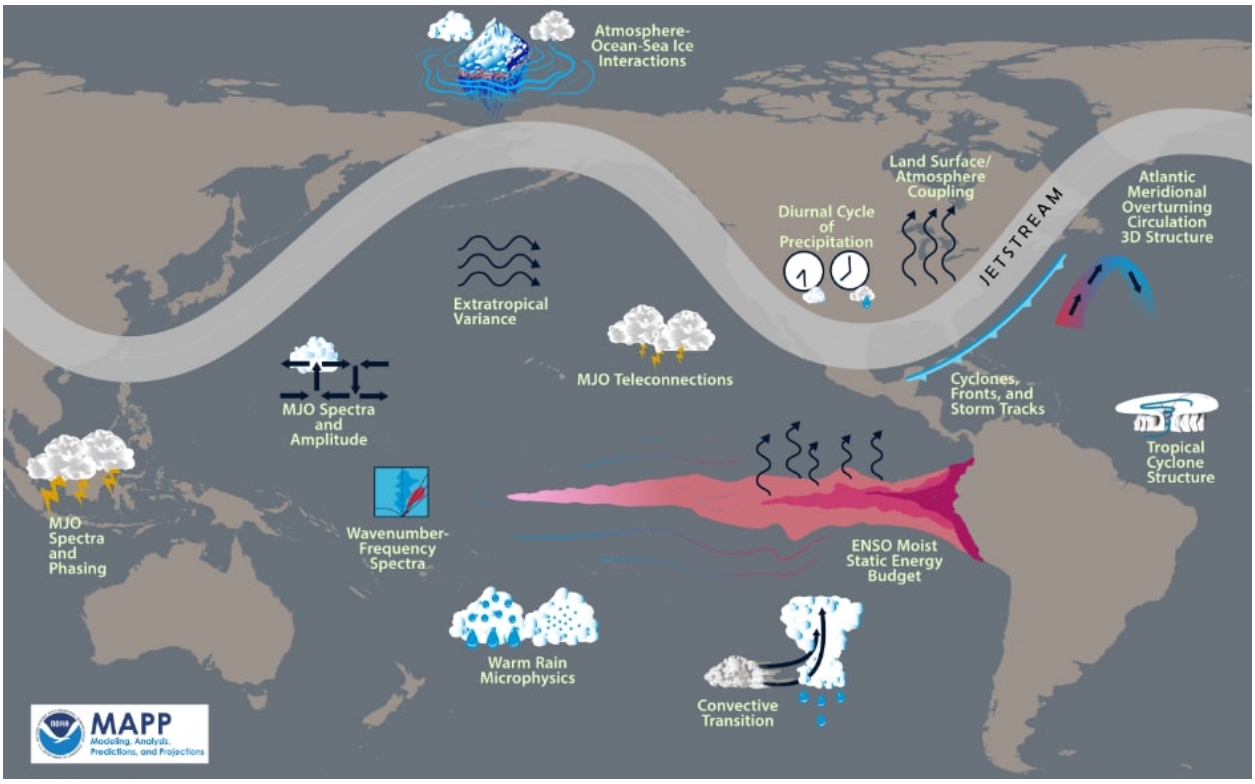


MDTF Getting Started Guide

Release 3.0 beta 2

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

Aug 16, 2020

CONTENTS

CHAPTER

ONE

GETTING STARTED

1.1 Overview

Welcome! In this section we'll describe what the Model Diagnostics Task Force (MDTF) framework is, how it works, and how you can contribute your own diagnostic scripts.

1.1.1 Purpose

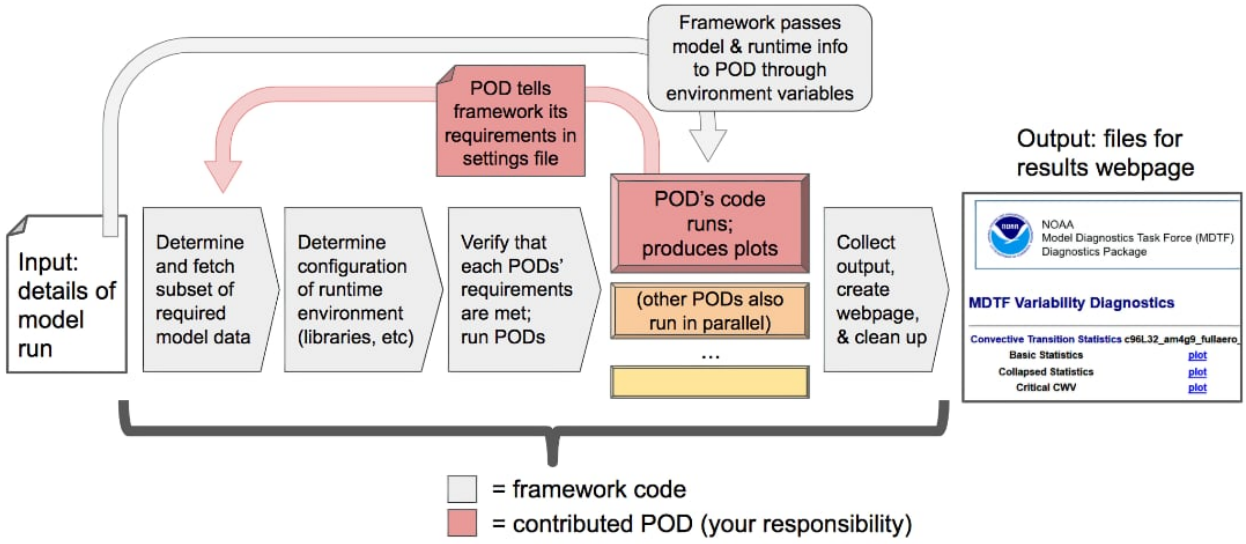
The scientific motivation and content behind the framework was described in E. D. Maloney et al. (2019): Process-Oriented Evaluation of Climate and Weather Forecasting Models. BAMS, 100 (9), 1665–1686, doi:10.1175/BAMS-D-18-0042.1[1].

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

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

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.



As shown in the figure above, the MDTF framework itself performs common data management and support tasks (gray boxes) before and after the individual POD scripts are run. The PODs (colored boxes) are developed by different research groups and run independently of one another. 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 3) the framework and run it on sample model data.
- We describe the most common configuration options (page 8) for running the framework on your own model data. Also see the full list of command-line options.
- If you encounter a bug, check the GitHub issue tracker[2].

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

1.1.4 Getting started for POD developers

Information for researchers wishing to contribute a POD to the framework is provided in the Developer section; consult the quickstart guide for an overview and the checklist of items needed for submitting your POD.

The framework is designed to require minimal changes to existing analysis scripts. We recommend that developers of new PODs start independently of the framework and adapt it for the framework's use once it's fully debugged. 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.
- 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.

1.2 Quickstart installation instructions

This section provides instructions for downloading, installing and running a test of the MDTF framework using sample model data. The MDTF framework has been tested on UNIX/LINUX, Mac OS, and Windows Subsystem for Linux.

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

Summary of steps for installing the framework

You will need to download the source code, digested observational data, and sample model data (Section 1.2.1). Afterwards, we describe how to install software dependencies using the conda[3] package manager (Section 1.2.2, Section 1.2.3) and run the framework on sample model data (Section 1.2.4 and Section 1.2.5).

1.2.1 Download the framework code and supporting data

Obtaining the code

The official repo for the MDTF code is hosted at the NOAA-GFDL GitHub account[4]. We recommend that end users download and test the latest official release[5].

To install the MDTF framework, create a directory named `mdtf` and unzip the code downloaded from the release page[6] there. This will create a directory titled `MDTF-diagnostics-3.0-beta.2` containing the files

[3] https://docs.conda.io/en/latest/
[4] https://github.com/NOAA-GFDL/MDTF-diagnostics
[5] https://github.com/NOAA-GFDL/MDTF-diagnostics/releases/tag/v3.0-beta.2
[6] https://github.com/NOAA-GFDL/MDTF-diagnostics/releases/tag/v3.0-beta.2

listed on the GitHub page. Below we refer to this MDTF-diagnostics directory as `$CODE_ROOT`. It contains the following subdirectories:

- `diagnostics/`: directory containing source code and documentation of individual PODs.
- `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 `main` 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 by locally cloning the repo and checking out this branch, as described in ref-dev-git-intro.

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 required files are:

- Digested observational data (159 Mb): MDTF_v2.1.a.obs_data.tar[7].
- NCAR-CESM-CAM sample data (12.3 Gb): model.QBOi.EXP1.AMIP.001.tar[8].
- NOAA-GFDL-CM4 sample data (4.8 Gb): model.GFDL.CM4.c96L32.am4g10r8.tar[9].

Note that the above paths are symlinks to the most recent versions of the data and will be reported as zero bytes in an FTP client.

Download these three files and extract the contents in the following hierarchy under the `mdtf` directory:

```
mdtf
   MDTF-diagnostics
   inputdata
       model ( = $MODEL_DATA_ROOT)
           GFDL.CM4.c96L32.am4g10r8
               day
                   GFDL.CM4.c96L32.am4g10r8.precip.day.nc
                   (... other .nc files )
           QBOi.EXP1.AMIP.001
               1hr
                   QBOi.EXP1.AMIP.001.PRECT.1hr.nc
                   (... other .nc files )
               3hr
                   QBOi.EXP1.AMIP.001.PRECT.3hr.nc
               day
```

(continues on next page)

[7] ftp://ftp.cgd.ucar.edu/archive/mdtf/MDTF_v2.1.a.obs_data.tar
[8] ftp://ftp.cgd.ucar.edu/archive/mdtf/model.QBOi.EXP1.AMIP.001.tar
[9] ftp://ftp.cgd.ucar.edu/archive/mdtf/model.GFDL.CM4.c96L32.am4g10r8.tar

(continued from previous page)

```
                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 )
```

The default test case uses the QBOi.EXP1.AMIP.001 data. The GFDL.CM4.c96L32.am4g10r8 data is only for testing the MJO Propagation and Amplitude POD.

You can put the observational data and model output in different locations (e.g., for space reasons) by changing the values of `OBS_DATA_ROOT` and `MODEL_DATA_ROOT` as described in Section 1.2.4.

1.2.2 Install the conda package manager, if needed

The MDTF framework code is written in Python 3, but supports running PODs written in a variety of scripting languages and combinations of libraries. We use conda[10], a free, open-source package manager, to install and manage these dependencies. Conda is one component of the Miniconda[11] and Anaconda[12] Python distributions, so having Miniconda or Anaconda is sufficient but not required.

For maximum portability and ease of installation, we recommend that all users manage dependencies through conda, even if they have a pre-existing installations of the required languages. A complete installation of all dependencies requires roughly 5 Gb, and the location of this installation can be set with the `$CONDA_ENV_DIR` setting described below. Note that conda does not create duplicates of dependencies that are already installed (instead using hard links by default).

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

Users with an existing conda installation should skip this section and proceed to Section 1.2.3.

- 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.

> Warning: Do not install a new copy of 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, e.g., environment modules.) The framework's environments are designed to coexist with an existing Miniconda/Anaconda installation.

[10] https://docs.conda.io/en/latest/
[11] https://docs.conda.io/en/latest/miniconda.html
[12] https://www.anaconda.com/

- If you do not have a pre-existing conda installation, we recommend installing Miniconda 3.x, available here[13]. This version is not required: any version of Miniconda/Anaconda (2 or 3) released after June 2019 will work equally well.
  - Follow the installation instructions[14] appropriate for your system. Toward the end of the installation process, enter "yes" at "Do you wish the installer to initialize Miniconda3 by running conda init?" (or similar) prompt. This will allow the installer to add the conda path to the user's shell startup script (e.g., `~/.bashrc` or `~/.cshrc`).
  - Restart the terminal to reload the updated shell startup script.
  - Mac OS users may encounter a message directing them to install the Java JDK. This can be ignored.

1.2.3 Install framework dependencies with conda

As described above, all software dependencies for the framework and PODs are managed through conda environments.

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 `$CONDA_ROOT` below. If you don't have write access to this location (eg, on a multi-user system), you'll need to tell conda to install files in a non-default location `$CONDA_ENV_DIR`, as described below.

Next, run

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

to install all dependencies, which takes ~10 min (depending on machine and internet connection). The names of all framework-created environments begin with "_MDTF", so as not to conflict with user-created environments in a preexisting conda installation.

- Substitute the actual paths for `$CODE_ROOT`, `$CONDA_ROOT`, and `$CONDA_ENV_DIR`.
- The optional `--env_dir` flag directs conda to install framework dependencies in `$CONDA_ENV_DIR` (for space reasons, or if you don't have write access). If this flag is omitted, the environments will be installed in `$CONDA_ROOT/envs/` by default.
- The `--all` flag makes the script install all dependencies for all PODs. To selectively update individual conda environments after installation, use the `--env` flag instead. For instance, `% ./src/conda/conda_env_setup.sh --env base --conda_root $CONDA_ROOT --env_dir $CONDA_ENV_DIR` will update the environment named "_MDTF_base" defined in `src/conda/env_base.yml`, and so on.

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

[13] https://docs.conda.io/en/latest/miniconda.html
[14] https://docs.conda.io/projects/conda/en/latest/user-guide/install/index.html

the above commands. These environments can be uninstalled by simply deleting the "_MDTF" directories under `$CONDA_ENV_DIR` (or `$CONDA_ROOT/envs/` by default).

1.2.4 Configure framework paths

The MDTF framework supports setting configuration options in a file as well as on the command line. An example of the configuration file format is provided at src/default_tests.jsonc[15]. We recommend configuring the following settings by editing a copy of this file.

Relative paths in the configuration file will be interpreted relative to `$CODE_ROOT`. The following settings need to be configured before running the framework:

- If you've saved the supporting data in the directory structure described in Obtaining supporting data (page 4), the default values for `OBS_DATA_ROOT` and `MODEL_DATA_ROOT` given in `src/default_tests.jsonc` (`../inputdata/obs_data` and `../inputdata/model`, respectively) will be correct. If you put the data in a different location, these paths should be changed accordingly.
- `OUTPUT_DIR` should be set to the desired location for output files. 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 `$CONDA_ROOT` used above in Install framework dependencies with conda (page 6).
- If you specified a non-default conda environment location with `$CONDA_ENV_DIR`, set `conda_env_root` to that value; otherwise, leave it blank.

1.2.5 Run the MDTF framework on sample data

Location of the MDTF executable

The MDTF framework is run via a wrapper script at `$CODE_ROOT/mdtf`.

This is created by the conda environment setup script used in Section 1.2.3. The wrapper script activates the framework's conda environment before calling the framework's code (and individual PODs). To verify that the framework and environments were installed successfully, run

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

This should print the current version of the framework.

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

Run the framework on sample data

If you've downloaded the NCAR-CESM-CAM sample data (described in Obtaining supporting data (page 4) above), you can now perform a trial run of the framework:

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

Run time may be 10-20 minutes, depending on your system.

- If you edited or renamed `src/default_tests.jsonc`, as recommended in the previous section, pass the path to that configuration file instead.
- The output files for this test case will be written to `$OUTPUT_DIR/MDTF_QBOi.EXP1.AMIP.001_1977_1981`. When the framework is finished, open `$OUTPUT_DIR/QBOi.EXP1.AMIP.001_1977_1981/index.html` in a web browser to view the output report.
- The framework defaults to running all available PODs, which is overridden by the `pod_list` option in the `src/default_tests.jsonc` configuration file. Individual PODs can be specified as a comma-delimited list of POD names.
- 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 section for QBOi.EXP1.AMIP.001 in `caselist` section of the configuration file, and uncomment the section for GFDL.CM4.c96L32.am4g10r8.

1.3 Framework configuration for user model data

In this section we describe how to run the framework with your own model data, and more configuration options than the test case described in Quickstart installation instructions (page 3).

The complete set of configuration options is described in ref_cli, or by running `% ./mdtf --help`. All options can be specified as a command-line flag (e.g., `--OUTPUT_DIR`) or as a JSON configuration input file of the form provided in src/default_tests.jsonc[16]. 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 frequently change in the file (e.g., the input/output data paths) and use command-line flags to set only those options you want to change from run to run (e.g., the analysis period start and end years). In all cases, the complete set of option values used in each run of the framework will be included in the log file as part of the output, for reproducibility and provenance.

Summary of steps for running the framework on user data

1. Save or link model data files following the framework's filename convention.
2. Select the variable name convention used by the model.
3. Edit the configuration input file accordingly, then
4. Run the framework.

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

1.3.1 Adding your 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, with one variable per file. 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
           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, we recommend creating symlinks[17] 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 $path_to_file/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)
```

(continues on next page)

[17] https://en.wikipedia.org/wiki/Symbolic_link

(continued from previous page)

```
    GFDL.CM4.c96L32.am4g10r8
    QBOi.EXP1.AMIP.001
    my_new_experiment
        day
            my_new_experiment.pr.day.nc
```

1.3.2 Select the model's variable name convention

The framework requires specifying a convention for variable names used in the model data. Currently recognized conventions are

- `CMIP`, for CF-compliant output produced as part of CMIP6;
- `CESM`, for the NCAR community earth system model[18];
- `AM4`, for the NOAA-GFDL atmosphere model[19];
- `SPEAR`, for the NOAA-GFDL seasonal model[20].

We hope to offer support for the variable naming conventions of a wider range of models in the future. For the time being, please process output of models not on this list with CMOR[21] to make them CF-compliant.

Alternatively, the framework will load any lookup tables of the form `src/fieldlist_$convention.jsonc` and use them for variable name conversion. Users can add new files in this format to specify new conventions. For example, in `src/fieldlist_CESM.jsonc` the line `"pr_var" : "PRECT"` means that the CESM name for the precipitation rate is PRECT (case sensitive). In addition, `"pr_conversion_factor" : 1000` specifies the conversion factor to CF standard units for this variable.

1.3.3 Running the code on your data

After adding your model 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 the configuration input file (see src/default_tests.jsonc[22] for an example/template), or individually as command-line flags (e.g., `--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, determined in the previous section.
- `FIRSTYR` and `LASTYR` specify the analysis period.
- `model` and `experiment` are recorded if given, but not currently used.

[18] http://www.cesm.ucar.edu/
[19] https://www.gfdl.noaa.gov/am4/
[20] https://www.gfdl.noaa.gov/research_highlight/spear-the-next-generation-gfdl-modeling-system-for-seasonal-to-multidecadal-prediction-and-pr
[21] https://cmor.llnl.gov/
[22] https://github.com/NOAA-GFDL/MDTF-diagnostics/blob/develop/src/default_tests.jsonc

When the framework is run, it determines whether the data each POD needs to run is present in the model data being provided. Specifically, the model must provide all variables needed by a POD at the required frequency. Consult the documentation for a POD to determine the data it requires.

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. We hope to add the ability to transform data (eg, to average daily data to monthly frequency) in order to simplify this process.

1.3.4 Other framework settings

The paths to input and output data (described in Configure framework 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 don't specify a configuration 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 data output at intermediate steps 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 (e.g.,. `--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.

1.3.5 Modifying POD settings

Individual PODs may provide user-configurable options in the `"pod_env_vars"` section of their `settings.jsonc` file, which is located in each POD's source code directory under `/diagnostics`. These only need to be changed in rare or specific cases. Consult the POD's documentation for details.

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[23].

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[24] 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`.

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

[24] 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[25] 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[26] 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[27] 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.

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

2.1.5 GFDL-specific defaults

The following paths are set to more useful default values:

- `--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[28] (NOAA) Climate Program Office Modeling, Analysis, Predictions and Projections[29] (MAPP) Program (grant # NA18OAR4310280). Additional support was provided by University of California Los Angeles[30], the Geophysical Fluid Dynamics Laboratory[31], the National Center for Atmospheric Research[32], Colorado State University[33], Lawrence Livermore National Laboratory[34] and the US Department of Energy[35].

Many of the process-oriented diagnostics modules (PODs) were contributed by members of the NOAA Model Diagnostics Task Force[36] 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.

[28] https://www.noaa.gov/
[29] https://cpo.noaa.gov/Meet-the-Divisions/Earth-System-Science-and-Modeling/MAPP
[30] https://www.ucla.edu/
[31] https://www.gfdl.noaa.gov/
[32] https://ncar.ucar.edu/
[33] https://www.colostate.edu/
[34] https://www.llnl.gov/
[35] https://www.energy.gov/
[36] https://cpo.noaa.gov/Meet-the-Divisions/Earth-System-Science-and-Modeling/MAPP/MAPP-Task-Forces/Model-Diagnostics-Task-Force