MDTF Getting Started Guide

Release 3.0 beta 1

Thomas Jackson (GFDL), Yi-Hung Kuo (UCLA), Dani Coleman (NCAR)

Jun 11, 2020

CHAPTER

ONE

GETTING STARTED

1.1 Overview

Welcome! In this section we'll describe what the MDTF diagnostics framework is, how it works, and how you can contribute your own diagnostic scripts.

1.1.1 Purpose

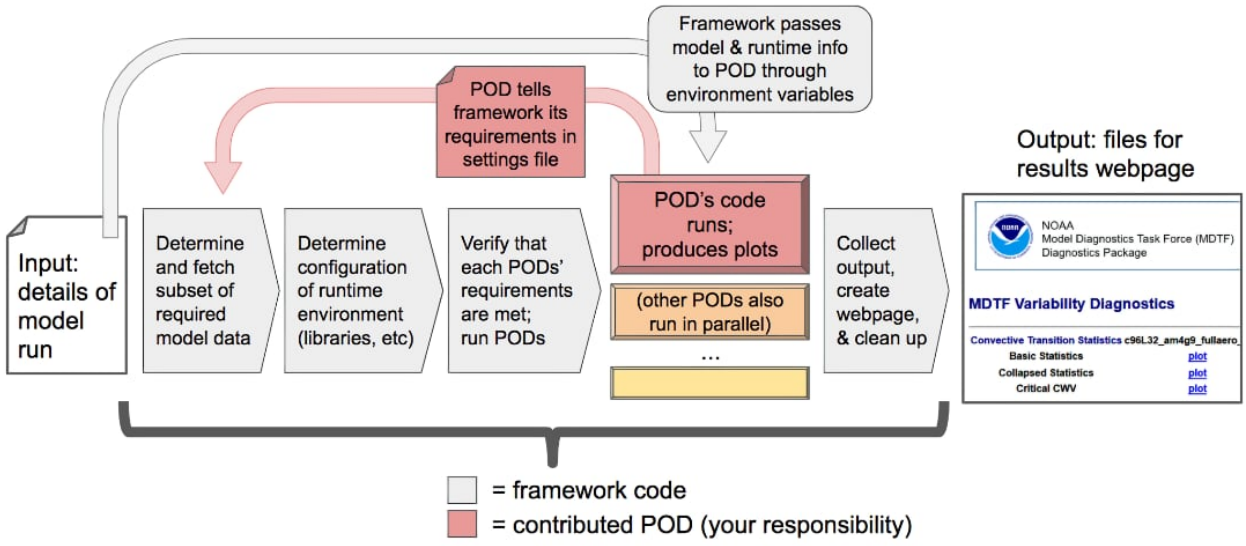
The scientific motivation and content behind the framework was described in E. D. Maloney et al. (2019): Process-Oriented Evaluation of Climate and Weather Forecasting Models. BAMS, 100 (9), 1665–1686, doi:10.1175/BAMS-D-18-0042.1[1].

Also see the section of this site devoted to documentation of individual diagnostics.

1.1.2 Framework operation

The design goal of the MDTF framework is to provide a portable and adaptable means to run process-oriented diagnostic scripts, abbreviated as PODs below. By "portability," we mean the ideal of "run once, run anywhere": the purpose of the framework is to automate retrieval of model data from different local or remote sources, and transform that data into a layout (field names, variable units, etc.) your script expects. This will empower your analysis to be run by a wider range of researchers on a wider range of models.

[1] https://doi.org/10.1175/BAMS-D-18-0042.1



The MDTF Diagnostic Framework consists of multiple Process-Oriented Diagnostic (POD) modules, each of which is developed by an individual research group. For clarity, the framework is the structure provided by the Model Diagnostics Task Force, and the PODs (or modules) are developed by individual groups (or developers). PODs are developed and run independently of each other. Each POD takes as input (1) requested variables from the model run, along with (2) any required observational or supporting data, performs an analysis, and produces (3) a set of figures which are presented to the user in a series of .html files. (We do not include or require a mechanism for publishing these webpages on the internet; html is merely used as a convenient way to present a multimedia report to the user.)

1.1.3 Getting started for users

The rest of the documentation in this section describes next steps for end users of the framework:

- We provide instructions on how to download and install (page 4) the framework and run it on some sample data.
- We describe the most common configuration options (page 8) for running the framework on your own model data.
- See also the list of command-line options.
- Known troubleshooting issues (page 10); also see the GitHub issue tracker[2].

1.1.4 Getting started for developers

As summarized in the figure above, the changes needed to convert an existing analysis script for use in the framework are:

- Provide a settings file which tells the framework what it needs to do: what languages and libraries your code need to run, and what model data your code takes as input.

[2] https://github.com/NOAA-GFDL/MDTF-diagnostics/issues

- Adapt your code to load data files from locations set in unix shell environment variables (we use this as a language-independent way for the framework to communicate information to the POD).
- Provide a template web page which links to, and briefly describes, the plots generated by the script.

Each of these are described in more detail in the developer-specific sections:

- We provide instructions on working with git for people who haven't used it before.
- Instructions and framework policies to keep in mind when developing your POD.
- Description of the settings file needed by the framework to process your POD's requirements.
- A more detailed walkthrough that elaborates on the flowchart above and describes the steps taken by the framework in order to run your POD.
- A checklist of items needed for submitting your POD for inclusion in the framework.
- A collection of links to relevant tutorials and resources.

1.2 Quickstart installation instructions

This document provides basic directions for downloading, installing and running a test of the Model Diagnostics Task Force (MDTF) Process-Oriented Diagnostics package using sample model data. The current MDTF package has been tested on UNIX/LINUX, Mac OS, and Windows Subsystem for Linux.

You will need to download a) the source code, b) digested observational data, and c) two sets of sample model data (Section 1.2.1). Afterwards, we describe how to install necessary Conda environments and languages (Section 1.2.2) and run the framework on the default test case (Section 1.2.4).

1.2.1 Download the package code and sample data for testing

Throughout this document, `%` indicates the UNIX/LINUX command line prompt and is followed by commands to be executed in a terminal in `fixed-width font`, and `$` indicates strings to be substituted, e.g., the string `$CODE_ROOT` below should be substituted by the actual path to the MDTF-diagnostics directory.

Obtaining the code

The official repo for the MDTF code is hosted at the GFDL GitHub account[3]. We recommend that end users download and test the latest official release[4].

To install the MDTF package on a local machine, create a directory named `mdtf`, and unzip the code downloaded from the release page[5] there. This will create a directory titled `MDTF-diagnostics-3.0-beta.1` containing the files listed on the GitHub page. Below we refer to this MDTF-diagnostics directory as `$CODE_ROOT`. It contains the following subdirectories:

- `diagnostics/`: directories containing source code of individual PODs.

[3] https://github.com/NOAA-GFDL/MDTF-diagnostics
[4] https://github.com/NOAA-GFDL/MDTF-diagnostics/releases/tag/v3.0-beta.1
[5] https://github.com/NOAA-GFDL/MDTF-diagnostics/releases/tag/v3.0-beta.1

- `doc/`: directory containing documentation (a local mirror of the documentation site).
- `src/`: source code of the framework itself.
- `tests/`: unit tests for the framework.

For advanced users interested in keeping more up-to-date on project development and contributing feedback, the `master` branch contains features that haven't yet been incorporated into an official release, which are less stable or thoroughly tested.

For POD developers, the `develop` branch is the "beta test" version of the framework. POD developers should begin work on this branch as described in ref-dev-git.

Obtaining supporting data

Supporting observational data and sample model data are available via anonymous FTP at ftp://ftp.cgd.ucar.edu/archive/mdtf. The observational data is required for the PODs' operation, while the sample model data is provided for default test/demonstration purposes. The files needed for package installation and default tests are:

- Digested observational data (159 Mb): MDTF_v2.1.a.20200410.obs_data.tar[6].
- NCAR-CESM-CAM sample data (12.3 Gb): model.QBOi.EXP1.AMIP.001.tar[7].
- NOAA-GFDL-CM4 sample data (4.8 Gb): model.GFDL.CM4.c96L32.am4g10r8.tar[8].

Users installing on Mac OS should use the Finder's Archive Utility instead of the command-line tar command to extract the files. Download these three files and extract the contents in the following hierarchy under the `mdtf` directory:

- `mdtf/inputdata/obs_data/...`
- `mdtf/inputdata/model/QBOi.EXP1.AMIP.001/...`
- `mdtf/inputdata/model/GFDL.CM4.c96L32.am4g10r8/...`

The default test case uses the QBOi.EXP1.AMIP.001 sample. The GFDL.CM4.c96L32.am4g10r8 sample is only for testing the MJO Propagation and Amplitude POD. Note that `mdtf` now contains both `MDTF-diagnostics` and `inputdata` directories.

1.2.2 Install the necessary programming languages and modules

The MDTF framework code is written in Python 2.7, but supports running PODs written in a variety of scripting languages and combinations of libraries. We use conda[9], a free, open-source package manager to install and manage these dependencies. Conda is one component of the Anaconda[10] python distribution, so having Anaconda is sufficient but not necessary.

[6] ftp://ftp.cgd.ucar.edu/archive/mdtf/MDTF_v2.1.a.20200410.obs_data.tar
[7] ftp://ftp.cgd.ucar.edu/archive/mdtf/model.QBOi.EXP1.AMIP.001.tar
[8] ftp://ftp.cgd.ucar.edu/archive/mdtf/model.GFDL.CM4.c96L32.am4g10r8.tar
[9] https://docs.conda.io/en/latest/
[10] https://www.anaconda.com/

For maximum portability and ease of installation, we recommend that all users manage dependencies through conda using the provided script, even if they have independent installations of the required languages. A complete installation of all dependencies will take roughly 5 Gb, less if you've already installed some of the dependencies through conda. The location of this installation can be changed with the `$CONDA_ENV_DIR` setting described below.

If these space requirements are prohibitive, we provide an alternate method of operation which makes no use of conda and relies on the user to install external dependencies, at the expense of portability. This is documented in a separate section.

Conda installation

The framework's environments will co-exist with an existing Anaconda or miniconda installation. Do not reinstall miniconda/Anaconda if it's already installed for the user who will be running the framework: the installer will break the existing installation (if it's not managed with, eg., environment modules.)

To determine if conda is installed, run `% conda --version` as the user who will be using the framework. The framework has been tested against versions of conda >= 4.7.5.

If you do not have a pre-existing Anaconda or miniconda installation on your system, we recommend using the miniconda2 (python 2.7) installer available here[11]. Any version of miniconda/Anaconda (2 or 3) released after June 2019 will work: the only differences are the modules that are pre-installed by default. Toward the end of the installation process, enter "yes" at "Do you wish the installer to initialize Miniconda2 by running conda init?" prompt. This will allow the installer to add the conda path to the user's shell login script (e.g., `~/.bashrc` or `~/.cshrc`).

Conda environment installation

Run `% conda info --base` as the user who will be using the framework to determine the location of your conda installation. This path will be referred to as `$CONDA_ROOT` below. After determining this path, run

```
% cd $CODE_ROOT
% ./src/conda/conda_env_setup.sh --all --conda_root $CONDA_ROOT
```

to install all needed environments. This takes ~10 min. The names of all framework-created environments begin with "_MDTF", so as not to conflict with any other environments that are defined.

By default, Conda will install program files within `$CONDA_ROOT` (the "active env location" listed by `% conda info`). To use a different location (for space reasons, or if you don't have write access), pass the desired directory as `$CONDA_ENV_DIR` below:

```
% ./src/conda/conda_env_setup.sh --all --conda_root $CONDA_ROOT --env_dir $CONDA_
↪ENV_DIR
```

After installing the framework-specific conda environments, you shouldn't manually alter them (i.e., never run `conda update` on them). To update the environments after updating the framework code, re-run the above commands.

[11] https://docs.conda.io/en/latest/miniconda.html

1.2.3 Configure package paths

Open `src/default_tests.jsonc` in an editor (we recommend working on a copy). This is a template/example of an input file you can use to define configuration options instead of re-typing them on the command line every time you run the framework.

- If you've installed the supporting data in the directory structure described in Obtaining supporting data (page 5), the existing values for `OBS_DATA_ROOT` and `MODEL_DATA_ROOT` will be correct. If you put the data in a different location, these values should be changed accordingly.
- `OUTPUT_DIR` should be set to the location you want the output files to be written to. The output of each run of the framework will be saved in a different subdirectory in this location.
- `conda_root` should be set to the value of `$CONDA_ROOT` you used above.
- If you specified a custom environment location with `$CONDA_ENV_DIR`, set `conda_env_root` to that value; otherwise, leave it blank.

1.2.4 Run the MDTF package with test settings

Location of the MDTF executable

Following Conda environment installation (page 6), the setup script will have created an executable at `$CODE_ROOT/mdtf` which sets the correct conda environment before running the framework. To test the installation, run

```
% cd $CODE_ROOT
% ./mdtf --help
```

This should print help text on the command-line options.

Run the framework on sample data

To run the framework on the CESM sample model data, run

```
% cd $CODE_ROOT
% ./mdtf -f src/default_tests.jsonc
```

If you edited a copy of `default_tests.jsonc`, pass that file instead. Run time may be 10-20 minutes, depending on your system.

The output files for this test case will be written to `$OUTPUT_DIR/QBOi.EXP1.AMIP.001_1977_1981`. When the framework is finished, open `file://$OUTPUT_DIR/QBOi.EXP1.AMIP.001_1977_1981/index.html` in a web browser to view the output report.

Currently the framework only analyzes data from one model run at a time. To run the MJO_prop_amp POD on the GFDL.CM4.c96L32.am4g10r8 sample data, delete or comment out the entry for QBOi.EXP1.AMIP.001 in the "caselist" section of the input file.

1.2.5 Next steps

Consult the following section (page 8) for how to run the framework on your own data and configure general settings.

1.3 Framework configuration

In this section we describe configuration options in more generality than the test case described in Quickstart installation instructions (page 4).

The complete set of options is described in ref_cli, or by running `% ./mdtf --help`. All options can be specified as a command-line flag (eg `--OUTPUT_DIR`) or as a JSON input file of the form provided in src/default_tests.jsonc[12]. We recommend using this file as a template, making copies and customizing it as needed.

Options given on the command line always take precedence over the input file. This is so you can store options that don't change in the file (eg. the input/output data paths) and use command-line flags to only set the options you want to change from run to run (eg. the analysis period start and end years). In all cases, the complete set of option values used in each run of the framework is included in the output directory, for reproducibility and provenance.

1.3.1 Adding your own model data

Currently the framework is only able to run on model data in the form of NetCDF files on a locally mounted disk following a specific directory hierarchy and filename convention. We hope to offer more flexibility in this area in the near future.

The directory/filename convention we use is

`$MODEL_DATA_ROOT`/<CASENAME>/<frequency>/<CASENAME>.<variable>.<frequency>.nc,

where

- <CASENAME> is any string used to identify the model run (experiment) that generated the data,
- <frequency> is the frequency at which the data is sampled: one of `1hr`, `3hr`, `6hr`, `day`, `mon` or `year`.
- <variable> is the name of the variable in your model's convention.

As an example, here's how the sample model data is organized:

```
inputdata
  model ( = $MODEL_DATA_ROOT)
      GFDL.CM4.c96L32.am4g10r8
          day
              GFDL.CM4.c96L32.am4g10r8.precip.day.nc
              (... other .nc files )
      QBOi.EXP1.AMIP.001
```

(continues on next page)

[12] https://github.com/NOAA-GFDL/MDTF-diagnostics/blob/develop/src/default_tests.jsonc

(continued from previous page)

```
          1hr
              QBOi.EXP1.AMIP.001.PRECT.1hr.nc
              (... other .nc files )
          3hr
              QBOi.EXP1.AMIP.001.PRECT.3hr.nc
          day
              QBOi.EXP1.AMIP.001.FLUT.day.nc
              (... other .nc files )
          mon
               QBOi.EXP1.AMIP.001.PS.mon.nc
               (... other .nc files )
   obs_data ( = $OBS_DATA_ROOT)
       (... supporting data for individual PODs )
```

If your model data is available on a locally mounted disk, you can make symlinks[13] that have the needed filenames and point to the data, rather than making copies of the files. For example,

```
% mkdir -p inputdata/model/my_new_experiment/day
% ln -s <long CMIP path>/pr_day_GFDL-ESM4_historical_r1i1p1f1_gr1_20100101-
 ↪20141231.nc \
? inputdata/model/my_experiment/day/my_new_experiment.pr.day.nc
```

will create a link to the file in the first argument that can be accessed normally:

```
inputdata
   model ( = $MODEL_DATA_ROOT)
       GFDL.CM4.c96L32.am4g10r8
       QBOi.EXP1.AMIP.001
       my_new_experiment
           day
               my_new_experiment.pr.day.nc
```

1.3.2 Running the code on your experiment's data

After adding your experiment's data to the directory hierarchy as described above, you can run the framework on that data using the following options. These can either be set in the "caselist" section of an input file (see src/default_tests.jsonc[14] for an example), or individually as command-line flags (eg. `--CASENAME my_new_experiment`). Required settings are:

- `CASENAME` should be the same string used to label your model run,
- `convention` describes the variable naming convention your model uses. Currently recognized values are `CMIP`, for CF-compliant output produced as part of CMIP6 (eg, by post-processing with CMOR) and `CESM`, `AM4` and `SPEAR`. We hope to offer support for the native variable naming conventions of a

[13] https://en.wikipedia.org/wiki/Symbolic_link
[14] https://github.com/NOAA-GFDL/MDTF-diagnostics/blob/develop/src/default_tests.jsonc

wider range of models in the future; for now, we recommend that you run the framework on cmorized[15] model data.

- `FIRSTYR` and `LASTYR` specify the analysis period.
- `model` and `experiment` are recorded if given, but not currently used.

When the framework is run, it determines if the variables each POD analyzes are present in the experiment data. Currently, the framework doesn't have the ability to transform data (eg, to average daily data to monthly frequency), so the match between your model data and each POD's requirements will need to be exact in order for the POD to run. If the framework can't find data requested by a POD, an error message will be logged in place of that POD's output that should help you diagnose the problem.

1.3.3 Other framework settings

The paths to input and output data described in Configure package paths (page 7) only need to be modified if the corresponding data is moved (or if you'd like to send output to a new location). Note that the framework doesn't retain default values for paths, so if you run it without an input file, all required paths will need to be given explicitly on the command line.

Other relevant flags controlling the framework's output are:

- `save_ps`: set to `true` to retain the vector .eps figures generated by PODs, in addition to the bitmap images linked to from the webpage.
- `save_nc`: set to `true` to retain netcdf files of any raw output data saved by PODs for further analysis.
- `make_variab_tar`: set to `true` to save the entire output directory as a .tar file, for archival or file transfer purposes.
- `overwrite`: set to `true` to overwrite previous framework output in `$OUTPUT_DIR`. By default, output with the same CASENAME and date range is assigned a unique name to ensure preexisting results are never overwritten.

These can be set as command-line flags each time the framework is run (eg. `--save_ps`), or as `true`/`false` values in the input file (`"save_ps": true`). Note that `true` and `false` in JSON must be written all lowercase, with no quotes.

1.3.4 Modifying POD settings

Individual PODs may provide user-configurable options in their `settings.jsonc` file, in the `"pod_env_vars"` section. These only need to be changed in rare or specific cases. Consult the POD's documentation for details.

1.4 Troubleshooting

Here we provide a short list of problems the MDTF team had previously encountered.

[15] https://cmor.llnl.gov/

1.4.1 The error message "convert: not authorized ..." shows up

The MDTF package generates figures in the PostScript (PS) format, and then uses the `convert` command (from the ImageMagick[16] software suite) to convert the PS files to PNG files. The convert error can occur after recent updates and can be solved as follows (requires permission):

In the file `/etc/ImageMagick/policy.xml`, change the `<policy domain="coder" rights="none" pattern="PS" />` to `<policy domain="coder" rights="read|write" pattern="PS" />`.

The folder name `ImageMagick` may depend on its version, e.g., `ImageMagick-6`.

1.4.2 Issues with standalone NCL installation

Many Linux distributions (Ubuntu, Mint, etc.) have offered a way of installing NCL[17] through their system package manager (apt, yum, etc.) This method of installation is not recommended: users may encounter errors when running the example PODs provided by NCAR, even if the environment variables and search path have been added.

The recommended method to install standalone NCL is by downloading the pre-compiled binaries from https://www.ncl.ucar.edu/Download/install_from_binary.shtml. Choose a download option according to the Linux distribution and hardware, unzip the file (results in 3 folders: `bin`, `include`, `lib`), create a folder ncl under the directory `/usr/local` (requires permission) and move the 3 unzipped folders into `/usr/local/ncl`. Then add the following lines to the `.bashrc` script (under the user's home directory; may be different if using shells other than bash, e.g., `.cshrc` for csh):

```
export NCARG_ROOT=/usr/local/ncl
export PATH:$NCARG_ROOT/bin:$PATH
```

1.4.3 Issues with the convective transition POD

The plotting scripts of this POD may not produce the desired figures with the latest version of matplotlib (because of the default size adjustment settings). The matplotlib version comes with the Anaconda 2 installer, version 5.0.1 has been tested. The readers can switch to this older version.

Depending on the platform and Linux distribution/version, a related error may occur with the error message "... ImportError: libcrypto.so.1.0.0: cannot open shared object file: No such file or directory". One can find the missing object file `libcrypto.so.1.0.0` in the subdirectory `~/anaconda2/pkgs/openssl-1.0.2l-h077ae2c_5/lib/`, where `~/anaconda2/` is where Anaconda 2 is installed. The precise names of the object file and openssl-folder may vary. Manually copying the object file to `~/anaconda2/lib/` should solve the error.

[16] https://imagemagick.org/index.php
[17] https://www.ncl.ucar.edu/

CHAPTER

TWO

SITE-SPECIFIC INFORMATION

2.1 GFDL-specific information

This page contains information specific to the site installation at the Geophysical Fluid Dynamics Laboratory[18].

2.1.1 Site installation

The DET team maintains a site-wide installation of the framework and all supporting data at `/home/mdteam/DET/analysis/mdtf/MDTF-diagnostics`. This is kept up-to-date and is accessible from both workstations and PPAN. Please contact us if your use case can't be accommodated by this installation.

2.1.2 FRE-centric modes of operation

In addition to the standard, interactive method of running MDTF diagnostics as described in the rest of the documentation, the site installation provides alternative ways to run the diagnostics within GFDL's existing workflow.

1. Within FRE XMLs. This is done by calling the mdtf_gfdl.csh[19] wrapper script from an `<analysis>` tag in the XML. Currently, FRE requires that each analysis script be associated with a single model `<component>`. This poses difficulties for diagnostics which use data generated by multiple components. We provide two ways to address this issue:

   A. If it's known ahead of time that a given `<component>` will dominate the run time and finish last, one can call `mdtf_gfdl.csh` from an `<analysis>` tag in that component only. In this case, the framework will search all data present in the /pp/ output directory when it's called. The `<component>` being used doesn't need to generate data analyzed by the diagnostics; in this case it's only used to schedule the diagnostics' execution.

   B. If one doesn't know which `<component>` will finish last, a more robust solution is to call `mdtf_gfdl.csh --component_only` from each `<component>` generating data to be analyzed. When the `--component_only` flag is set, every time the framework is called it will only run the diagnostics for which all the input data is available and which haven't run already (which haven't written their output to `$OUTPUT_DIR`.

[18] https://www.gfdl.noaa.gov/

[19] https://github.com/NOAA-GFDL/MDTF-diagnostics/blob/feature/gfdl-data/src/mdtf_gfdl.csh

2. As a batch job on PPAN, managed via slurm. This is handled via the mdtf_gfdl_interactive.csh[20] wrapper script.
3. Called from an interactive shell on PPAN or workstations.

2.1.3 Data retrieval options

The framework is currently configured to search data from two types of directory hierarchies. The framework will determine what's intended based on its input, but this choice can be overridden by passing the following options with the `--data_manager` flag:

- The /pp/ hierarchy used by FRE (`--data_manager Gfdl_PP`). In this case `CASE_ROOT_DIR` should be set to the root of the directory hierarchy (ie, ending in `/pp`).
- The CMIP6 DRS for published data on the Unified Data Archive (`--data_manager Gfdl_UDA_CMIP6`). In this case `CASE_ROOT_DIR` should not be set, but the `--model` and `--experiment` settings should be populated.
- The CMIP6 DRS for unpublished data on `/data_cmip6`. This option must be requested explicitly with `--data_manager Gfdl_data_cmip6`. In this case `CASE_ROOT_DIR` should not be set, but the `--model` and `--experiment` settings should be populated.

2.1.4 GFDL-specific options

In addition to the framework's normal command-line options, the following site-specific options are recognized:

- `--GFDL-PPAN-TEMP, --GFDL_PPAN_TEMP <DIR>`: If running on the GFDL PPAN cluster, set the `$MDTF_GFDL_TMPDIR` environment variable to this location and create temp files here. Note: must be accessible via gcp. Defaults to `$TMPDIR`.
- `--GFDL-WS-TEMP, --GFDL_WS_TEMP <DIR>`: If running on a GFDL workstation, set the `$MDTF_GFDL_TMPDIR` environment variable to this location and create temp files here. The directory will be created if it doesn't exist. Note: must be accessible via gcp. Defaults to `/net2/$USER/tmp`.
- `--frepp`: Normally this is set by the mdtf_gfdl.csh[21] wrapper script, and not directly by the user. Set flag to run framework in "online" mode (1a. or 1b. above), processing data as part of the FRE pipeline.
- `--ignore-component, --ignore_component`: Normally this is set by the mdtf_gfdl.csh[22] wrapper script, and not directly by the user. If set, this flag tells the framework to search the entire /pp/ directory for model data (1a. above); default is to restrict to model component passed by FRE. Ignored if `--frepp` is not set.

2.1.5 GFDL-specific defaults

The following paths are set to more useful default values:

[20] https://github.com/NOAA-GFDL/MDTF-diagnostics/blob/feature/gfdl-data/src/mdtf_gfdl_interactive.csh
[21] https://github.com/NOAA-GFDL/MDTF-diagnostics/blob/feature/gfdl-data/src/mdtf_gfdl.csh
[22] https://github.com/NOAA-GFDL/MDTF-diagnostics/blob/feature/gfdl-data/src/mdtf_gfdl.csh

- `--OBS-DATA-REMOTE, --OBS_DATA_REMOTE <DIR>`: Site-specific installation of observational data used by individual PODs at `/home/Oar.Gfdl.Mdteam/DET/analysis/mdtf/obs_data`. If running on PPAN, this data will be GCP'ed to the current node.
- `--OBS-DATA-ROOT, --OBS_DATA_ROOT <DIR>`: Local directory for observational data. Defaults to `$MDTF_GFDL_TMPDIR/inputdata/obs_data`, where the environment variable `$MDTF_GFDL_TMPDIR` is defined as described above.
- `--MODEL-DATA-ROOT, --MODEL_DATA_ROOT <DIR>`: Local directory for model data. Defaults to `$MDTF_GFDL_TMPDIR/inputdata/model`, where the environment variable `$MDTF_GFDL_TMPDIR` is defined as described above.
- `--WORKING-DIR, --WORKING_DIR <DIR>`: Working directory. Defaults to `$MDTF_GFDL_TMPDIR/wkdir`, where the environment variable `$MDTF_GFDL_TMPDIR` is defined as described above.
- `--OUTPUT-DIR, --OUTPUT_DIR, -o <DIR>`: Destination for output files. Defaults to `$HOME/mdtf_out`, which will be created if it doesn't exist.

CHAPTER

THREE

ACKNOWLEDGEMENTS

Development of this code framework for process-oriented diagnostics was supported by the National Oceanic and Atmospheric Administration[23] (NOAA) Climate Program Office Modeling, Analysis, Predictions and Projections[24] (MAPP) Program (grant # NA18OAR4310280). Additional support was provided by University of California Los Angeles[25], the Geophysical Fluid Dynamics Laboratory[26], the National Center for Atmospheric Research[27], Colorado State University[28], Lawrence Livermore National Laboratory[29] and the US Department of Energy[30].

Many of the process-oriented diagnostics modules (PODs) were contributed by members of the NOAA Model Diagnostics Task Force[31] under MAPP support. Statements, findings or recommendations in these documents do not necessarily reflect the views of NOAA or the US Department of Commerce.

3.1 Disclaimer

This repository is a scientific product and is not an official communication of the National Oceanic and Atmospheric Administration, or the United States Department of Commerce. All NOAA GitHub project code is provided on an 'as is' basis and the user assumes responsibility for its use. Any claims against the Department of Commerce or Department of Commerce bureaus stemming from the use of this GitHub project will be governed by all applicable Federal law. Any reference to specific commercial products, processes, or services by service mark, trademark, manufacturer, or otherwise, does not constitute or imply their endorsement, recommendation or favoring by the Department of Commerce. The Department of Commerce seal and logo, or the seal and logo of a DOC bureau, shall not be used in any manner to imply endorsement of any commercial product or activity by DOC or the United States Government.

[23] https://www.noaa.gov/
[24] https://cpo.noaa.gov/Meet-the-Divisions/Earth-System-Science-and-Modeling/MAPP
[25] https://www.ucla.edu/
[26] https://www.gfdl.noaa.gov/
[27] https://ncar.ucar.edu/
[28] https://www.colostate.edu/
[29] https://www.llnl.gov/
[30] https://www.energy.gov/
[31] https://cpo.noaa.gov/Meet-the-Divisions/Earth-System-Science-and-Modeling/MAPP/MAPP-Task-Forces/Model-Diagnostics-Task-Force