subprograms that are compiled into a single binary library file.

Static libraries are bound to an executable before execution. Static libraries have the suffix "a" which denotes "archive."

Dynamic libraries can be bound to an executable at runtime. Dynamic libraries have the suffix "so" which denotes "shared object."

### 7.2.1 Build log from the first attempt

```
/glade/scratch/johnsonb/FHIST.cesm2_2_0.ne30_g17.002/bld/atm.bldlog.220611-170051.gz
```

Altering the gmake command for `libatm.a` to what it would be for a new case (with case suffix `.003`) results in:

```
gmake complib -j 8 MODEL=cam COMPLIB=/glade/scratch/johnsonb/FHIST.cesm2_2_0.ne30_g17.
↪003/bld/lib/libatm.a -f
/glade/work/johnsonb/cesm_runs/FHIST.cesm2_2_0.ne30_g17.003/Tools/Makefile
CIME_MODEL=cesm  SMP=FALSE CASEROOT="/glade/work/johnsonb/cesm_runs/FHIST.cesm2_2_0.ne30_
↪g17.003"
CASETOOLS="/glade/work/johnsonb/cesm_runs/FHIST.cesm2_2_0.ne30_g17.003/Tools"
CIMEROOT="/glade/work/johnsonb/cesm2_2_0/cime" COMP_INTERFACE="mct" COMPILER="intel"
DEBUG="FALSE" EXEROOT="/glade/scratch/johnsonb/FHIST.cesm2_2_0.ne30_g17.003/bld"
INCROOT="/glade/scratch/johnsonb/FHIST.cesm2_2_0.ne30_g17.003/bld/lib/include"
LIBROOT="/glade/scratch/johnsonb/FHIST.cesm2_2_0.ne30_g17.003/bld/lib"
MACH="cheyenne" MPILIB="mpt" NINST_VALUE="c1a1l1i1o1r1g1w1i1e1" OS="LINUX"
PIO_VERSION="1" SHAREDLIBROOT="/glade/scratch/johnsonb/FHIST.cesm2_2_0.ne30_g17.003/bld"
SMP_PRESENT="FALSE" USE_ESMF_LIB="FALSE" USE_MOAB="FALSE" CAM_CONFIG_OPTS="-physcam6"
COMP_LND="clm" COMPARE_TO_NUOPC="FALSE" CISM_USE_TRILINOS="FALSE" USE_TRILINOS="FALSE"
USE_ALBANY="FALSE" USE_PETSC="FALSE"   USER_CPPDEFS='    -DPLON=1 -DPLAT=1 -DNUM_COMP_
↪INST_ATM=1
```

(continues on next page)

(continued from previous page)

```
-DNUM_COMP_INST_LND=1 -DNUM_COMP_INST_OCN=1 -DNUM_COMP_INST_ICE=1 -DNUM_COMP_INST_GLC=1
-DNUM_COMP_INST_ROF=1 -DNUM_COMP_INST_WAV=1 -DNUM_COMP_INST_IAC=1 -DNUM_COMP_INST_ESP=1
-DCAM  -D_WK_GRAD -DNP=4 -DHAVE_F2003_PTR_BND_REMAP -D_MPI -DPLEV=32 -DPCNST=33
-DPCOLS=16 -DPSUBCOLS=1 -DN_RAD_CNST=30 -DPTRM=1 -DPTRN=1 -DPTRK=1 -DSPMD -DMODAL_AERO
-DMODAL_AERO_4MODE  -DCLUBB_SGS -DCLUBB_CAM -DNO_LAPACK_ISNAN -DCLUBB_REAL_TYPE=dp'
```

When trying to run the `gmake` command for a case that is set up but not built, `$SOURCES` and `$BASENAMES` aren't set.

```
cd /glade/scratch/johnsonb/FHIST.cesm2_2_0.ne30_g17.002
grep -Rl SOURCES ./
[ ... ]
# Returns many things
grep -Rl BASENAMES ./
# Returns nothing...
```

## 7.3 Second attempt

This attempt will build a CESM case using `./case.build` after editing the Makefile to echo the values of environmental variables.

```
./create_newcase --case /glade/work/johnsonb/cesm_runs/FHIST.cesm2_2_0.ne30_g17.003 --
↪compset FHIST --res ne30_g17 --mach cheyenne --project P86850054 --run-unsupported
```

What actually builds `libatm.a`?

```
cd /glade/work/johnsonb/cesm_runs/FHIST.cesm2_2_0.ne30_g17.003
grep -Rl libatm ./
./Tools/Makefile
```

I edited the Makefile to echo $(SOURCES), $(BASENAMES), $(OBJS), $(INCS):

```
801 #-------------------------------------------------------------------------------
802 # Build & include dependency files
803 #-------------------------------------------------------------------------------
[ ... ]
829
830 Filepath:
831     @echo "SOURCES=$(SOURCES)"
832     @echo "BASENAMES=$(BASENAMES)"
833     @echo "OBJS=$(OBJS)"
834     @echo "INCS=$(INCS)"
835     @echo "$(VPATH)" > $@
```

After making the above edits, build the case:

```
qcmd -q share -l select=1 -A $DARES_PROJECT -- ./case.build
[ ... ]
Building cesm from /glade/work/johnsonb/cesm2_2_0/cime/src/drivers/mct/cime_config/
↪buildexe with output to /glade/scratch/johnsonb/FHIST.cesm2_2_0.ne30_g17.003/bld/cesm.
↪bldlog.220707-143352
Time spent not building: 52.000454 sec
```

(continues on next page)

(continued from previous page)

```
Time spent building: 1138.138125 sec
MODEL BUILD HAS FINISHED SUCCESSFULLY
```

Try to find where `$(BASENAMES)` was echoed:

```
cd /glade/scratch/johnsonb/FHIST.cesm2_2_0.ne30_g17.003
grep -Rl BASENAMES ./
# Still returns nothing ...
```

### 7.3.1 Resulting object and library files

All of the CAM object files are compiled here:

```
cd /glade/scratch/johnsonb/FHIST.cesm2_2_0.ne30_g17.003/bld/atm/obj
ls *dyn_grid*
dyn_grid.mod  dyn_grid.o  dyn_grid.optrpt
```

The actual library files are compiled here:

```
cd /glade/scratch/johnsonb/FHIST.cesm2_2_0.ne30_g17.003/bld/lib
ls
include  libatm.a  libesp.a  libglc.a  libiac.a  libice.a  libocn.a  librof.a  libwav.a
```

Note that `libatm.a` is a 53MB file.

CHAPTER

EIGHT

# CSLAM

## 8.1 Compatible grids

There are four grid resolutions in CAM that interface with CSLAM:

| Resolution | Description |
|---|---|
| ne30pg3_ne30pg3_mg17 | Approximately 1 degree CAM-SE-CSLAM |
| ne30pg2_ne30pg2_mg17 | Approximately 1 degree CAM-SE-CSLAM with 1.5 degree physics grid |
| ne120pg3_ne120pg3_mt13 | Approximately 1/4 degree CAM-SE-CSLAM |
| ne120pg2_ne120pg2_mt12 | Approximately 1/4 degree CAM-SE-CSLAM with 3/8 degree physics grid |

## 8.2 Location within CESM

```
cd $CESMROOT
grep -Rl CSLAM ./
```

- `./components/cam/tools/topo_tool/cube_to_target/reconstruct.F90`
  - Provides functions for performing conservative interpolation between cubed sphere and lat lon grids.
- `./components/cam/src/dynamics/se/dyn_comp.F90`
  - CAM interfaces to the SE Dynamical Core
- `./components/cam/src/dynamics/se/interp_mod.F90`
  - Module containing subroutines and functions for interpolation.
- `./components/cam/src/dynamics/se/dycore/global_norms_mod.F90`
  - Module for computing global integrals and CFL conditions
- `./components/cam/src/dynamics/se/dycore/prim_advance_mod.F90`
  - Contains logic to advance the model a single timestep
- `./components/cam/src/dynamics/se/dycore/dimensions_mod.F90`
  - This is a non-monophyletic module that contains node specification and other things such as, “scaling of viscosity in sponge layer.”
- `./components/cam/src/dynamics/se/dycore/fvm_mod.F90`
  - FVM_MOD File for the fvm project
- `./components/cam/src/dynamics/se/dycore/prim_driver_mod.F90`

  - Primary driver mod?

- `./components/cam/src/dynamics/se/dycore/fvm_mapping.F90`
  - Two things in this module:
    1. pg2->pg3 mapping as discussed in Herrington et al., 2019a[1] . The pg3 grid divides each GLL grid cell into 3x3 control volumes, while the pg2 grid divides each GLL grid cell into 2x2 control volumes. Herrington et al., 2019a claim that, "the effective resolution of the model is not degraded through the use of a coarser-resolution physics grid. Since the physics makes up about half the computational cost of the conventional CAM-SE-CSLAM configuration, the coarser physics grid may allow for significant cost savings with little to no downside."
    2. pg3->GLL and GLL->pg3 mapping, (Herrington et al., 2019b[2] )
- `./components/cam/src/dynamics/se/dycore/fvm_consistent_se_cslam.F90`
  -
- ./components/cam/src/dynamics/se/dycore/prim_advection_mod.F90
- ./components/cam/src/dynamics/se/dycore/prim_state_mod.F90
- ./components/cam/src/dynamics/se/dycore/hybrid_mod.F90
- ./components/cam/src/dynamics/se/dycore/fvm_analytic_mod.F90
- ./components/cam/src/dynamics/se/dycore/fvm_control_volume_mod.F90
- ./components/cam/src/dynamics/se/restart_dynamics.F90
- ./components/cam/src/dynamics/se/dyn_grid.F90
- ./components/cam/src/dynamics/se/dp_mapping.F90
- ./ChangeLog

## 8.3 The meaning of the physics grid

When the documentation talks about the "physics" grid when CAM-SE is being used, it is referring to the finite volume method grid upon which CSLAM is being executed:

```
vim ./components/cam/doc/ChangeLog
[...]
components/cam/src/dynamics/se/dyn_comp.F90
[...]
the loop that sets analytic ICs for constituents directly on the
physics grid has been removed.  Instead all constituents are initially
set on the GLL grid, then mapped to the physics grid when CSLAM is used.
[...]
```

[1] Herrington, A. R., P. H. Lauritzen, K. A. Reed, S. Goldhaber, and B. E. Eaton, 2019a: Exploring a Lower-Resolution Physics Grid in CAM-SE-CSLAM. Journal of Advances in Modeling Earth Systems, 11, 1894–1916, https://doi.org/10.1029/2019MS001684.

[2] Herrington, A. R., P. H. Lauritzen, M. A. Taylor, S. Goldhaber, B. E. Eaton, J. T. Bacmeister, K. A. Reed, and P. A. Ullrich, 2019b: Physics–Dynamics Coupling with Element-Based High-Order Galerkin Methods: Quasi-Equal-Area Physics Grid. Monthly Weather Review, 147, 69–84, https://doi.org/10.1175/MWR-D-18-0136.1.

## 8.4 Running the physics grid at lower resolution

Herrington et al., 2019a[Page 24, 1] show that the physics grid can be run at lower resolution on the pg2 grid (which subdivides the cube sphere into 2x2 control volumes) with negligible ill-effects when compared to running at a higher resolution on the pg3 grid (which subdivides the cube sphere into 3x3 control volumes).

## 8.5 References

CHAPTER

NINE

# TERMINOLOGY

## 9.1 CSLAM

A semi-Lagrangian, finite volume advection scheme known as the Conservative Semi-Lagrangian Multitracer (CSLAM; Lauritzen et al., 2017[1] ).

Note that when implemented in WACCM, CSLAM is four times faster than the CAM-SE (Lauritzen, 2019[2] ).

## 9.2 FVM

If CSLAM is used for advection, it uses a finite volume method (FVM) grid and the results are then coupled to the cube-sphere grid.

## 9.3 GLL

In the SE dynamical core, the grid is known as the GLL grid because its columns are located at the Gauss-Lobatto-Legendre quadrature points.

## 9.4 References

[1] Lauritzen, P. H., M. A. Taylor, J. Overfelt, P. A. Ullrich, R. D. Nair, S. Goldhaber, and R. Kelly, 2017: CAM-SE–CSLAM: Consistent Coupling of a Conservative Semi-Lagrangian Finite-Volume Method with Spectral Element Dynamics. *Monthly Weather Review*, **145**, 833–855, https://doi.org/10.1175/MWR-D-16-0258.1.

[2] Lauritzen, P. H., 2019: Dynamical cores across scales in CESM. 2019 CESM Workshop. https://www.cesm.ucar.edu/events/workshops/ws.2019/presentations/cross/lauritzen.pdf

CHAPTER

TEN

# PARALLEL I/O

The Parallel I/O library enables applications to read and write netCDF files from a large number of processors.

CESM uses it to handle its reading and writing of netCDF files. Within the Common Infrastructure for Modeling the Earth (CIME) it is structured as an external.

```
cd $CESMROOT
grep -Rl PIO_LIBDIR ./
[ ... ]
./cime/scripts/Tools/Makefile
./cime/src/build_scripts/buildlib.pio
```

## 10.1 Makefile

A literal makefile used by CIME to build components.

```
731 CMAKE_OPTS += -D CMAKE_Fortran_FLAGS:STRING="$(FFLAGS) $(EXTRA_PIO_FPPDEFS)
↪$(INCLDIR)" \
732           -D CMAKE_C_FLAGS:STRING="$(CFLAGS) $(EXTRA_PIO_CPPDEFS) $(INCLDIR)" \
733           -D CMAKE_CXX_FLAGS:STRING="$(CXXFLAGS) $(EXTRA_PIO_CPPDEFS) $(INCLDIR)" \
734           -D CMAKE_VERBOSE_MAKEFILE:BOOL=ON \
735           -D GPTL_PATH:STRING=$(INSTALL_SHAREDPATH) \
736           -D PIO_ENABLE_TESTS:BOOL=OFF \
737           -D PIO_USE_MALLOC:BOOL=ON \
738           -D USER_CMAKE_MODULE_PATH:LIST="$(CIMEROOT)/src/CMake;$(CIMEROOT)/src/
↪externals/pio2/cmake" \
[ ... ]
777 # CMake doesn't seem to like it when you define compilers via -D
778 # CMAKE_C_COMPILER, etc., when you rerun cmake with an existing
779 # cache. So doing this via environment variables instead.
780 ifndef CMAKE_ENV_VARS
781   CMAKE_ENV_VARS :=
782 endif
783 CMAKE_ENV_VARS += CC=$(CC) \
784           CXX=$(CXX) \
785           FC=$(FC) \
786           LDFLAGS="$(LDFLAGS)"
[ ... ]
797 $(PIO_LIBDIR)/Makefile:
```

(continues on next page)

(continued from previous page)

```
798     cd $(PIO_LIBDIR); \
799     $(CMAKE_ENV_VARS) cmake $(CMAKE_OPTS) $(PIO_SRC_DIR)
```

## 10.2 buildlib.pio

A python script that may be capable of building pio.

CHAPTER

# ELEVEN

# OVERVIEW

**Important:** The CAM Reference Manual is more comprehensive than the current CESM2.X documentation. It even says, "This manual is intended for anyone who plans to get their hands dirty modifying CAM code."

For example, this CESM1.2 Physics Driver page in the reference manual doesn't seem to have an analog in the CESM2.X documentation.

## 11.1 Nomenclature

CAM distinguishes between *Dynamical cores* and *Physics packages*. State variables are updated first by the dynamics and then by the physics. Both the dynamical cores and the physics packages are implemented in a modular fashion and plug into the model infrastructure using an interface.

### 11.1.1 Dynamical cores

Dynamical cores are numerical methods implemented on specific model grids that simulate the large scale atmospheric flow. CAM supports several dynamical cores including the:

- EUL (Eulerian spectral-transform)
- SLD (semi-Lagrangian spectral-transform)
- FV (finite-volume)
- SE (spectral-elements)

### 11.1.2 Physics packages

Physics packages implement parameterized physical processes within a single column of the model grid. This functionality is often called a *vertical solver* within the space weather community.

Kalnay et al. (1989)[1] describe eleven rules to ensure physics package interoperability. Packages should only be responsible for performing a calculation upon either a single column or a limited section of the model's state. Other aspects of model integration such as communication, parallelization, input and output are handled by the support infrastructure within the model.

[1] Kalnay, E., M. Kanamitsu, J. Pfaendtner, J. Sela, M. Suarez, J. Stackpole, J. Tuccillo, L. Umscheid, and D. Williamson., 1989: Rules for Interchange of Physical Parameterizations. Bulletin of the American Meteorological Society, 70, 620–622, https://journals.ametsoc.org/view/journals/bams/70/6/1520-0477_1989_070_0620_rfiopp_2_0_co_2.xml.

These standards allow for modularity and were developed in response to the difficulty of inserting and comparing physics schemes into a given model.

## 11.2 Coding standards

CAM doesn't have a strict set of coding standards because much of its source code was contributed by the community. However, the UCAR wiki has a draft of coding standards for CAM. The document nodes that most of the code is preprocessed with Fortran preprocessor, `fpp`, which is not officially a part of the Fortran language standard.

## 11.3 Directory structure

This collection contains pages corresponding to each of the subdirectories in `cesm/components/cam/src/` that describes what each of the subdirectories contains.

- *advection*
- *atmos_phys*
- *chemistry*
- *control*
- *cpl*
- *dynamics*
- *ionosphere*
- *Overview*
- *physics*
- *unit_drivers*
- *utils*

## 11.4 References

CHAPTER

# TWELVE

# ADVECTION

Contains a single subdirectory `slt`, which stands for "Semi-Lagrangian Transport advection" that has many files in it containing subroutines.

- `bandij.F90` contains a single suroutine: "Calculate longitude and latitude indices that identify the intervals on the extended grid that contain the departure points."
- `basdy.F90` Compute weights for the calculation of derivative estimates at the two center points of the four point stencil for each interval in the unequally spaced latitude grid. Estimates are from differentiating a Lagrange cubic polynomial through the four point stencil.
- `basdz.F90` Compute weights for the calculation of derivative estimates at two center points of the four point stencil for each interval in the unequally spaced vertical grid (as defined by the array sig).
- `basiy.F90` Compute weights used in Lagrange cubic polynomial interpolation in the central interval of a four point stencil. Done for each interval in the unequally spaced latitude grid.
- `difcor.F90` Add correction term to t and q horizontal diffusions and determine the implied heating rate due to momentum diffusion.
- `engy_tdif.F90` Calculate contribution of current latitude to del-T integral.
- `engy_te.F90` Calculate contribution of current latitude to total energy.
- `extx.F90` Copy data to the longitude extensions of the extended array.
- `extys.F90` Fill latitude extensions of a scalar extended array and copy data to the longitude extensions of the extended array.
- `extyv.F90` Fill latitude extensions of a vector component extended array.
- `flxint.F90` Calculate contribution of current latitude to energy flux integral.
- `grdxy.F90` Define the "extended" grid used in the semi-Lagrangian transport scheme.
- `hadvtest.h` Looks like a short file that just has references to three functions: `usave`, `vsave`, `pssave` in it.
- `hordif1.F90` Horizontal diffusion of z,d,t,q.
- `kdpfnd.F90` Determine vertical departure point indices that point into a grid containing the full or half sigma levels.
- `lcbas.F90` Evaluate the partial Lagrangian cubic basis functions for the grid points rather than grid values.
- `lcdbas.F90` Calculate weights used to evaluate derivative estimates at the inner grid points of a four point stencil based on Lagrange cubic polynomial through four unequally spaced points.
- `omcalc.F90` Calculate vertical pressure velocity (omga = dp/dt).
- `pdelb0.F90` Compute the pressure intervals between the interfaces for the "B" portion of the hybrid grid only.

- `phcs.F90` Compute associated Legendre functions of the first kind of order m and degree n, and the associated derivatives for arg x1.
- `plevs0.F90` Define the pressures of the interfaces and midpoints from the coordinate definitions and the surface pressure.
- `qmassa.F90` Calculate contribution of current latitude to mass of constituents being advected by slt.
- `qmassd.F90` Compute comtribution of current latitude to global integral of q2*|q2 - q1|*eta.
- `reordp.F90` Renormalize associated Legendre polynomials and their derivatives.
- `scm0.F90` Apply SCM0 limiter to derivative estimates.
- `xqmass.F90` Compute comtribution of current latitude to global integrals necessary to compute the fixer for the non-water constituents.

CHAPTER

# THIRTEEN

# ATMOS_PHYS

This subdirectory is curiously not present in other versions of the repository.

- `LICENSE.txt`
- `README.md`
- `kessler` The Kessler warm rain scheme was first included to support the Dynamical Core.
  - `kessler.F90` Implements the Kessler (1969) microphysics parameterization as described by Soong and Ogura (1973) and Klemp and Wilhelmson (1978).
  - `kessler.meta`
  - `kessler_update.F90`
  - `kessler_update.meta`
- `suite_cam6.xml`
- `suite_cam6_silhs.xml`
- `suite_kessler.xml`
- `utilities`
  - `geopotential_t.F90` Compute the geopotential height (above the surface) at the midpoints and interfaces using the input temperatures and pressures.
  - `geopotential_t.meta`
  - `state_converters.F90` Contains various converters such as a conversion between temperature and potential temperature, dry pressure to dry air density, wet and dry, etc.
  - `state_converters.meta`

CHAPTER

# FOURTEEN

# CHEMISTRY

**Note:** I'm skipping extensive documentation of this subdirectory since it isn't immediately relevant to the project. My hunch is that `pp*` stands for physics package, but I'm not sure if that's true.

Contains many subdirectories for computing chemistry:

- `aerosol`
- `bulk_aero`
- `modal_aero`
- `mozart`
- `pp_none`
- `pp_terminator`
- `pp_trop_mam3`
- `pp_trop_mam4`
- `pp_trop_mam7`
- `pp_trop_mozart`
- `pp_trop_strat_mam4_ts2`
- `pp_trop_strat_mam4_vbs`
- `pp_trop_strat_mam4_vbsext`
- `pp_waccm_ma`
- `pp_waccm_ma_mam4`
- `pp_waccm_ma_sulfur`
- `pp_waccm_mad`
- `pp_waccm_mad_mam4`
- `pp_waccm_sc`
- `pp_waccm_sc_mam4`
- `pp_waccm_tsmlt_mam4`
- `utils`

CHAPTER

# FIFTEEN

# CONTROL

- cam_comp.F90
- cam_control_mod.F90
- cam_history.F90
- cam_history_buffers.F90
- cam_history_support.F90
- cam_initfiles.F90
- cam_instance.F90
- cam_logfile.F90
- cam_restart.F90
- cam_snapshot.F90
- camsrfexch.F90
- filenames.F90
- history_defaults.F90
- history_scam.F90
- ncdio_atm.F90
- runtime_opts.F90
- sat_hist.F90
- scamMod.F90

CHAPTER

# SIXTEEN

# CPL

Has two subdirectories:

**Note:** These source code files seem quite important. For example, the `atm_comp_mct.F90` file contains a subroutine named `atm_init_mct`.

- `mct`
  - `atm_comp_mct.F90` Contains a subroutine named `atm_init_mct`.
  - `atm_import_export.F90` Contains `atm_export`, which copies from component arrays into chunk array data structure. Rearrange data from chunk structure into lat-lon buffer and subsequently create attribute vector.
  - `cam_cpl_indices.F90` Contains `cam_cpl_indices_set` which queries booleans to determine whether certain fileds are passed by the coupler.
- `nuopc`
  - `atm_comp_nuopc.F90` Contains the NUOPC cap for CAM.
  - `atm_import_export.F90` Contains import and export fields.
  - `atm_shr_methods.F90` Contains shared methods.

CHAPTER

# SEVENTEEN

# DYNAMICS

Has all of the different dynamics schemes:

- `eul`
- `fv`
- `fv3`
- `se`
- `tests`

CHAPTER

# EIGHTEEN

# IONOSPHERE

This is where WACCMX seems to be stored:

- epotential_params.F90``
- ionosphere_interface.F90``
- waccmx
  - amie.F90
  - dpie_coupling.F90
  - edyn_esmf.F90
  - edyn_geogrid.F90
  - edyn_init.F90
  - edyn_maggrid.F90
  - edyn_mpi.F90
  - edyn_mud.F90
  - edyn_mudcom.F90
  - edyn_mudmod.F90
  - edyn_muh2cr.F90
  - edyn_params.F90
  - edyn_solve.F90
  - edynamo.F90
  - filter.F90
  - getapex.F90
  - heelis.F90
  - ionosphere_interface.F90
  - oplus.F90
  - rgrd_mod.F90
  - savefield_waccm.F90
  - wei05sc.F90

CHAPTER

# NINETEEN

# PHYSICS

- cam
  - physpkg.F90 provides an interface to the various physics packages included with cam. My hunch is that the references to pp in the cesm/components/cam/src/chemsitry stands for *physics package* but I'm just not quite sure whether this is true.
- camrt
- carma
- cosp2
- rrtmg
- simple
- spcam
- waccm
- waccmx

CHAPTER

# TWENTY

# UNIT_DRIVERS

- aur
- drv_input_data.F90
- offline_driver.F90
- rad
- stub

CHAPTER

# TWENTYONE

# UTILS

**Important:** `physconst.F90` contains many of the physical constants that are used throughout the model.

- `CMakeLists.txt`
- `bnddyi.F90`
- `buffer.F90.in`
- `cam_abortutils.F90`
- `cam_grid_support.F90`
- `cam_map_utils.F90`
- `cam_pio_utils.F90`
- `ccpp_kinds.F90`
- `coords_1d.F90`
- `datetime.F90`
- `dtypes.h`
- `error_messages.F90`
- `fft99.F90`
- `gauaw_mod.F90`
- `gmean_mod.F90`
- `hycoef.F90`
- `infnan.F90`
- `interpolate_data.F90`
- `intp_util.F90`
- `ioFileMod.F90`
- `linear_1d_operators.F90`
- `marsaglia.F90`
- `mpishorthand.F`
- `namelist_utils.F90`
- `orbit.F90`

- physconst.F90
- pilgrim
- quicksort.F90
- sgexx.F90
- spmd_utils.F90
- srchutil.F90
- srf_field_check.F90
- std_atm_profile.F90
- string_utils.F90
- time_manager.F90
- units.F90
- vrtmap.F90
- wrap_mpi.F90
- wrap_nf.F90
- xpavg_mod.F90

CHAPTER

# TWENTYTWO

# WACCM-X

The Community Atmosphere Model (CAM) extends throughout Earth's troposphere. The Whole Atmosphere Community Climate Model (WACCM) extends CAM further into the stratosphere and thermosphere. The Whole Atmosphere Community Climate Model - eXtended (WACCM-X) is an extension of WACCM that extends upward to ~500 km altitude and includes the ionosphere.

The overview paper of WACCM-X 2.0 (Liu et al. 2018[1] ) uses the finite volume dynamical core. The article also mentions that WACCM-X is based on CAM-4 physics and uses the `f19` atmospheric grid which has a horizontal resolution of 1.9° in latitude and 2.5° in longitude.

**Important:** It may not be possible to compile WACCM-X with the spectral element dycore. The safest thing to do would be to build a test case of the model with a finite volume dycore first.

[1] Liu, H.-L., and Coauthors, 2018: Development and Validation of the Whole Atmosphere Community Climate Model With Thermosphere and Ionosphere Extension (WACCM-X 2.0). *Journal of Advances in Modeling Earth Systems*, **10**, 381–402, https://doi.org/10.1002/2017MS001232.

CHAPTER

# TWENTYTHREE

# TUTORIAL

The WACCM-X tutorial demonstrates how to build a WACCM-X case.

```
cd /glade/work/johnsonb/cesm2_2_0/cime/scripts
./create_newcase --res f19_f19 --compset FXHIST --case /glade/work/johnsonb/cases/f.e20.
↪FXHIST.f19_f19.001 --mach cheyenne --project $DARES_PROJECT --run-unsupported
cd /glade/work/johnsonb/cases/f.e20.FXHIST.f19_f19.001
```

**Note:** Liu et al. (2018) note that the time steps for WACCM-X and CAM are significantly different. For example, CAM's time step for the `f19_f19` is 30 minutes, while it is 5 minutes for WACCM-X. Lauritzen et al. (2017)[2] note that the timestep for the roughly ~2° spectral element grid, `ne16np4`, is also 30 minutes.

[2] Lauritzen, P. H., and Coauthors, 2018: NCAR Release of CAM-SE in CESM2.0: A Reformulation of the Spectral Element Dynamical Core in Dry-Mass Vertical Coordinates With Comprehensive Treatment of Condensates and Energy. *Journal of Advances in Modeling Earth Systems*, **10**, 1537–1570, https://doi.org/10.1029/2017MS001257.

CHAPTER

# TWENTYFOUR

# FIRST ATTEMPT AT BUILDING FHISTX FOR THE NE16 GRID

```
cd /glade/work/johnsonb/cesm2_2_0/cime/scripts
./create_newcase --res ne16_g17 --compset FXHIST --case /glade/work/johnsonb/cases/f.e20.
↪FXHIST.ne16_g17.001 --mach cheyenne --project $DARES_PROJECT --run-unsupported
cd /glade/work/johnsonb/cases/f.e20.FXHIST.ne16_g17.001
./case.setup
./case.build
```

This results in the following error:

**Error:** ERROR: Command: '/glade/work/johnsonb/cesm2_2_0/components/cam/bld/configure -s -fc_type intel -dyn se -hgrid ne16np4 -cpl mct -usr_src /glade/work/johnsonb/cases/f.e20.FXHIST.ne16_g17.001/SourceMods/src.cam -spmd -nosmp -ocn docn -phys cam6 -waccmx -ionosphere wxie -chem waccm_ma_mam4' failed with error 'ERROR: Ionosphere is only available for FV dycore' from dir '/glade/work/johnsonb/cases/f.e20.FXHIST.ne16_g17.001/Buildconf/camconf'

This limitation in capability is also reflected in ACOM's geospace roadmap.

CHAPTER

TWENTYFIVE

# SECOND ATTEMPT AT BUILDING FHISTX FOR THE NE16 GRID

This CAM pull request suggests that the limitation in the ionosphere was fixed for a more-recent version of CAM for CESM2.3. Checkout a newer release of CESM and try again.

```
cd /glade/work/johnsonb
git clone https://github.com/ESCOMP/CESM cesm2_3_0
cd cesm2_3_0
git checkout cesm2_3_beta01
./manage_externals/checkout_externals
[ ... ]
cd cime/scripts
./create_newcase --res ne16_g17 --compset FXHIST --case /glade/work/johnsonb/cases/f.e20.
↪FXHIST.ne16_g17.002 --mach cheyenne --project $DARES_PROJECT --run-unsupported
cd /glade/work/johnsonb/cases/f.e20.FXHIST.ne16_g17.002
./case.setup
```

**Error:** `ERROR: Ionosphere is only available for FV dycore`

Try again to checkout a newer tag.

```
cd /glade/work/johnsonb
rm -rf cesm2_3_0
git clone https://github.com/ESCOMP/CESM cesm2_3_0
cd cesm2_3_0
git checkout cesm2_3_beta09
./manage_externals/checkout_externals
[ ... ]
cd cime/scripts
./create_newcase --res ne16_g17 --compset FXHIST --case /glade/work/johnsonb/cases/f.e20.
↪FXHIST.ne16_g17.003 --mach cheyenne --project $DARES_PROJECT --run-unsupported
```

**Error:** `SyntaxError:  invalid syntax`

Try again by checking out a slightly older tag.

```
git clone https://github.com/ESCOMP/CESM cesm2_3_0
cd cesm2_3_0
git checkout cesm2_3_beta09
```

(continues on next page)

(continued from previous page)

```
./manage_externals/checkout_externals
[ ... ]
cd cime/scripts
./create_newcase --res ne16_g17 --compset FXHIST --case /glade/work/johnsonb/cases/f.e20.
↪FXHIST.ne16_g17.004 --mach cheyenne --project $DARES_PROJECT --run-unsupported
ERROR: Python 3, minor version 6 is required, you have 3.4
source activate py37
cd /glade/work/johnsonb/cases/f.e20.FXHIST.ne16_g17.004
./case.setup
./case.build
```

**Error:** ERROR: Command /glade/work/johnsonb/cesm2_3_0/components/clm/bld/build-namelist failed rc=255 out= err=ERROR : CLM build-namelist::CLMBuildNamelist::add_default() : No default value found for flanduse_timeseries. Are defaults provided for this resolution and land mask?

Well this is progress.

Doing a triage of which beta releases of cesm2_3_0 provide the most plausible path toward compilation.

CHAPTER

# TWENTYSIX

# FAILED ATTEMPTS UNTIL SUCCESS WITH THE NE30 GRID

This might be simple to fix. According to this CGD BB post, It could merely be that there is a missing timeseries file that CLM needs.

Create a stock FHIST case and see how this is specified.

```
cd /glade/work/johnsonb/cesm2_1_3/cime/scripts
export CASEROOT='/glade/work/johnsonb/cases/f.e213.FHIST.f09_g17.001'
./create_newcase --res f09_g17 --compset FHIST --case $CASEROOT --mach cheyenne --
↪project $DARES_PROJECT --run-unsupported
cd $CASEROOT
./case.setup
./preview_namelists
grep -Rl flanduse_timeseries ./
./Buildconf/clmconf/lnd_in
./Buildconf/clm.input_data_list
./CaseDocs/lnd_in
```

There is no lnd_in file for the ne16_g17 cases. I attempted to set up a case with the ne16_g17 grid and the FHIST compset (instead of FXHIST) and ran into the same error. However, it was possible to build the namelist for a case with the ne30_g17 grid and the FHIST compset.

```
cd /glade/work/johnsonb/cesm2_2_0/cime/scripts
export CASEROOT='/glade/work/johnsonb/cases/f.e220.FHIST.ne30_g17.001'
./create_newcase --res ne30_g17 --compset FHIST --case $CASEROOT --mach cheyenne --
↪project $DARES_PROJECT --run-unsupported
cd $CASEROOT
./case.setup
./preview_namelists
grep -Rl flanduse_timeseries ./
./Buildconf/clmconf/lnd_in
./Buildconf/clm.input_data_list
./CaseDocs/lnd_in
```

**Important:** The key here to realize is that most of the spectral element dycore work is done on the ne30 grid (approximately 1° horizontal resolution) while most of the WACCM-X work is done on the f19 grid (approximately 2° horizontal resolution and the finite volume analog of the ne16 spectral element grid). The question now is: can a case be built using the ne30_g17 grid and the FXHIST compset?

```
cd /glade/work/johnsonb/git/cesm2_3_0_beta09/cime/scripts
export CASEROOT='/glade/work/johnsonb/cases/f.e230b9.FXHIST.ne30_g17.001'
./create_newcase --res ne30_g17 --compset FXHIST --case $CASEROOT --mach cheyenne --
↪project $DARES_PROJECT --run-unsupported
cd $CASEROOT
./case.setup
./preview_namelists
grep -Rl flanduse_timeseries ./
./Buildconf/clmconf/lnd_in
./Buildconf/clm.input_data_list
./CaseDocs/lnd_in
./case.build
MODEL BUILD HAS FINISHED SUCCESSFULLY
```

**Note:** Hooray! However, I don't know where the preprocessed source files are contained.

There is a list of the source files in `/glade/scratch/johnsonb/f.e230b9.FXHIST.ne30_g17.001/bld/atm/obj/Srcfiles` but I don't know where the files actually are.

For example one of the files is `cam_history.F90`:

```
cd /glade/scratch/johnsonb/f.e230b9.FXHIST.ne30_g17.001
find . -name cam_history.F90
[ Returns nothing ]
cd /glade/work/johnsonb/cases/f.e230b9.FXHIST.ne30_g17.001
find . -name cam_history.F90
[ Returns nothing ]
```

CHAPTER

TWENTYSEVEN

# GETTING THE COMPILER TO SAVE THE POST-PREPROCESSED FILES

While there is a directory for `build_scripts` in `cime/src/build_scripts`, each of the scripts in that subdirectory import `CIME.buildlib` which is in `cime/scripts/lib/CIME/buildlib.py`.

## 27.1 buildlib.py

This python script contains three functions: `parse_input`, `build_cime_component_lib` and `run_gmake`. The last function actually invokes `gmake` to build a component executable. The tractable path forward seems to be to see if we can get these functions to preprocess the files and save them.

Editing `buildlib.py` to print the commands within `run_gmake`:

```
vim /glade/work/johnsonb/git/cesm2_3_0_beta09/cime/scripts/lib/CIME/buildlib.py
102     print('BKJ inserted this: ', cmd)
103     stat, out, err = run_cmd(cmd, combine_output=True)
```

CHAPTER

# TWENTYEIGHT

# APPENDING -E TO THE COMPILER FLAGS

Helen's suggestion at the 2022-08-30 standup was to append `-E` as a compiler flag in `/glade/work/johnsonb/git/cesm2_3_0_beta09/cime/config/cesm/machines/config_compilers.xml`.

```
903 <compiler MACH="cheyenne" COMPILER="intel">
904   <CFLAGS>
905     <append> -qopt-report -xCORE_AVX2 -no-fma -E</append>
906   </CFLAGS>
907   <FFLAGS>
908     <append> -qopt-report -xCORE_AVX2 -no-fma -E</append>
909   </FFLAGS>
[ ... ]
917 </compiler>
```

Tried this both with `cesm2_3_0_beta09` and `cesm2_3_0_beta02` and it doesn't work:

**Error:** ERROR: /glade/work/johnsonb/git/cesm2_3_0_beta02/cime/src/build_scripts/buildlib.gptl FAILED, cat /glade/scratch/johnsonb/f.e230b2.FXHIST.ne30_g17.001/bld/gptl.bldlog.220831-140809

CHAPTER

TWENTYNINE

# APPENDING -SAVE-TEMPS TO THE COMPILER FLAGS

This page within the iFort guide suggests that the -save-temps compile flag will save the preprocessed files.

```
vim /glade/work/johnsonb/git/cesm2_3_0_beta09/cime/config/cesm/machines/config_compilers.
↪xml

903 <compiler MACH="cheyenne" COMPILER="intel">
904   <CFLAGS>
905     <append> -qopt-report -xCORE_AVX2 -no-fma -save-temps</append>
906   </CFLAGS>
907   <FFLAGS>
908     <append> -qopt-report -xCORE_AVX2 -no-fma -save-temps</append>
909   </FFLAGS>
[ ... ]
917 </compiler>
```

Then build the case:

```
cd /glade/work/johnsonb/git/cesm2_3_0_beta09/cime/scripts
export CASEROOT='/glade/work/johnsonb/cases/f.e230b9.FXHIST.ne30_g17.004'
./create_newcase --res ne30_g17 --compset FXHIST --case $CASEROOT --mach cheyenne --
↪project $DARES_PROJECT --run-unsupported
cd $CASEROOT
./case.setup
./case.build
[ ... ]
MODEL BUILD HAS FINISHED SUCCESSFULLY
cd /glade/scratch/johnsonb/f.e230b9.FXHIST.ne30_g17.004/bld/atm/obj
ls *.i90
# This shows all of the post-preprocessed files.
```

CHAPTER

# THIRTY

# REFERENCES

CHAPTER

# THIRTYONE

# WACCM-X

The Community Atmosphere Model (CAM) extends throughout Earth's troposphere. The Whole Atmosphere Community Climate Model (WACCM) extends CAM further into the stratosphere and thermosphere. The Whole Atmosphere Community Climate Model - eXtended (WACCM-X) is an extension of WACCM that extends upward to ~500 km altitude and includes the ionosphere.

The overview paper of WACCM-X 2.0 (Liu et al. 2018[1] ) uses the finite volume dynamical core. The article also mentions that WACCM-X is based on CAM-4 physics and uses the `f19` atmospheric grid which has a horizontal resolution of 1.9° in latitude and 2.5° in longitude.

**Important:** It may not be possible to compile WACCM-X with the spectral element dycore. The safest thing to do would be to build a test case of the model with a finite volume dycore first.

[1] Liu, H.-L., and Coauthors, 2018: Development and Validation of the Whole Atmosphere Community Climate Model With Thermosphere and Ionosphere Extension (WACCM-X 2.0). *Journal of Advances in Modeling Earth Systems*, **10**, 381–402, https://doi.org/10.1002/2017MS001232.

CHAPTER

# THIRTYTWO

# TUTORIAL

The WACCM-X tutorial demonstrates how to build a WACCM-X case.

```
cd /glade/work/johnsonb/cesm2_2_0/cime/scripts
./create_newcase --res f19_f19 --compset FXHIST --case /glade/work/johnsonb/cases/f.e20.
↪FXHIST.f19_f19.001 --mach cheyenne --project $DARES_PROJECT --run-unsupported
cd /glade/work/johnsonb/cases/f.e20.FXHIST.f19_f19.001
```

**Note:** Liu et al. (2018) note that the time steps for WACCM-X and CAM are significantly different. For example, CAM's time step for the `f19_f19` is 30 minutes, while it is 5 minutes for WACCM-X. Lauritzen et al. (2017)[2] note that the timestep for the roughly ~2° spectral element grid, `ne16np4`, is also 30 minutes.

[2] Lauritzen, P. H., and Coauthors, 2018: NCAR Release of CAM-SE in CESM2.0: A Reformulation of the Spectral Element Dynamical Core in Dry-Mass Vertical Coordinates With Comprehensive Treatment of Condensates and Energy. *Journal of Advances in Modeling Earth Systems*, **10**, 1537–1570, https://doi.org/10.1029/2017MS001257.

CHAPTER

# THIRTYTHREE

# FIRST ATTEMPT AT BUILDING FHISTX FOR THE NE16 GRID

```
cd /glade/work/johnsonb/cesm2_2_0/cime/scripts
./create_newcase --res ne16_g17 --compset FXHIST --case /glade/work/johnsonb/cases/f.e20.
↪FXHIST.ne16_g17.001 --mach cheyenne --project $DARES_PROJECT --run-unsupported
cd /glade/work/johnsonb/cases/f.e20.FXHIST.ne16_g17.001
./case.setup
./case.build
```

This results in the following error:

**Error:** ERROR: Command: '/glade/work/johnsonb/cesm2_2_0/components/cam/bld/configure -s -fc_type intel -dyn se -hgrid ne16np4 -cpl mct -usr_src /glade/work/johnsonb/cases/f.e20.FXHIST.ne16_g17.001/SourceMods/src.cam -spmd -nosmp -ocn docn -phys cam6 -waccmx -ionosphere wxie -chem waccm_ma_mam4' failed with error 'ERROR: Ionosphere is only available for FV dycore' from dir '/glade/work/johnsonb/cases/f.e20.FXHIST.ne16_g17.001/Buildconf/camconf'

This limitation in capability is also reflected in ACOM's geospace roadmap.

CHAPTER

THIRTYFOUR

# SECOND ATTEMPT AT BUILDING FHISTX FOR THE NE16 GRID

This CAM pull request suggests that the limitation in the ionosphere was fixed for a more-recent version of CAM for CESM2.3. Checkout a newer release of CESM and try again.

```
cd /glade/work/johnsonb
git clone https://github.com/ESCOMP/CESM cesm2_3_0
cd cesm2_3_0
git checkout cesm2_3_beta01
./manage_externals/checkout_externals
[ ... ]
cd cime/scripts
./create_newcase --res ne16_g17 --compset FXHIST --case /glade/work/johnsonb/cases/f.e20.
↪FXHIST.ne16_g17.002 --mach cheyenne --project $DARES_PROJECT --run-unsupported
cd /glade/work/johnsonb/cases/f.e20.FXHIST.ne16_g17.002
./case.setup
```

**Error:** `ERROR: Ionosphere is only available for FV dycore`

Try again to checkout a newer tag.

```
cd /glade/work/johnsonb
rm -rf cesm2_3_0
git clone https://github.com/ESCOMP/CESM cesm2_3_0
cd cesm2_3_0
git checkout cesm2_3_beta09
./manage_externals/checkout_externals
[ ... ]
cd cime/scripts
./create_newcase --res ne16_g17 --compset FXHIST --case /glade/work/johnsonb/cases/f.e20.
↪FXHIST.ne16_g17.003 --mach cheyenne --project $DARES_PROJECT --run-unsupported
```

**Error:** `SyntaxError: invalid syntax`

Try again by checking out a slightly older tag.

```
git clone https://github.com/ESCOMP/CESM cesm2_3_0
cd cesm2_3_0
git checkout cesm2_3_beta09
```

(continues on next page)

(continued from previous page)

```
./manage_externals/checkout_externals
[ ... ]
cd cime/scripts
./create_newcase --res ne16_g17 --compset FXHIST --case /glade/work/johnsonb/cases/f.e20.
↪FXHIST.ne16_g17.004 --mach cheyenne --project $DARES_PROJECT --run-unsupported
ERROR: Python 3, minor version 6 is required, you have 3.4
source activate py37
cd /glade/work/johnsonb/cases/f.e20.FXHIST.ne16_g17.004
./case.setup
./case.build
```

**Error:** ERROR: Command /glade/work/johnsonb/cesm2_3_0/components/clm/bld/build-namelist failed rc=255 out= err=ERROR : CLM build-namelist::CLMBuildNamelist::add_default() : No default value found for flanduse_timeseries. Are defaults provided for this resolution and land mask?

Well this is progress.

Doing a triage of which beta releases of cesm2_3_0 provide the most plausible path toward compilation.

CHAPTER

# THIRTYFIVE

# FAILED ATTEMPTS UNTIL SUCCESS WITH THE NE30 GRID

This might be simple to fix. According to this CGD BB post, It could merely be that there is a missing timeseries file that CLM needs.

Create a stock FHIST case and see how this is specified.

```
cd /glade/work/johnsonb/cesm2_1_3/cime/scripts
export CASEROOT='/glade/work/johnsonb/cases/f.e213.FHIST.f09_g17.001'
./create_newcase --res f09_g17 --compset FHIST --case $CASEROOT --mach cheyenne --
↪project $DARES_PROJECT --run-unsupported
cd $CASEROOT
./case.setup
./preview_namelists
grep -Rl flanduse_timeseries ./
./Buildconf/clmconf/lnd_in
./Buildconf/clm.input_data_list
./CaseDocs/lnd_in
```

There is no lnd_in file for the ne16_g17 cases. I attempted to set up a case with the ne16_g17 grid and the FHIST compset (instead of FXHIST) and ran into the same error. However, it was possible to build the namelist for a case with the ne30_g17 grid and the FHIST compset.

```
cd /glade/work/johnsonb/cesm2_2_0/cime/scripts
export CASEROOT='/glade/work/johnsonb/cases/f.e220.FHIST.ne30_g17.001'
./create_newcase --res ne30_g17 --compset FHIST --case $CASEROOT --mach cheyenne --
↪project $DARES_PROJECT --run-unsupported
cd $CASEROOT
./case.setup
./preview_namelists
grep -Rl flanduse_timeseries ./
./Buildconf/clmconf/lnd_in
./Buildconf/clm.input_data_list
./CaseDocs/lnd_in
```

**Important:** The key here to realize is that most of the spectral element dycore work is done on the ne30 grid (approximately 1° horizontal resolution) while most of the WACCM-X work is done on the f19 grid (approximately 2° horizontal resolution and the finite volume analog of the ne16 spectral element grid). The question now is: can a case be built using the ne30_g17 grid and the FXHIST compset?

```
cd /glade/work/johnsonb/git/cesm2_3_0_beta09/cime/scripts
export CASEROOT='/glade/work/johnsonb/cases/f.e230b9.FXHIST.ne30_g17.001'
./create_newcase --res ne30_g17 --compset FXHIST --case $CASEROOT --mach cheyenne --
↪project $DARES_PROJECT --run-unsupported
cd $CASEROOT
./case.setup
./preview_namelists
grep -Rl flanduse_timeseries ./
./Buildconf/clmconf/lnd_in
./Buildconf/clm.input_data_list
./CaseDocs/lnd_in
./case.build
MODEL BUILD HAS FINISHED SUCCESSFULLY
```

**Note:** Hooray! However, I don't know where the preprocessed source files are contained.

There is a list of the source files in `/glade/scratch/johnsonb/f.e230b9.FXHIST.ne30_g17.001/bld/atm/obj/Srcfiles` but I don't know where the files actually are.

For example one of the files is `cam_history.F90`:

```
cd /glade/scratch/johnsonb/f.e230b9.FXHIST.ne30_g17.001
find . -name cam_history.F90
[ Returns nothing ]
cd /glade/work/johnsonb/cases/f.e230b9.FXHIST.ne30_g17.001
find . -name cam_history.F90
[ Returns nothing ]
```

CHAPTER

# THIRTYSIX

# GETTING THE COMPILER TO SAVE THE POST-PREPROCESSED FILES

While there is a directory for `build_scripts` in `cime/src/build_scripts`, each of the scripts in that subdirectory import `CIME.buildlib` which is in `cime/scripts/lib/CIME/buildlib.py`.

## 36.1 buildlib.py

This python script contains three functions: `parse_input`, `build_cime_component_lib` and `run_gmake`. The last function actually invokes `gmake` to build a component executable. The tractable path forward seems to be to see if we can get these functions to preprocess the files and save them.

Editing `buildlib.py` to print the commands within `run_gmake`:

```
vim /glade/work/johnsonb/git/cesm2_3_0_beta09/cime/scripts/lib/CIME/buildlib.py
102     print('BKJ inserted this: ', cmd)
103     stat, out, err = run_cmd(cmd, combine_output=True)
```

CHAPTER

# THIRTYSEVEN

# APPENDING -E TO THE COMPILER FLAGS

Helen's suggestion at the 2022-08-30 standup was to append `-E` as a compiler flag in `/glade/work/johnsonb/git/cesm2_3_0_beta09/cime/config/cesm/machines/config_compilers.xml`.

```
903 <compiler MACH="cheyenne" COMPILER="intel">
904   <CFLAGS>
905     <append> -qopt-report -xCORE_AVX2 -no-fma -E</append>
906   </CFLAGS>
907   <FFLAGS>
908     <append> -qopt-report -xCORE_AVX2 -no-fma -E</append>
909   </FFLAGS>
[ ... ]
917 </compiler>
```

Tried this both with `cesm2_3_0_beta09` and `cesm2_3_0_beta02` and it doesn't work:

**Error:** ERROR: /glade/work/johnsonb/git/cesm2_3_0_beta02/cime/src/build_scripts/buildlib.gptl FAILED, cat /glade/scratch/johnsonb/f.e230b2.FXHIST.ne30_g17.001/bld/gptl.bldlog.220831-140809

CHAPTER

THIRTYEIGHT

# APPENDING -SAVE-TEMPS TO THE COMPILER FLAGS

This page within the iFort guide suggests that the -save-temps compile flag will save the preprocessed files.

```
vim /glade/work/johnsonb/git/cesm2_3_0_beta09/cime/config/cesm/machines/config_compilers.
↪xml

903 <compiler MACH="cheyenne" COMPILER="intel">
904   <CFLAGS>
905     <append> -qopt-report -xCORE_AVX2 -no-fma -save-temps</append>
906   </CFLAGS>
907   <FFLAGS>
908     <append> -qopt-report -xCORE_AVX2 -no-fma -save-temps</append>
909   </FFLAGS>
[ ... ]
917 </compiler>
```

Then build the case:

```
cd /glade/work/johnsonb/git/cesm2_3_0_beta09/cime/scripts
export CASEROOT='/glade/work/johnsonb/cases/f.e230b9.FXHIST.ne30_g17.004'
./create_newcase --res ne30_g17 --compset FXHIST --case $CASEROOT --mach cheyenne --
↪project $DARES_PROJECT --run-unsupported
cd $CASEROOT
./case.setup
./case.build
[ ... ]
MODEL BUILD HAS FINISHED SUCCESSFULLY
cd /glade/scratch/johnsonb/f.e230b9.FXHIST.ne30_g17.004/bld/atm/obj
ls *.i90
# This shows all of the post-preprocessed files.
```

CHAPTER

THIRTYNINE

# REFERENCES

CHAPTER

FORTY

# COMPILING WITH MKMF

mkmf is a tool written in perl and developed at GFDL that takes raw FORTRAN source code, maps out the dependencies and then creates a `Makefile` to enable compilation of the code.

`mkmf` already comes within the DART repository. It can be used in the following manner.

1. Run a script to stage the source code files that are intended to be compiled
2. Navigate to the DART `build_templates` where `mkmf` is located.
3. Export the location of the DART installation
4. Run mkmf
5. Use `gmake` to compile the code.

```
python ~/python_scripts/rename_i90_files.py
cd /glade/work/johnsonb/git/DART/build_templates/
export DART=/glade/work/johnsonb/git/DART
./mkmf pathnames /glade/scratch/johnsonb/mkmf_target
make
```

## 40.1 Errors encountered

### 40.1.1 seq_timemgr_mod.F90

This file gets preprocessed by setting the *-E* flag in *config_compilers.xml* but when it gets compiled by `gmake`, various errors get thrown.

**Error:** seq_timemgr_mod.F90(1988): error #6634: The shape matching rules of actual arguments and dummy arguments have been violated. call ESMF_ClockGetAlarmList(EClock, alarmListFlag, &

The unpreprocessed file has two versions of this `ESMF_ClockGetAlarmList` function that are selected by the preprocessor:

```
vim /glade/work/johnsonb/git/cesm2_3_0_beta09/cime/src/drivers/mct/shr/seq_timemgr_mod.
↪F90
[ ... ]
1829 #ifdef USE_ESMF_LIB
1830     allocate(EAlarm_list(AlarmCount))
1831     call ESMF_ClockGetAlarmList(EClock, alarmListFlag=ESMF_ALARMLIST_ALL, &
```

(continues on next page)

(continued from previous page)

```
1832          alarmList=EAlarm_list, alarmCount=AlarmCount, rc=rc)
1833 #else
1834     call ESMF_ClockGetAlarmList(EClock, EAlarm_list, rc=rc)
1835 #endif
```

### 40.1.2 ESMF_FIELD

**Error:** /glade/work/johnsonb/git/cesm2_3_0_beta09/components/cam/src/ionosphere/waccmx/utils_mod.F90(5): error #6580: Name in only-list does not exist or is not accessible. [ESMF_FIELD] use esmf ,only: ESMF_FIELD ————————————-^ compilation aborted for /glade/scratch/johnsonb/mkmf_target/utils_mod.f90 (code 1)

Comment out the use statement in the processed file:

```
vim /glade/scratch/johnsonb/mkmf_target/utils_mod.f90
! use esmf           ,only: ESMF_FIELD
```

CHAPTER

# FORTYONE

# LINKING TO ESMF

When CIME compiles CESM, it loads the esmf_libs module to link to during compilation.

Here is an excerpt from an example `software_environment.txt` from a successful CESM build:

```
vim /glade/work/johnsonb/cases/f.e230b9.FXHIST.ne30_g17.009/software_environment.txt
Currently Loaded Modules:
1) ncarenv/1.3    3) intel/19.1.1      5) mkl/2020.0.1              7) mpt/2.22         ␣
↪ 9) pnetcdf/1.12.2
2) cmake/3.18.2   4) esmf_libs/8.2.0   6) esmf-8.1.1-ncdfio-mpt-O   8) netcdf-mpi/4.7.4 ␣
↪10) ncarcompilers/0.5.0
[ ... ]
ESMF_LIBDIR=/glade/p/cesmdata/cseg/PROGS/esmf/8.1.1/mpt/2.22/intel/19.1.1/lib/libO/Linux.
↪intel.64.mpt.default
```

So when trying to compile this source code outside of CIME, I load the same libraries:

```
module purge
module load ncarenv/1.3 cmake intel/19.1.1 esmf_libs mkl
module use /glade/p/cesmdata/cseg/PROGS/modulefiles/esmfpkgs/intel/19.1.1/
module load esmf-8.1.1-ncdfio-mpt-O mpt/2.22 netcdf-mpi/4.7.4 pnetcdf/1.12.2␣
↪ncarcompilers/0.5.0
```

In my `mkmf.template` which is here:

```
/glade/work/johnsonb/git/DART/build_templates/mkmf.template
```

I reference the same library and also include an additional directory which contains the module interface files, `-I$(ESMF)/mod/modO/Linux.intel.64.mpt.default`:

```
ESMF = /glade/p/cesmdata/cseg/PROGS/esmf/8.1.1/mpt/2.22/intel/19.1.1
INCS = -I$(NETCDF)/include -I$(ESMF)/include -I$(ESMF)/mod/modO/Linux.intel.64.mpt.
↪default
LIBS = -L$(NETCDF)/lib -lnetcdff -lnetcdf -L$(ESMF)/lib/libO/Linux.intel.64.mpt.default -
↪lesmf
```

The file that throws a compile-time error is `edyn_esmf.f90`. There are a few precursor steps to running make:

```
python /glade/u/home/johnsonb/python_scripts/rename_i90_files.py
cd /glade/work/johnsonb/git/DART/build_templates/
export DART=/glade/work/johnsonb/git/DART
./mkmf pathnames /glade/scratch/johnsonb/mkmf_target
make edyn_esmf.o
```

CHAPTER

# FORTYTWO

# DART

The Data Assimilation Research Testbed (DART) implements many ensemble assimilation methodologies including the Ensemble Kalman Filter (Evensen, 2003)[1] and the Ensemble Adjustment Kalman Filter (Anderson, 2001).[2]

Forward operators required to assimilate observations perform line-of-sight integration, volume integration, interpolation, and other common techniques.

These forward operators are optimized given the grid structures and the parallelization scheme.

## 42.1 References

[1] Evensen, G., 2003: The Ensemble Kalman Filter: theoretical formulation and practical implementation. *Ocean Dynamics*, **53**, 343–367, doi:10.1007/s10236-003-0036-9

[2] Anderson, J. L., 2001: An Ensemble Adjustment Kalman Filter for Data Assimilation. *Monthly Weather Review*, **129**, 2884-2903, doi:10.1175/1520-0493(2001)129<2884:AEAKFF>2.0.CO;2

CHAPTER

# FORTYTHREE

# FILTERING THEORY

Assume a dynamical system that is governed by a stochastic difference equation:

$$dx_t = f(x_t, t) + G(x_t, t)d\beta_t$$

for all times, $t \geq 0$. Observations occur at discrete times:

$$y_k = h(x_k, t_k) + \nu_k$$

where $k = 1, 2, ...$; and $t_{k+1} > t_k \geq t_0$.

The observational error is white in time and is Gaussian (this latter assumption is not essential).

$$\nu_k \rightarrow N(0, R_k)$$

The complete history of observations is:

$$Y_\tau = \{y_l; t_l \leq \tau\}$$

Our goal is to find the probability distribution for the state at time $t$.

$$p(x, t|Y_t)$$

The state between observation times is obtained from the difference equation. We need to update the state given new observations:

$$p(x, t_k|Y_{t_k}) = p(x, t_k|y_k, Y_{t_{k-1}})$$

We do so by applying Bayes' rule:

$$p(x, t_k|Y_{t_k}) = \frac{p(y_k|x_k, Y_{t_{k-1}})p(x, t_k|Y_{t_{k-1}})}{p(y_k, Y_{t_{k-1}})}$$

Since the error is white in time:

$$p(y_k|x_k, Y_{t_{k-1}}) = p(y_k|x_k)$$

We integrate the numerator to obtain a normalizing denominator:

$$p(y_k|x_k, Y_{t_{k-1}}) = \int p(y_k|x)p(x, t_k|Y_{t_{k-1}})dx$$

This allows us to update the probability after a new observation:

$$p(x, t_k|Y_{t_k}) = \frac{p(y_k|x)p(x, t_k|Y_{t_{k-1}})}{\int p(y_k|\xi)p(\xi, t_k|Y_{t_{k-1}})d\xi}$$

CHAPTER

FORTYFOUR

# MONTE CARLO ENSEMBLES

Uncertainty quantification is incorporated at a fundamental level in Aether by making Monte Carlo ensemble simulations standard. Users can specify a default range of acceptable parameters, coefficient, rates, and drivers.

Users can modify the ensemble input by specifying which parameters, coefficients, rates, and driver indices they would like to vary and how many ensemble members they would like to create.

Post processing codes store raw ensemble members, create mean states, standard deviations, and statistical analyses that provide confidence levels.

CHAPTER

FORTYFIVE

# DOCKER

In order to implement a GitHub Actions workflow in which Armadillo, nlohmann_json, NetCDF and their dependencies can be linked or included in the compiled model, Docker can be used to create an image on which these dependencies can be kept.

The NetCDF documentation suggests using a package management tool to install NetCDF.

> "The easiest way to get netCDF is through a package management program, such as rpm, yum, homebrew, macports, adept, and others."

This document describes how to use Conda to install NetCDF and Armadillo and how to create a Docker image that can be activated by GitHub Actions in order to run automated tests when pull requests are made to a specified branch.

## 45.1 Conda

The conda package manager can quickly install the dependencies needed by Aether.

If you don't have conda installed on your system, it merely requires downloading and running a shell script. For more information, see conda's installation guide.

If you would like to install the dependencies on your local machine without using a Docker image, the following commands create and activate a virtual environment named `aether-armadillo-json-netcdf` in which the `armadillo` and `nlohmann_json` header files and the `netcdf-cxx4` library are installed.

```
conda create --name netcdf-armadillo-json --channel conda-forge netcdf-cxx4 armadillo␣
↪nlohmann_json
conda activate netcdf-armadillo-json
```

## 45.2 Building a Docker image

Since conda was used to install the dependencies, it's necessary to activate conda in a Docker file in order to create the Docker image. This post by Itamar Turner-Trauring describes the difficulty of getting conda activate to run in the bash shell

The bash shell started by Docker isn't configured to activate a conda environment.

Building a Docker image

> can be linked into the source code, it is useful to

Aether's source code relies on a few dependencies in order to run properly:

- Armadillo

- NetCDF
- nlohmann JSON library

Aether manages these dependencies using git's submodule capability. Submodules are their own repositories that are contained within a larger repository. Submodules can nest within other submodules and their contents can be edited and updated.

## 45.3 Verify your version of git

In order to get started, you will need to verify which version of git you have installed because git's support for submodules has changed throughout its various releases. These instructions will work for releases equal to or newer than git 2.7. In order to verify which version of git you are using, type:

```
git --version
```

## 45.4 Adding a submodule to a repository

```
mkdir external
git submodule add https://github.com/nlohmann/json external/json
```

## 45.5 Modifying your git config to show the status of submodules

```
git config --global status.submoduleSummary true
```

## 45.6 Organization of the .gitmodules file

The submodules are defined in a dot file stored in the root directory of the repository, `.gitmodules`.

```
cat .gitsubmodules
[submodule "external/json"]
    path = external/json
    url = https://github.com/nlohmann/json
```

## 45.7 Cloning a repository and its submodules

```
git clone https://github.com/AetherModel/Aether.git
cd Aether
git submodule init
git submodule update
```

## 45.8 Removing a submodule from a repository

### 45.8.1 Temporary removal

```
git submodule deinit external/json
```

### 45.8.2 Permament removal

```
git submodule deinit external/json
git rm external/json
git commit -m "Removed json submodule"
```

CHAPTER

# FORTYSIX

# VERIFICATION TESTS

Verification tests determine whether a model does what it is designed to do. Aether is verified by constructing unit tests and integration tests for the model's functions during the development process, implementing a structured walk-through policy for new model modules, comparing both intermediate and final simulation results to analytic results, and using a range of input combinations that are both typical and atypical.

There are many challenges in performing these different verification tests, especially for models of large systems. However, the benefits, which include but are not limited to catching bugs during software development and providing users with information on appropriate input ranges, are significant.

Unit tests are run on a predetermined schedule to determine whether mistakes have been made in committed code, with reports being sent to core developers.

One example of a commonly used unit test is the Sod shocktube, used to verify the performance of hydrodynamic solvers in computational fluid dynamics codes.

This particular problem has an analytical solution and is thus highly useful for verification of the numerical implementation. Variations of this problem can be used to test each dimension in the code separately. Similar higher dimensional versions of this test, such as the blast wave, can also be used for verification.

CHAPTER

# FORTYSEVEN

# CODING SCHOOL

The Aether *Core Team* runs a coding school that offers multiple levels of training on software and model development. The school is logically scaled in experience level as a function of time, so that users who know nothing about using models can attend the early days of the school, while more experienced users who would like to start contributing can attend later days of the school.

Student hackathons are also conducted to add new features to the model. Overlapping of experience levels allows users/contributors to interact and optimize training and development.

Sustainability of the coding school is promoted by encouraging students from previous years to act as assistants and then as teachers.

Wider community involvement is encouraged through the creation and dissemination of online training courses and videos.

These materials are developed by undergraduate and graduate students. They introduce relatively simple concepts to students and other new users, allowing them to rapidly climb the steep learning curve involved in using a model like Aether.

These tutorials are directed towards undergraduates and beginning graduate students with topics including logging into NASA and NSF computing resources for the first time, managing the queue system, checking out and configuring the model, submitting jobs, post processing, and making plots. These are simple tutorials, but the topics contained within are either not available or not easily discoverable.

CHAPTER

FORTYEIGHT

# CONTRIBUTORS' GUIDE

## 48.1 Contributing source code

Aether is an open-source model that welcomes your contributions. If you want to contribute to the project, you should be familiar with the standard GitHub fork and pull request workflow. These are the basic steps involved in the workflow.

1. Fork the repository
2. Clone your fork
3. Checkout a feature branch
4. Make changes, commit them, and push them to your fork
5. Make a pull request

## 48.2 GitHub instructional tutorials

Many universities provide their faculty, staff and students access to instructional materials from LinkedIn Learning. Two videos from this service are useful for learning the basics of git and GitHub:

- Git Essential Training
- Git Branches, Merges, and Remotes

## 48.3 Contributing documentation

Aether's documentation follows the Google developer documentation style guide. If you want to contribute documentation, you should read through the style guide and follow its guidelines as you write.

This documentation is output as HTML using the Sphinx documentation tool. The text is written using the reStructuredText specification. See the *reStructuredText Style Guide* for formatting instructions.

### 48.3.1 Adding pages to the documentation

Sphinx uses a table of contents tree, or `toctree`, directive to organize its contents. To add a new page, first create a `.rst` document in the repository and then add a reference to the document you created in one of the `toctree` lists in `index.rst`.

CHAPTER

# FORTYNINE

# CORE TEAM

While Aether is an open-source project, its development is lead by a core team of scientists.

## 49.1 Team members

**Aaron Ridley** has significant experience developing models and teaching students. He has worked on porting the TIEGCM to the Linux architecture and coupling it to a global MHD code. He rewrote the Assimilative Mapping of Ionospheric Electrodynamics technique, developed an ionospheric electrodynamics solver, helped to develop the Space Weather Modeling Framework, developed GITM, helped to develop a state-of-the-art orbit propagator for determining the probability of collisions, and is currently coupling GITM to the SAMI3 model of the ionosphere. He teaches engineering and science classes at every level. Prof. Ridley manages the effort to develop Aether and the supporting education program. He leads team meetings, helps to develop the architecture of Aether (including core model, UQ, DA, and OSSE support), leads the development and gathering of educational resources including the school lesson planning, and helps with community support.

**Jeffrey Anderson** is a senior scientist at the National Center for Atmospheric Research where he leads the Data Assimilation Research Section. He is the lead architect of Data Assimilation Research Testbed, including its application to large models on high-performance computing. He has extensive experience in developing ensemble data assimilation algorithms and applying them to earth system models and observations. Dr. Anderson has experience applying DART to upper atmosphere and space weather models including GITM, TIEGCM, Open-GGCM, and WACCM-X and to models that used the cubed-sphere grid like CAM-SE. He was the lead developer of an earlier version of the GFDL atmospheric prediction system and the original developer of the GFDL Flexible Modeling System, a software system for efficient model development.

**Jared Bell** is a planetary atmospheres modeler at GSFC. He has modified the Earth version of GITM to work at Titan, and has helped to develop the Mars version of GITM. He has upgraded the time stepping in GITM to be 4th order, the vertical boundary conditions to be 4th order, and the vertical solver from a Rusanov-type to the AUSM+-up solver. He is currently working on an oblated spheroid version of GITM, allowing the radius to the lower boundary to vary as a function of latitude, so that fast rotating planets, such as Jupiter and Saturn, can be simulated. His role on the Aether team is to support the model development, specifically the solvers and the grid system, and its application.

**Alex Glocer** has extensive experience developing and coupling models of the space environment. He is the primary developer of the Polar Wind Outflow Model having expanded it from a single field line code to a global code and coupling it to a global magnetosphere model. He further expanded PWOM to a combined fluid-kinetic model, and worked on the development of the multi- fluid MHD BATS-R-US magnetosphere model. He contributes to the development of ring current and radiation belt models and their coupling with the global magnetosphere. His role on the Aether team is to support the model development and application.

**Angeline Burrell** has extensive experience developing scientific programs in collaborative environments. She is actively working on model validation efforts at NRL, and is working to create a score card that can be used to track the global and regional improvement of ionospheric models. Her role will include supporting the model development by

performing code reviews, creating tools for model validation, validating model results against publicly available data, and providing community support.

**Meghan Burleigh** is an early career scientist and has experience developing ionospheric models. She created GEMINI-TIA, a local scale, multi-fluid model designed for the high-latitude ionosphere. In addition, she is working on incorporating 2-way coupling of GITM to the SWMF to facilitate self-consistent physics. She is developing a new course at UM titled "Programming Practices for Scientists", with Qusai Al Shidi, that focuses on teaching students good coding practices, including methods that promote collaboration and facilitate version control. Her role on the Aether team includes assisting with the development of the architecture of Aether, contributing to educational resources, teaching at the coding school, and providing community support.

**Qusai Al Shidi** is an early career scientist with experience in both space physics and computer science, having developed a solar chromosphere model from scratch. The model is a two-fluid collisional MHD model. Ionosphere and chromosphere MHD models are usually presented together since they share the same multi-fluid and collisional physics. He is currently working on studying the energy transfer of ICME's into storms by running multi-scale Space Weather Modeling Framework simulations of the Sun-Earth system, from solar wind to ionosphere. His role will include Aether development and developing software standards for Aether and its teaching.

**Ben Johnson** is an early career scientist who works with the NCAR software engineer and Jeff Anderson to develop DART interfaces to Aether that will support the science requirements of the project. He assists software engineer in implementing these interfaces, provide scientific expertise in the evaluation of both OSSE and real-data tests of DART/Aether, and lead the implementation of enhanced documentation and tutorial material for ensemble data assimilation.

CHAPTER

FIFTY

# RESTRUCTUREDTEXT STYLE GUIDE

This page contains example reStructuredText syntax including title headers, citations, links, code snippets, tables, and quotes.



## 50.1 Major Title

### 50.1.1 Minor Title

#### Micro Title

## 50.2 Link Examples

- *Working link* will take us to the paragraph that begins with, "This link target works."

Headers are also link targets by default. See, for example, how this makes a link to the *useful syntax* section even though the header is capitalized and this link is not.

Links to external websites, such as the University of Michigan, are easily accomodated using < > to contain the link and double underscores at the end of the link syntax.

### 50.2.1 Full Citations in Footnotes

The Lorenz (1963)[1] model is specified by a set of three ordinary differential equations. The Lorenz (1996)[2] model is more complex. The transition line below separates this paragraph from the two paragraphs below.

---

This *link target* works. It works because there is a new line separating the link target and the text. It is often the case when using reStructuredText that proper syntax includes a new line to delineate the parts of a particular structure.

## 50.3 Useful Syntax

### 50.3.1 Blockquotes

Long sections of quoted text can be included simply by indenting the paragraph. Blockquotes can also be nested. Here, we quote from Holton and Hakim (2013)[3]:

> A problem with the traditional form of the omega equation is that there exists significant cancellation between the two right-side terms. To expose this cancellation and develop two different forms of the omega equation, we need to expand the right side of (6.42). This involves taking the gradient of vector products, for which vector notation is not well suited.
>
> > Just as we use vector notation to simplify mathematical manipulations when scalar notation becomes cumbersome, it is often prudent to move to *indicial notation* for situations where vector notation becomes awkward.

### 50.3.2 Definition Lists

These lists can be used to introduce terms to the reader.

**Forward Operator**
: A routine that interpolates values from the model grid to the observation location.

**Observation Converter**
: A program that converts observations from other formats into obs_seq format.

### 50.3.3 Field Lists

Field lists can be used for describing properties of a model.

**Model**
: Aether

**Grid**
: Cube sphere

[1] Lorenz, Edward N. (1963) "Deterministic Nonperiodic Flow." *Journal of the Atmospheric Sciences* **20** (2): 130–141.
[2] Lorenz, Edward N. (1996) "Predictability – A problem partly solved." *Seminar on Predictability* **I**: ECMWF.
[3] Holton, James R. and Gregory J. Hakim (2013) *An Introduction to Dynamic Meteorology.* Fifth Edition, 552 pages. Academic Press, San Diego, USA.

## 50.4 Code Examples

reStructuredText recognizes C syntax and highlights it appropriately:

```c
#include <stdio.h>
int main() {
    printf("Hello, World!");
    return 0;
}
```

Syntax highlighting also works in other languages that we might use for scripting such as bash:

```bash
#!/bin/bash

for a in `seq 1 10`; do
    echo "$a/10 to Exit."
    sleep 1;
done

echo "We are done bashing"
```

or python:

```python
#!/usr/bin/env python

def save(obj):
    return (obj.__class__, obj.__dict__)

def load(cls, attributes):
    obj = cls.__new__(cls)
    obj.__dict__.update(attributes)
    return obj
```

## 50.5 Nested Lists and Inline Literals

Directories such as `Aether/src/bfield.cpp` or even commands such as `grep -Rl dipole ./` can be called out within a paragraph using what are known as "inline literals" – just wrap the desired text by two backticks.

1. Multiple commands can be stacked to instruct users to do several commands at once, even a list element:

   ```
   git clone https://github.com/AetherModel/Aether.git
   ```

   ```
   cd Aether
   ```

2. Here the list continues even after we include three lines of commands.
3. And we have a third list element.

Even more complicated list structures are possible by using spaces to indent the nested list to the same character column as the content of the outer list.

1. First element in outer list
2. Second element in outer list

    1. First element in nested list is indented by three spaces and separated from the outer list by a new line.

    2. Second element in nested list is also indented by three spaces.
3. Third element in outer list is not indented but is separated from the nested list by a new line.

## 50.6 Bullet List

- Bullet lists are easy to make
- Just make sure there is a new line before and after the list

## 50.7 Tables

Complex tables are straightforward to make. See here that the first row of table data after the table header has only one column instead of three.

| year | month/day of first,middle,last | obs_seq #### of first,middle,last |
|---|---|---|
| Include GPS when it becomes available? | | |
| 2006 | 1/ 1, 1/16, 1/31 | 2954 - 2969 - 2984 |
| 2006 | 2/ 1, 2/16, 2/28 | 2985 - 3000 - 3012 |
| 2006 | 3/ 1, 3/16, 3/31 | 3013 - 3028 - 3043 |
| 2006 | 4/ 1, 4/16, 4/30 | 3044 - 3059 - 3073 |
| 2006 | 5/ 1, 5/16, 5/31 | 3074 - 3089 - 3104 |
| 2006 | 6/ 1, 6/16, 6/30 | 3105 - 3120 - 3134 |
| 2006 | 7/ 1, 7/16, 7/31 | 3135 - 3150 - 3165 |
| 2006 | 8/ 1, 8/16, 8/31 | 3166 - 3181 - 3196 |
| 2006 | 9/ 1, 9/16, 9/30 | 3197 - 3212 - 3226 |
| 2006 | 10/ 1, 10/16, 10/31 | 3227 - 3242 - 3257 |
| 2006 | 11/ 1, 11/16, 11/30 | 3258 - 3273 - 3287 |
| 2006 | 12/ 1, 12/16, 12/31 | 3288 - 3303 - 3318 |

Table 1: Demonstration of simple table syntax.

| Right | Left | Center | Default |
|---|---|---|---|
| 12 | 12 | 12 | 12 |
| 123 | 123 | 123 | 123 |
| 1 | 1 | 1 | 1 |

## 50.8 Citations

Clicking on the number that denotes each citation links back to its original mention within the text.

CHAPTER

# FIFTYONE

# GIT SUBMODULES

Aether's source code relies on a few dependencies in order to run properly:

- Armadillo
- NetCDF
- nlohmann JSON library

Aether manages these dependencies using git's submodule capability. Submodules are their own repositories that are contained within a larger repository. Submodules can nest within other submodules and their contents can be edited and updated.

## 51.1 Verify your version of git

In order to get started, you will need to verify which version of git you have installed because git's support for submodules has changed throughout its various releases. These instructions will work for releases equal to or newer than git 2.7. In order to verify which version of git you are using, type:

```
git --version
```

## 51.2 Adding a submodule to a repository

```
mkdir external
git submodule add https://github.com/nlohmann/json external/json
```

## 51.3 Modifying your git config to show the status of submodules

```
git config --global status.submoduleSummary true
```

## 51.4 Organization of the .gitmodules file

The submodules are defined in a dot file stored in the root directory of the repository, `.gitmodules`.

```
cat .gitsubmodules
[submodule "external/json"]
    path = external/json
    url = https://github.com/nlohmann/json
```

## 51.5 Cloning a repository and its submodules

```
git clone https://github.com/AetherModel/Aether.git
cd Aether
git submodule init
git submodule update
```

## 51.6 Removing a submodule from a repository

### 51.6.1 Temporary removal

```
git submodule deinit external/json
```

### 51.6.2 Permament removal

```
git submodule deinit external/json
git rm external/json
git commit -m "Removed json submodule"
```

CHAPTER

# FIFTYTWO

# SPHINX AS A DOCUMENTATION TOOL

This is created using a documentation generation tool written in python known as Sphinx.

If you are a regular python user, and would like to use Sphinx, install it using the package management procedure with which you are most comfortable.

## 52.1 Installing Conda

If you are novice python user, here are instructions for installing Sphinx using the package manager known as Conda (short for Anaconda) on Mac OSX.

Open a terminal window and execute these three commands:

1. Download the installation script:

```
$ curl -O https://repo.anaconda.com/miniconda/Miniconda3-latest-MacOSX-x86_64.sh
```

2. Modify the execution permissions of the recently downloaded script:

```
$ chmod 755 ./Miniconda3-latest-MacOSX-x86_64.sh
```

3. Run the installation script and follow its instructions:

```
$ ./Miniconda3-latest-MacOSX-x86_64.sh
```

## 52.2 Downloading Sphinx

Now that Conda is installed, use it to download Sphinx. Close and reopen the terminal window to ensure conda is activated.

1. Invoke Conda and instruct it to install two packages, Sphinx and the "Read the Docs" CSS theme for Sphinx:

```
$ conda install sphinx sphinx-rtd-theme
```

## 52.3 Clone the Repository

You're ready to clone the repository:

```
$ git clone https://github.com/AetherModel/AetherDocumentation.git
$ cd AetherDocumentation
```

## 52.4 Edit Files and Remake the Documentation

The files that comprise the documentation are written in reStructuredText which offers a reasonably simple syntax for writing documentation while still accommodating the elements needed for technical writing such as equations, references, code snippets, tables, etc.

Find the reStructuredText documents contained in this repository:

```
find . -name "*.rst"
```

Edit them using your favorite text editor and then remake the documentation:

```
make clean
make html
```

View the remade documentation in your favorite web browser by opening the `./_build/html/index.html` using your graphical user interface or via the command line:

```
open ./_build/html/index.html
```