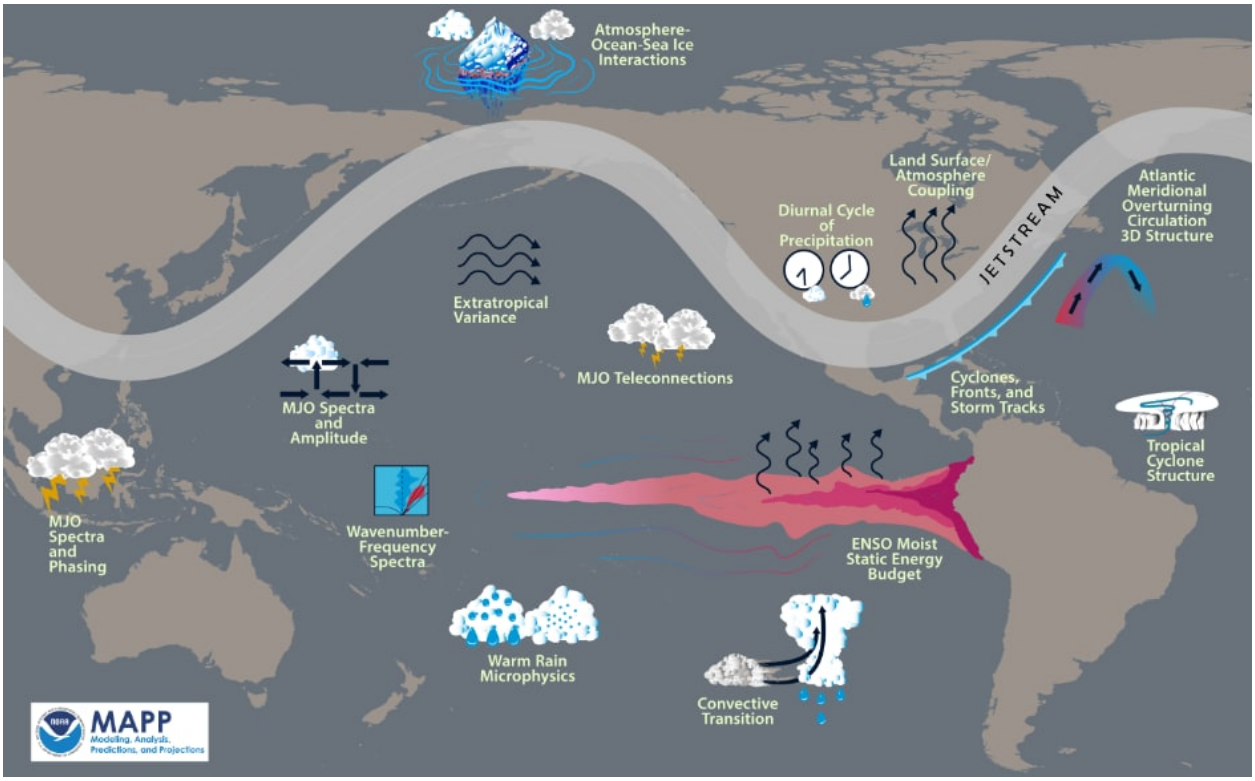


MDTF Getting Started Guide

Release 3.0 beta 4

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

Jun 13, 2022

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

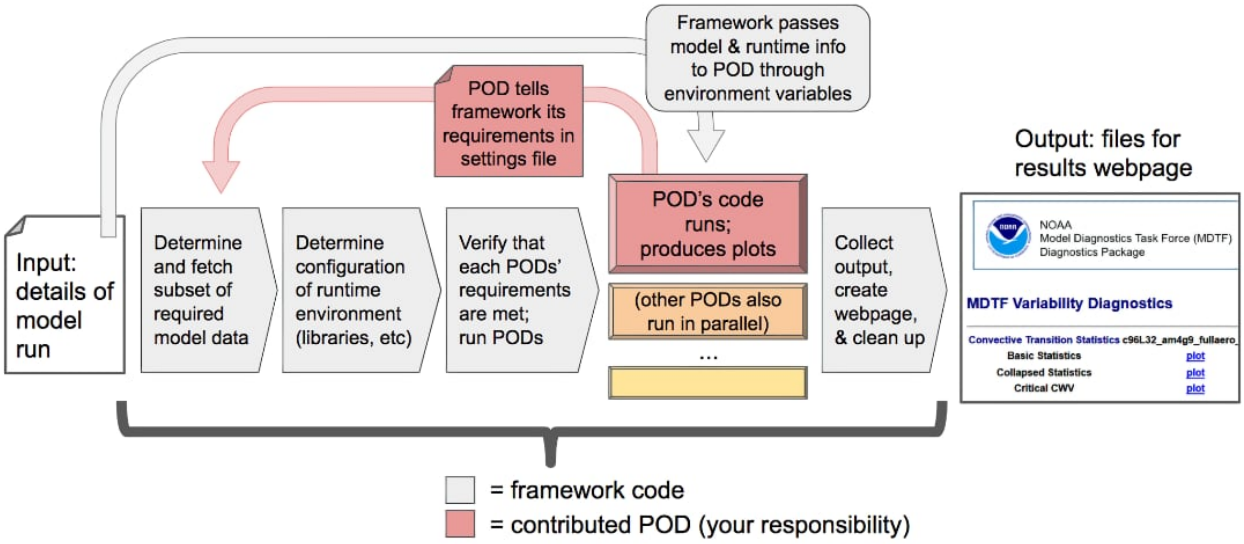
The scientific motivation and content behind the framework was described in E. D. Maloney et al. (2019): Process-Oriented Evaluation of Climate and Weather Forecasting Models. BAMS, 100 (9), 1665–1686, doi:10.1175/BAMS-D-18-0042.1[1].

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 10) 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 Installation instructions

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

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 1.2.3) and run the framework on sample model data (Section 1.2.4 and Section 1.2.5).

Throughout this document, `%` indicates the shell prompt and is followed by commands to be executed in a terminal in `fixed-width font`. Variable values are denoted by angle brackets, e.g. <HOME> is the path to your home directory returned by running `% echo $HOME`.

1.2.1 Obtaining the code

The official repo for the package's code is hosted at the NOAA-GFDL GitHub account[4]. To simplify updating the code, we recommend that all users obtain the code using git. For more in-depth instructions on how to use git, see dev_git_intro.

To install the MDTF package on a local machine, open a terminal and create a directory named mdtf. Instructions for end-users and new developers are then as follows:

- For end users:
  1. `% cd mdtf`, then clone your fork of the MDTF repo on your machine:
     `% git clone https://github.com/<your GitHub account name>/MDTF-diagnostics`.

[3] https://docs.conda.io/en/latest/
[4] https://github.com/NOAA-GFDL/MDTF-diagnostics

2. Verify that you are on the main branch: `% git branch`. This is the default, but it never hurts to get in the habit of running git branch before you start working.
3. Check out the latest official release[5]:
   `% git checkout tags/v3.0-beta.3`.
4. Proceed with the installation process described below.
5. Check out a new branch that will contain your edited config files:
   `% git checkout -b <branch name>`.
6. Update the config files, then commit the changes:
   `% git commit -m "description of your changes"`.
7. Push the changes on your branch to your remote fork:
   `% git push -u origin <branch name>`.

- For new POD developers:

1. `% cd mdtf`, then clone your fork of the MDTF repo on your machine:
   `% git clone https://github.com/<your GitHub account name>/MDTF-diagnostics`.
2. Check out the `develop` branch: `% git checkout develop`.
3. Proceed with the installation process described below.
4. Check out a new branch for your POD:
   `% git checkout -b feature/<Your POD's name>`.
5. Edit existing files/create new files, then commit the changes:
   `% git commit -m "description of your changes"`.
6. Push the changes on your branch to your remote fork:
   `% git push -u origin feature/<Your POD's name>`.

The path to the code directory (`.../mdtf/MDTF-diagnostics`) is referred to as <CODE_ROOT> in what follows. It contains the following subdirectories:

- `diagnostics/`: directory containing source code and documentation of individual PODs.
- `doc/`: source code for the documentation website.
- `shared/`: shared code and resources for use by both the framework and PODs.
- `sites/`: site-specific code and configuration files.
- `src/`: source code of the framework itself.
- `tests/`: general tests for the framework.

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

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

1.2.2 Obtaining supporting data

Supporting observational data and sample model data are available via anonymous FTP from ftp://ftp.cgd.ucar.edu/archive/mdtf. The observational data is required for the PODs' operation, while the sample model data is optional and only needed for test and demonstration purposes. The files you will need to download are:

- Digested observational data (159 Mb): MDTF_v2.1.a.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].

The default test case uses the `QBOi.EXP1.AMIP.001` sample dataset, and the `GFDL.CM4.c96L32.am4g10r8` sample dataset is only for testing the MJO Propagation and Amplitude POD. Note that the above paths are symlinks to the most recent versions of the data, and will be reported as having a size of zero bytes in an FTP client.

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

```
mdtf
   MDTF-diagnostics ( = <CODE_ROOT>)
   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 )
```

Note that `mdtf` now contains both the `MDTF-diagnostics` and `inputdata` directories.

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

[6] ftp://ftp.cgd.ucar.edu/archive/mdtf/MDTF_v2.1.a.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

1.2.3 Installing dependencies

Installing XQuartz on MacOS

If you're installing on a MacOS system, you will need to install XQuartz[9]. If the XQuartz executable isn't present in `/Applications/Utilities`, you will need to download and run the installer from the previous link.

The reason for this requirement is that the X11 libraries are required dependencies[10] for the NCL scripting language, even when it's run non-interactively. Because the required libraries cannot be installed through conda (next section), this installation needs to be done as a manual step.

Managing dependencies with the conda package manager

The MDTF framework code is written in Python 3.7, but supports running PODs written in a variety of scripting languages and combinations of libraries. To ensure that the correct versions of these dependencies are installed and available, we use conda[11], a free, open-source package manager. Conda is one component of the Miniconda[12] and Anaconda[13] python distributions, so having Miniconda/Anaconda is sufficient but not necessary.

For maximum portability and ease of installation, we recommend that all users manage dependencies through conda using the steps below, 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_root` and `--env_dir` flags described below. Users may install their own copies of Anaconda/Miniconda on their machine, or use a centrally-installed version managed by their institution. Note that installing your own copy of Anaconda/Miniconda will re-define the default locations of the conda executable and environment directory defined in your .bashrc or .cshrc file if you have previously used a version of conda managed by your institution, so you will have to re-create any environments made using central conda installations.

If these space requirements are prohibitive, we provide an alternate method of installation which makes no use of conda and instead assumes the user has installed the required external dependencies, at the expense of portability. This is documented in a separate section.

Installing the conda package manager

In this section, we install the conda package manager if it's not already present on your system.

- To determine if conda is installed, run `% conda info` as the user who will be using the package. The package has been tested against versions of conda >= 4.7.5. If a pre-existing conda installation is present, continue to the following section to install the package's environments. These environments will co-exist with any existing installation.

[9] https://www.xquartz.org/
[10] https://www.ncl.ucar.edu/Download/macosx.shtml#InstallXQuartz
[11] https://docs.conda.io/en/latest/
[12] https://docs.conda.io/en/latest/miniconda.html
[13] https://www.anaconda.com/

Note: Do not reinstall Miniconda/Anaconda if it's already installed for the user who will be running the package: the installer will break the existing installation (if it's not managed with, e.g., environment modules.)

- If `% conda info` doesn't return anything, you will need to install conda. We recommend doing so using the Miniconda installer (available here[14]) for the most recent version of python 3, although any version of Miniconda or Anaconda released after June 2019, using python 2 or 3, will work.
- Follow the conda installation instructions[15] appropriate to 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 login script (e.g., `~/.bashrc` or `~/.cshrc`). It's necessary to modify your login script due to the way conda is implemented.
- Start a new shell to reload the updated shell login script.

Installing the package's conda environments

In this section we use conda to install the versions of the language interpreters and third-party libraries required by the package's diagnostics.

- First, determine the location of your conda installation by running `% conda info --base` as the user who will be using the package. This path will be referred to as <CONDA_ROOT> below.
- If you don't have write access to <CONDA_ROOT> (for example, if conda has been installed for all users of a multi-user system), you will need to tell conda to install its files in a different, writable location. You can also choose to do this out of convenience, e.g. to keep all files and programs used by the MDTF package together in the `mdtf` directory for organizational purposes. This location will be referred to as <CONDA_ENV_DIR> below.
- Install all the package's conda environments by running

```
% cd <CODE_ROOT>
% ./src/conda/conda_env_setup.sh --all --conda_root <CONDA_ROOT> --env_dir <CONDA_
↪ENV_DIR>
```

  The names of all conda environments used by the package begin with "_MDTF", so as not to conflict with other environments in your conda installation. The installation process should finish within ten minutes.

  - Substitute the paths identified above for <CONDA_ROOT> and <CONDA_ENV_DIR>.
  - If the `--env_dir` flag is omitted, the environment files will be installed in your system's conda's default location (usually <CONDA_ROOT>/envs).

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

Note: After installing the framework-specific conda environments, you shouldn't alter them manually (i.e., never run `conda update` on them). To update the environments after an update to a new release of the framework code, re-run the above commands.

These environments can be uninstalled by deleting their corresponding directories under <CONDA_ENV_DIR> (or <CONDA_ROOT>/envs/).

Location of the installed executable

The script used to install the conda environments in the previous section creates a script named `mdtf` in the MDTF-diagnostics directory. This script is the executable you'll use to run the package and its diagnostics. To test the installation, run

```
% cd <CODE_ROOT>
% ./mdtf --version
```

The output should be

```
=== Starting <CODE_ROOT>/mdtf_framework.py

mdtf 3.0 beta 4
```

1.2.4 Configuring framework paths

In order to run the diagnostics in the package, it needs to be provided with paths to the data and code dependencies installed above. In general, there are two equivalent ways to configure any setting for the package:

- All settings are configured with command-line flags. The full documentation for the command line interface is at ref_cli.
- Long lists of command-line options are cumbersome, and many of the settings (such as the paths to data that we set here) don't change between different runs of the package. For this purpose, any command-line setting can also be provided in an input configuration file.
- The two methods of setting options can be freely combined. Any values set explicitly on the command line will override those given in the configuration file.

For the remainder of this section, we describe how to edit and use configuration files, since the paths to data, etc., we need to set won't change.

An example of the configuration file format is provided at src/default_tests.jsonc[16]. This is meant to be a template you can customize according to your purposes: save a copy of the file at <config_file_path> and open it in a text editor. The following paths need to be configured before running the framework:

- `OBS_DATA_ROOT` should be set to the location of the supporting data that you downloaded in Section 1.2.2. If you used the directory structure described in that section, the default value provided in the configuration file (`../inputdata/obs_data/`) will be correct. If you put the data in a different location,

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

this value should be changed accordingly. Note that relative paths can be used in the configuration file, and are always resolved relative to the location of the MDTF-diagnostics directory (<CODE_ROOT>).

- Likewise, `MODEL_DATA_ROOT` should be updated to the location of the NCAR-CESM-CAM sample data (`model.QBOi.EXP1.AMIP.001.tar`)downloaded in Section 1.2.2. This data is required to run the test in the next section. If you used the directory structure described in Section 1.2.2, the default value provided in the configuration file (`../inputdata/model/`) will be correct.
- `conda_root` should be set to the location of your conda installation: the value of <CONDA_ROOT> that was used in Section 1.2.3.
- Likewise, if you installed the package's conda environments in a non-default location by using the `--env_dir` flag in Section 1.2.3, the option `conda_env_root` should be set to this path (<CONDA_ENV_DIR>).
- Finally, `OUTPUT_DIR` should be set to the location you want the output files to be written to (default: `mdtf/wkdir/`; will be created by the framework). The output of each run of the framework will be saved in a different subdirectory in this location.

In Running the package on your data (page 10), we describe more of the most important configuration options for the package, and in particular how you can configure the package to run on different data. A complete description of the configuration options is at ref_cli, or can be obtained by running `% ./mdtf --help`.

1.2.5 Running the package on sample model data

You are now ready to run the package's diagnostics on the sample data from NCAR's CESM-CAM model. We assume you've edited a copy of src/default_tests.jsonc[17], which is saved at <config_file_path>, as described in the previous section.

```
% cd <CODE_ROOT>
% ./mdtf -f <config_file_path>
```

The first few lines of output will be

```
=== Starting <CODE_ROOT>/mdtf_framework.py

PACKAGE SETTINGS:
case_list(0):
   CASENAME: QBOi.EXP1.AMIP.001
   model: CESM
   convention: CESM
   FIRSTYR: 1977
   LASTYR: 1981
[...]
```

Run time may be up to 10-20 minutes, depending on your system. The final lines of output should be:

```
Exiting normally from <CODE_ROOT>/src/core.py
Summary for QBOi.EXP1.AMIP.001:
```

(continues on next page)

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

(continued from previous page)

```
All PODs exited cleanly.
Output written to <OUTPUT_DIR>/MDTF_QBOi.EXP1.AMIP.001_1977_1981
```

This shows that the output of the package has been saved to a directory named `MDTF_QBOi.EXP1.AMIP.001_1977_1981` in <OUTPUT_DIR>. The results are presented as a series of web pages, with the top-level page named index.html. To view the results in a web browser, run (e.g.,)

```
% google-chrome <OUTPUT_DIR>/MDTF_QBOi.EXP1.AMIP.001_1977_1981/index.html &
```

Currently the framework only analyzes data from one model run at a time. To run another test for the the MJO Propagation and Amplitude POD on the sample data from GFDL's CM4 model, open the configuration file at <config_file_path>, delete or comment out the section for `QBOi.EXP1.AMIP.001` in the `caselist` section of that file, and uncomment the section for `GFDL.CM4.c96L32.am4g10r8`.

In Running the package on your data (page 10), we describe further options to customize how the package is run.

1.3 Running the package on your data

In this section we describe how to proceed beyond running the simple test case described in the previous section (page 3), in particular how to run the framework on your own model data.

1.3.1 Preparing your data for use by the package

You have multiple options for organizing or setting up access to your model's data in a way that the framework can recognize. This task is performed by a "data source," a code plug-in that handles obtaining model data from a remote location for analysis by the PODs.

For a list of the available data sources, what types of model data they provide and how to configure them, see the data source reference. In the rest of this section, we describe the steps required to add your own model data for use with the LocalFile data source, since it's currently the most general-purpose option.

Selecting and formatting the model data

Consult the POD summary to identify which diagnostics you want to run and what variables are required as input for each. In general, if the data source can't find data that's required by a POD, an error message will be logged in place of that POD's output that should help you diagnose the problem.

The LocalFile data source works with model data structured with each variable stored in a separate netCDF file. Some additional conditions on the metadata are required: any model output compliant with the CF conventions[18] is acceptable, but only a small subset of those conventions are required by this data source. See the data format reference for a complete description of what's required.

[18] http://cfconventions.org/

Naming variables according to a convention

The LocalFile data source is intended to deal with output produced by different models, which poses a problem because different models use different variable names for the same physical quantity. For example, in NCAR's CESM2[19] the name for total precipitation is `PRECT`, while the name for the same quantity in GFDL's CM4[20] is `precip`.

In order to identify what variable names correspond to the physical quantities requested by each POD, the LocalFile data source requires that model data follow one of several recognized variable naming conventions defined by the package. The currently recognized conventions are:

- `CMIP`: Variable names and units as used in the CMIP6[21] data request[22]. There is a web interface[23] to the request. Data from any model that has been published as part of CMIP6 (e.g., made available via ESGF[24]) should follow this convention.
- `NCAR`: Variable names and units used in the default output of models developed at the National Center for Atmospheric Research[25], such as CAM[26] (all versions) and CESM2[27].
- `GFDL`: Variable names and units used in the default output of models developed at the Geophysical Fluid Dynamics Laboratory[28], such as AM4[29], CM4[30] and SPEAR[31].

The names and units for the variables in the model data you're adding need to conform to one of the above conventions in order to be recognized by the LocalFile data source. For models that aren't currently supported, the workaround we recommend is to generate `CMIP`-compliant data by postprocessing model output with the CMOR[32] tool. We hope to offer support for the naming conventions of a wider range of models in the future.

Adding your model data files

The LocalFile data source reads files from a local directory that follow the filename convention used for the sample model data. Specifically, the files should be placed in a subdirectory in <MODEL_DATA_ROOT> and named following the pattern

<MODEL_DATA_ROOT>/<dataset_name>/<frequency>/<dataset_name>.<variable_name>.<frequency>.nc,

where

- <MODEL_DATA_ROOT> is the path where the sample model data was installed (see Configuring framework paths (page 8)),

[19] https://www.cesm.ucar.edu/models/cesm2/
[20] https://www.gfdl.noaa.gov/coupled-physical-model-cm4/
[21] https://www.wcrp-climate.org/wgcm-cmip/wgcm-cmip6
[22] https://doi.org/10.5194/gmd-2019-219
[23] http://clipc-services.ceda.ac.uk/dreq/index.html
[24] https://esgf-node.llnl.gov/projects/cmip6/
[25] https://ncar.ucar.edu
[26] https://www.cesm.ucar.edu/models/cesm2/atmosphere/
[27] https://www.cesm.ucar.edu/models/cesm2/
[28] https://www.gfdl.noaa.gov/
[29] https://www.gfdl.noaa.gov/am4/
[30] https://www.gfdl.noaa.gov/coupled-physical-model-cm4/
[31] https://www.gfdl.noaa.gov/spear/
[32] https://cmor.llnl.gov/

- <dataset_name> is any string uniquely identifying the dataset,
- <frequency> is a string describing the frequency at which the data is sampled, e.g. `1hr`, `3hr`, `6hr`, `day`, `mon` or `year`.
- <variable_name> is the name of the variable in the convention chosen in the previous section.

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 )
```

Note that the `GFDL.CM4.c96L32.am4g10r8` dataset uses the `GFDL` convention (precipitation = `precip`), while the `QBOi.EXP1.AMIP.001` dataset uses the `NCAR` convention (precipitation = `PRECT`).

If the data you want to analyze is available on a locally mounted disk, we recommend creating symlinks[33] that have the needed filenames, rather than making copies of the data files. For example,

```
% mkdir -p inputdata/model/my_dataset/day
% ln -s <path> inputdata/model/my_dataset/day/my_dataset.pr.day.nc
```

will create a symbolic link to the file at <path> that follows the filename convention used by this data source:

```
inputdata
  model ( = <MODEL_DATA_ROOT>)
      GFDL.CM4.c96L32.am4g10r8
      QBOi.EXP1.AMIP.001
      my_dataset
          day
              my_dataset.pr.day.nc
```

Finally, we note that it's not necessary to place the files (or symlinks) for all experiments in <MODEL_DATA_ROOT>. To point the LocalFile data source to data stored in the subdirectory hierarchy following the pattern described above, but located in a different place, pass that location to the package as <CASE_ROOT_DIR>.

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

1.3.2 Running the package on your data

How to configure the package

All configuration options for the package are set via its command line interface, which is described in ref_cli, or by running `% mdtf --help`. Because it's cumbersome to deal with long lists of command-line flags, options can also be set in a JSON configuration file passed to the package with the `-f`/`--input-file` flag. An example of this input file is given in src/default_tests.jsonc[34], which you used previously (page 9) to run the package on test data. We recommend using this file as a template, making copies and customizing it as needed.

Option values given on the command line always take precedence over those set in the configuration file. This is so that you can store options that don't frequently change in the file (e.g., input/output paths) and then use flags to set only those options you want to change from run to run (e.g., the start and end years for the analysis). In all cases, the complete set of option values used in each run of the package is saved as a JSON configuration file in the package's output, so you can always reproduce your results.

Options controlling the analysis

The configuration options required to specify what analysis the package should do are:

- `--CASENAME` <name>: Identifier used to label this run of the package. Can be set to any string.
- `--experiment` <dataset_name>: The name (subdirectory) you assigned to the data files in the previous section. If this option isn't given, its value is set from <CASENAME>.
- `--convention` <convention name>: The naming convention used to assign the <variable_name>s, from the previous section.
- `--FIRSTYR` <YYYY>: The starting year of the analysis period.
- `--LASTYR` <YYYY>: The end year of the analysis period. The analysis period includes all data that falls between the start of 1 Jan on <FIRSTYR> and the end of 31 Dec on <LASTYR>. An error will be raised if the data provided for any requested variable doesn't span this date range.

If specifying these in a configuration file, these options should given as entry in a list titled `case_list` (following the example in src/default_tests.jsonc[35]). Using the package to compare the results of a list of different experiments is a major feature planned for an upcoming release.

You will also need to specify the list of diagnostics to run. This can be given as a list of POD names (as given in the diagnostics/[36] directory), or `all` to run all PODs. This list can be given by the `--pods` command-line flag, or by a `pod_list` attribute in the `case_list` entry.

[34] https://github.com/NOAA-GFDL/MDTF-diagnostics/blob/main/src/default_tests.jsonc
[35] https://github.com/NOAA-GFDL/MDTF-diagnostics/blob/main/src/default_tests.jsonc
[36] https://github.com/tsjackson-noaa/MDTF-diagnostics/tree/main/diagnostics

Other options

Some of the most relevant options which control the package's output are:

- `--save-ps`: Set this flag to have PODs save copies of all plots as postscript files (vector graphics) in addition to the bitmaps used in the HTML output pages.
- `--save-nc`: Set this flag to have PODs retain netCDF files of any intermediate calculations, which may be useful if you want to do further analyses with your own tools.
- `--make-variab-tar`: Set this flag to save the collection of files (HTML pages and bitmap graphics) output by the package as a single .tar file, which can be useful for archival purposes.

The full list of configuration options is given at ref_cli.

Running the package

From this point, the instructions for running the package are the same as for running it on the sample data (page 9), assuming you've set the configuration options by editing a copy of the configuration file template at src/default_tests.jsonc[37]. The package is run in the same way:

```
% cd <CODE_ROOT>
% ./mdtf -f <new config file path>
```

The first few lines of console output will echo the values you've provided for <CASENAME>, etc., as confirmation.

The output of the package will be saved as a series of web pages in a directory named MDTF_<CASENAME>_<FIRSTYR>_<LASTYR> within <OUTPUT_DIR>. If you run the package multiple times with the same configuration values, it's not necessary to change the <CASENAME>: by default, the suffixes ".v1", ".v2", etc. will be added to duplicate output directory names so that results aren't accidentally overwritten.

The results of the diagnostics are presented as a series of web pages, with the top-level page named index.html. To view the results in a web browser, run (e.g.,)

```
% google-chrome <OUTPUT_DIR>/MDTF_<CASENAME>_<FIRSTYR>_<LASTYR>/index.html &
```

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

CHAPTER

TWO

SITE-SPECIFIC DOCUMENTATION

2.1 Customizing your installation with the 'local' site

2.1.1 About the 'local' site

The `--site` command-line flag is used to implement plug-in functionality for site-specific code, (eg, enabling data search from a lab's internally-accessible filesystem.) The default value for this flag is `local`: code and configuration files placed in the sites/local/ directory will be used to customize the general-purpose framework code in src/.

The most important use case for this functionality is allowing you to set default values for command-line flags, as described in the next section. This lets you set configuration options that are the same for each run (e.g., <OBS_DATA_ROOT>, the path to your local copy of the observational data used by the diagnostics) once, in a file in this directory, without having to remember to include the corresponding command-line flag every time you run the package.

This function only sets default values: any value may be overridden for any run of the package by specifying it explicitly on the command line (or in an input file). Regardless of where they originate, the complete list of configuration settings used in a run of the package is saved in the output, so you can always recreate a run of the package even if you change these defaults.

The full API for the `--site` functionality, as well as instructions on how to develop your own site-specific data sources and other code plug-ins, will be documented in an upcoming release.

2.1.2 How to set default values

When run, the framework looks for a file named `defaults.jsonc` in the directory for the chosen site. An example of the format for this file is provided in src/default_tests.jsonc[38], which we encourage you to make a copy of, rename, and edit.

More specifically, the `defaults.jsonc` file should be a JSON file (with `//`-comments allowed) listing <key>:<value> pairs. The <key> should be the long name of one of the command-line flags, with hyphens replaced by underscores. The <value> should be the desired default value. <key>s which don't correspond to recognized command-line flags, such as the `caselist` in src/default_tests.jsonc[39], are ignored.

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

You can test the settings in your `defaults.jsonc` file by running `% mdtf --help`. The beginning of the help text will list the path to the default settings files being used, and the help text for each command-line flag will note its default value.

2.1.3 Switching between multiple defaults with multiple sites

A site is simply a subdirectory of the `sites/` directory. You can manage and easily switch between multiple sets of default values by creating additional subdirectories within `sites/`, along with a `defaults.jsonc` file for each, and selecting one at runtime with the `--site` flag.

This can be useful if you frequently need to analyze data from a variety of different data sources: you can create one site per data source, and add the settings specifying the desired data set from that source at runtime.

2.2 GFDL-specific information

This page contains information specific to the site installation at the Geophysical Fluid Dynamics Laboratory[40] (GFDL), Princeton, NJ, USA.

2.2.1 Site installation

The DET team maintains a site-wide installation of the framework and all supporting data at /home/oar.gfdl.mdtf/mdtf/MDTF-diagnostics. This is kept up-to-date and is accessible from both workstations and PPAN; in particular it is not necessary for an end user to set up conda environments or download any supporting data, as described in the installation instructions.

Invoking the package from the site installation's wrapper script automatically prepends `--site="NOAA_GFDL"` to the user's command-line flags.

Please contact us if your use case can't be accommodated by this installation.

2.2.2 Additional ways to invoke the package

The site installation provides alternative ways to run the diagnostics within GFDL's existing workflow:

1. Called from an interactive shell on PPAN or workstations. This is the standard mode of running the package, described in the rest of the documentation.
2. As a batch job on PPAN, managed via slurm. This previously required its own wrapper script, but now can be done using the same entry point and CLI options as for interactive execution.
3. Within FRE XMLs. This is done by calling the mdtf_gfdl.csh[41] wrapper script from an `<analysis>` tag in the XML:

```
<analysis switch="on" cumulative="no" script="/home/oar.gfdl.mdtf/mdtf/MDTF-
↪diagnostics/sites/NOAA_GFDL/mdtf_gfdl.csh"/>
```

[40] https://www.gfdl.noaa.gov/
[41] https://github.com/NOAA-GFDL/MDTF-diagnostics/blob/main/sites/NOAA_GFDL/mdtf_gfdl.csh

The MDTF package behaves as any other analysis script called by FRE from an experiment XML: FRE will populate the wrapper script with the correct paths, date range of the run, etc., so these options don't need to be passed in the XML tag.

The wrapper script calls the site installation of the package with the `--data-manager="GFDL_PP"` (see below) option. `GFDL_PP` defaults to assuming GFDL variable naming conventions; data which follows other conventions (e.g. fremetarized runs intended for publication as part of CMIP6) requires the `--convention` flag to be set explicitly. In general, the wrapper script passes through any additional options set in the tag's `script` attribute, in addition to setting the data attributes provided by FRE. Passing through package flags in the `<analysis>` tag can be used to, e.g., only run specific PODs for each `<component>` with the `--pods` option.

Currently, FRE requires that each analysis script be associated with a single model `<component>`. This poses difficulties for the MDTF package, which analyzes data from multiple modeling realms/`<component>`s. We provide two ways to address this issue:

A. If it's known ahead of time that a given model `<component>` will dominate the run time and finish last, one can call `mdtf_gfdl.csh --run_once` 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 runs. The `<component>` being used doesn't need to generate any 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, an alternate solution is to call `mdtf_gfdl.csh` from each `<component>` that generates data to be analyzed. This is assumed to be the default use case for the wrapper script (when `--run_once` is not set), where the package is called multiple times on a single model run to incrementally update the analysis as data from different components finishes postprocessing. Every time the package 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`).

In case 3A or 3B, you can optionally pass the `--component_only` flag to the wrapper script if you wish to restrict the package to only use data from the `<component>` it's associated with in the XML. Otherwise, the default behavior is for the package to search all the data that's present in the /pp/ directory hierarchy when it runs.

The `--run_once` flag should be used whenever you don't need the incremental update capability of case 3B (or if the package is only being called from one `<component>` in an XML), since the default behavior in 3B necessarily disables logging warnings if individual PODs aren't able to run.

2.2.3 Additional data sources

In addition to the framework's built-in data sources, several data sources are defined that are only accessible to GFDL users.

All the data sources in this section use GFDL's in-house General Copy Program (GCP, not to be confused with Google Compute Platform) for all file transfers. If GCP is not present on `$PATH` when the package is started, the package will load the appropriate environment module.

Any data which is on GFDL's DMF tape-backed filesystem will be requested with `dmget` prior to copy. All files requested by all PODs are batched into a single call to `dmget` and to GCP. Framework execution blocks

after the call to `dmget` is issued (the framework has no other tasks to do until the data is transferred locally), which can lead to long or unpredictable run times if data that has been migrated to tape is requested.

CMIP6 data on the Unified Data Archive

Selected via `--data-manager="CMIP6_UDA"`.

Data source for analyzing CMIP6 data made available on on the Unified Data Archive (UDA)'s high-priority storage at /uda/CMIP6. Command-line options and method of operation are the same as documented in ref-data-source-cmip6.

CMIP6 data on the /archive filesystem

Selected via `--data-manager="CMIP6_archive"`.

The same as above, but for analyzing the wider range of CMIP6 data on the DMF filesystem at /archive/pcmdi/repo/CMIP6. Command-line options and method of operation are the same as documented in ref-data-source-cmip6.

CMIP6 data on /data_cmip6

Selected via `--data-manager="CMIP6_data_cmip6"`.

The same as above, but for analyzing pre-publication data on /data_cmip6/CMIP6 (only mounted on PPAN). Command-line options and method of operation are the same as documented in ref-data-source-cmip6.

Results of FREPP-processed runs

Selected via `--data-manager="GFDL_PP"`.

This data source searches for model data produced using GFDL's in-house postprocessing tool, FREPP. Note that this is a completely separate concern from invoking the package from the FRE pipeline (described above): data that has been processed and saved in this convention can be analyzed equally well in any of the package's modes of operation.

Command-line options

<CASE_ROOT_DIR> should be set to the root of the postprocessing directory hierarchy (i.e., should end in `/pp`).

--component
: If set, only run the package on data from the specified model component name. If this flag is not set, the data source will return data from different model `<component>`s requested by the same POD; see the description of the heuristics used for `<component>` selection below. This is necessary for, e.g., PODs that compare data from different modeling realms. The main use case for this flag is passing options from FRE to the package via the wrapper script.

--chunk_freq	If set, only run the package on data with the specified timeseries chunk length. If not set, default behavior is to use the smallest chunks available. The main use case for this flag is passing options from FRE to the package via the wrapper script.

When using this data source, `-c`/`--convention` should be set to the convention used to assign variable names. If not given, `--convention` defaults to `GFDL`.

Data selection heuristics

This data source implements the following logic to guarantee that all data it provides to the PODs are consistent, i.e. that the variables selected have been generated from the same run of the same model. An error will be raised if no set of variables can be found that satisfy the user's input above and the following requirements:

- This data source only searches data saved as time series (`/ts/`), rather than time averages, since no POD is currently designed to use time-averaged data.
- If the same data has been saved in files of varying chronological length (`<chunk_freq>`), the shortest `<chunk_freq>` is used, in order to minimize the amount of data that is transferred but not used (because it falls outside of the user's analysis period).
- By default, any variable can come from model `<component>`, with the same component used for all variables requested by a POD if possible. This setting is required to enable the execution of PODs that use data from different `<component>`s or realms.
  - Specifying a model component with the `--component` flag does one of two things, depending on whether the package is being run once or incrementally.
  - If the package is being run once, all data used must come from that component (e.g., multi-realm PODs will not run). In this case we assume the user wants to focus their attention on this component exclusively.
  - If the package is being run incrementally (called from FRE without the `--run_once` flag, see above, or called in general with the `--frepp` flag), all data for each POD must come from the same component, but different PODs may use data from different components. This is because we're operating according to scenario 3B (above) and are analyzing multiple components, but still want to focus on component-specific diagnostics.
- If the same data is provided by multiple model `<component>`s, a single `<component>` is selected via the following heuristics:
  - Preference is given to model components starting with "cmip" (case insensitive), in order to support analysis of data produced as part of CMIP6.
  - If multiple `<component>`s are still eligible, the one with the fewest words in the identifier (separated by underscores) is selected; in case of a tie, the `<component>` name with the shortest overall string length is used.
  - This is haphazard, but it's the best we can do given that `<component>` names may be arbitrary strings, with only partial standardization.

Quasi-automated source selection

Selected via `--data-manager="GFDL_auto"`.

Provided mostly for backwards compatibility, this dispatches operation to the `CMIP6_UDA` or `GFDL_PP` data sources based on whether <CASE_ROOT_DIR> is a valid postprocessing directory. Command-line options are the union of those for the `CMIP6_UDA` or `GFDL_PP` data sources.

2.2.4 Additional command-line options

In addition to the framework's built-in command-line options, the following site-specific options are recognized.

For long command line flags, words may be separated with hyphens (GNU standard) or with underscores (python variable name convention). For example, `--file-transfer-timeout` and `--file_transfer_timeout` are both recognized by the package as synonyms for the same setting.

GFDL-specific flags

The following new flags are added:

--GFDL-PPAN-TEMP <DIR> If running on the GFDL PPAN cluster, set the `$MDTF_TMPDIR` environment variable to this location and create temp files here. This must be a location accessible via GCP, and the package does not currently verify this. Defaults to `$TMPDIR`.

--GFDL-WS-TEMP <DIR> If running on a GFDL workstation, set the `$MDTF_TMPDIR` environment variable to this location and create temp files here. The directory will be created if it doesn't exist. This must be accessible via GCP, and the package does not currently verify this. Defaults to /net2/`$USER`/tmp.

--frepp Normally this is set by the mdtf_gfdl.csh[42] wrapper script (by default, unless the `--run_once` flag is set), and not directly by the user. This should only be set if you're using the package in scenario 3B. above, where the package will be called multiple times when each model component is finished running. When the package is invoked with this flag, it only runs PODs for which i) the data has finished post-processing (is present in the /pp/ directory) and ii) haven't been run by a previous invocation of the package. The bookkeeping for this is done by having each invocation write placeholder directories for each POD it's executing to `$OUTPUT_DIR`. Setting this flag disables the package's warnings for PODs with missing data, since that may be a normal occurrence in this scenario.

[42] https://github.com/NOAA-GFDL/MDTF-diagnostics/blob/main/sites/NOAA_GFDL/mdtf_gfdl.csh

GFDL-specific default values

The following paths are set to more useful default values:

--OBS-DATA-REMOTE <DIR> Site-specific installation of observational data used by individual PODs at /home/oar.gfdl.mdtf/mdtf/inputdata/obs_data. If running on PPAN, this data will be GCP'ed to the current node. If running on a workstation, it will be symlinked.

--OBS-DATA-ROOT <OBS_DATA_ROOT> Local directory for observational data. Defaults to `$MDTF_TMPDIR`/inputdata/obs_data, where the environment variable `$MDTF_TMPDIR` is defined as described above.

--MODEL-DATA-ROOT <MODEL_DATA_ROOT> Local directory used as a destination for downloaded model data. Defaults to `$MDTF_TMPDIR`/inputdata/model, where the environment variable `$MDTF_TMPDIR` is defined as described above.

--WORKING-DIR <WORKING_DIR> Working directory. Defaults to `$MDTF_TMPDIR`/wkdir, where the environment variable `$MDTF_TMPDIR` is defined as described above.

-o, --OUTPUT-DIR <OUTPUT_DIR> Destination for output files. Defaults to `$MDTF_TMPDIR`/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[43] (NOAA) Climate Program Office Modeling, Analysis, Predictions and Projections[44] (MAPP) Program (grant # NA18OAR4310280). Additional support was provided by University of California Los Angeles[45], the Geophysical Fluid Dynamics Laboratory[46], the National Center for Atmospheric Research[47], Colorado State University[48], Lawrence Livermore National Laboratory[49] and the US Department of Energy[50].

Many of the process-oriented diagnostics modules (PODs) were contributed by members of the NOAA Model Diagnostics Task Force[51] 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.

[43] https://www.noaa.gov/
[44] https://cpo.noaa.gov/Meet-the-Divisions/Earth-System-Science-and-Modeling/MAPP
[45] https://www.ucla.edu/
[46] https://www.gfdl.noaa.gov/
[47] https://ncar.ucar.edu/
[48] https://www.colostate.edu/
[49] https://www.llnl.gov/
[50] https://www.energy.gov/
[51] https://cpo.noaa.gov/Meet-the-Divisions/Earth-System-Science-and-Modeling/MAPP/MAPP-Task-Forces/Model-Diagnostics-Task-Force